EUROPEAN COMMISSION

Community Research

SEVENTH FRAMEWORK PROGRAMME

Private Public Partnership Project (PPP)

Large-scale Integrated Project (IP)

fi-ware

D.2.3.1b: FI-WARE Architecture

Project acronym: FI-WARE

Project full title: Future Internet Core Platform

Contract No.: 285248

Strategic Objective: FI.ICT-2011.1.7 Technology foundation: Future Internet Core Platform

Project Document Number: ICT-2011-FI-285248-WP2-D.2.3.1b

Project Document Date: 2012-10-29

Deliverable Type and Security: Public

Author: FI-WARE Consortium

Contributors: FI-WARE Consortium

Table of Contents

1 Introduction

1.1 Executive Summary

This resubmitted version of the deliverable "FI-WARE Architecture" outlines updated information for the FI-WARE project.

A description of the Reference Architecture linked to the different chapters of FI-WARE is given in detail. A description of FI-WARE Generic Enablers (GEs) being supported in each chapter is provided, including the description of the APIs that each FI-WARE Generic Enabler (GE) exposes to application developers or it uses to connect to another FI-WARE GEs.

The given content was improved significantly and additional information requested by the reviewers were adhere to.

Parts of this document is being reused within the Open Specifications and help to distribute the information on detailed design and architecture to the general public and make the Open Specifications more self-contained.

1.2 About This Document

The FI-WARE Architecture describes the GEs of FI-WARE being implemented, interfaces between them and properties of both. It also brings a description of the architecture of components linked to the implementation of each of the FI-WARE GEs being developed within the FI-WARE project.

Application developers reading contents of this document should be able to understand how applications are programmed using APIs exposed by FI-WARE GEs (i.e., what is the programming model). They will learn what are the names, basic description of arguments, behavior and responses of the main operations in those APIs.

However, the detailed specification of FI-WARE APIs are outlined in the FI-WARE GE Open Specifications and related deliverables.

1.3 Intended Audience

The architecture description of components is given with new developers in mind, who join the development team linked to that component to make it effectively. Additionally service providers can learn about capabilities and functionality the FI-WARE platform provides. Thirdly use case projects and related colleagues might use the provided information to understand FI-WARE design and adapt their own architecture or adopt certain parts of the system.

1.4 Chapter Context WP2 Global Technical Activities

Due to the complexity and scope of FI-WARE’s vision and objectives project-level coordination and direction of some key technical activities is critical for the success of the project. The transversal work package (WP2) "Global Technical Activities" focuses on global technical activities such as analysis and consolidation of requirements requested by FI-

WARE's stakeholders, high level architecture, Core Platform roadmap, open calls coordination.

All these technical coordination activities will be performed in close cooperation with all the research and development work packages (WP3-WP8). Specific WP2 resources assigned to representatives of the research and development WPs (WP3-WP8) will allow the execution of these coordination activities and the generation of the global consolidated results, both driving and drawing upon the expertise and resources of the development WPs.

Requirements specification will be a key activity in constructing the FI-WARE platform and revision/refinement of requirement specifications will take place continuously through all the sprints planned during development of reference implementations of FI-WARE Generic Enablers (GEs).

The realization of the FI-WARE vision and objectives through the concept of Generic Enablers will require a carefully designed global architecture where the role of each GE and the interfaces between GEs as well as the interfaces exported by GEs to the users of FI-WARE are properly defined.

In order to fulfil the FI-WARE promise all Generic Enablers will be accompanied by Open Specifications that will facilitate usage and integration in any FI-WARE Instance as well as the development of compliant implementations of GEs by third parties. While GE Specifications themselves will be done inside WP3-WP10, overall coordination and consolidation of these specifications is part of this work package.

1.5 Structure of this Document

The document is generated out of a set of documents provided in the public FI-WARE wiki. For the current version of the documents, please visit the public wiki at http://wiki.fi-ware.eu/

The following resources were used to generate this document:

- FI-WARE Architecture
- Cloud Hosting Architecture
- FIWARE.ArchitectureDescription.Cloud.DCRM
- FIWARE.ArchitectureDescription.Cloud.SM
- FIWARE.ArchitectureDescription.Cloud.CloudEdge
- FIWARE.ArchitectureDescription.Cloud.ObjectStorage
- Data/Context Management Architecture
- FIWARE.ArchitectureDescription.Data.BigData
- FIWARE.ArchitectureDescription.Data.PubSub
- FIWARE.ArchitectureDescription.Data.CEP
- FIWARE.ArchitectureDescription.Data.Location
- FIWARE.ArchitectureDescription.Data.MetadataPreprocessing
- FIWARE.ArchitectureDescription.Data.CompressedDomainVideoAnalysis
- FIWARE.ArchitectureDescription.Data.QueryBroker
- FIWARE.ArchitectureDescription.Data.SemanticAnnotation
- FIWARE.ArchitectureDescription.Data.SemanticSupport

1.6 Typographical Conventions

Starting with October 2012 the FI-WARE project improved the quality and streamlined the submission process for deliverables, generated out of the public and private FI-WARE wiki. The project is currently working on the migration of as many deliverables as possible towards the new system.

This document is rendered with semi-automatic scripts out of a MediaWiki system operated by the FI-WARE consortium.

1.6.1 Links within this document

The links within this document point towards the wiki where the content was rendered from. You can browse these links in order to find the "current" status of the particular content.

Due to technical reasons not all pages that are part of this document can be linked document-local within the final document. For example, if an open specification references and "links" an API specification within the page text, you will find this link firstly pointing to the wiki, although the same content is usually integrated within the same submission as well.

1.6.2 Figures

Figures are mainly inserted within the wiki as the following one:

```
[[Image:....|size|alignment|Caption]]
```

Only if the wiki-page uses this format, the related caption is applied on the printed document. As currently this format is not used consistently within the wiki, please understand that the rendered pages have different caption layouts and different caption formats in general. Due to technical reasons the caption can't be numbered automatically.

1.6.3 Sample software code

Sample API-calls may be inserted like the following one.

```
http://[SERVER_URL]?filter=name:Simth*&index=20&limit=10
```

1.7 Acknowledgements

The current document has been elaborated using a number of collaborative tools, with the participation of Working Package Leaders and Architects as well as those partners in their teams they have decided to involve.

1.8 Keyword list

FI-WARE, PPP, Architecture Board, Steering Board, Roadmap, Reference Architecture, Generic Enabler, Open Specifications, I2ND, Cloud, IoT, Data/Context Management, Applications/Services Ecosystem, Delivery Framework , Security, Developers Community and Tools , ICT, es.Internet, Latin American Platforms, Cloud Edge, Cloud Proxy.

1.9 Changes History

Release	Major changes description	Date	Editor
v0	First draft of deliverable submission generated	2012-10-29	SAP, TID
v1	Recompilation for internal Review	2012-11-02	SAP, TID
v2	Final review	2012-11-07	TID
V3	Submission	2012-11-17	TID

2 FI-WARE Architecture

You can find the content of this chapter as well in the wiki of fi-ware.

Following is a description of the Reference Architecture linked to the different chapters of FI-WARE. A description of FI-WARE Generic Enablers (GEs) being supported in each chapter is provided, including the high-level description of the APIs that each FI-WARE Generic Enabler (GE) exposes to application developers or it uses to connect to another FI-WARE GEs.

- Cloud Hosting
- Data/Context Management
- Internet of Things (IoT) Services Enablement
- Applications/Services Ecosystem and Delivery Framework
- Security
- Interface to Networks and Devices (I2ND)

The Reference Architecture associated to each FI-WARE chapter can be instantiated into a concrete architecture by means of selecting an integrating products implementing the corresponding FI-WARE GEs (i.e., products which are compliant with the corresponding FI-WARE GE Open Specifications). However, the description of the Reference Architecture associated to a chapter does not depend on how FI-WARE GEs in that chapter can be implemented. Any implementation of a FI-WARE GE will be, by nature, replaceable.

Application developers reading contents of this document will understand how applications are programmed using APIs exposed by FI-WARE GEs (i.e., what is the programming model). They will learn what are the names, basic description of arguments, behavior and responses of the main operations in those APIs. On the other hand, FI-WARE Instance Providers will learn how FI-WARE GEs can be connected to build FI-WARE Instances.

Complementing description of the FI-WARE Reference Architecture you can visit the Summary of FI-WARE Open Specifications to learn the rest of details about FI-WARE GE Open Specifications, particularly detailed API Open Specifications. Note that FI-WARE GE Open Specifications will be public and royalty-free.

Developer Community and Tools (DCT) wants to offer a comprehensive environment enabling the FI-PPP program developers (and others) to use in the more efficient, easy and effective way the FI-WARE outcomes (i.e. GE Implementations and FI-WARE Instances) and exploiting collaboration means to benefit the support of the community. For more details visit the following link.

- Developer Community and Tools

3 Cloud Hosting Architecture

You can find the content of this chapter as well in the wiki of fi-ware.

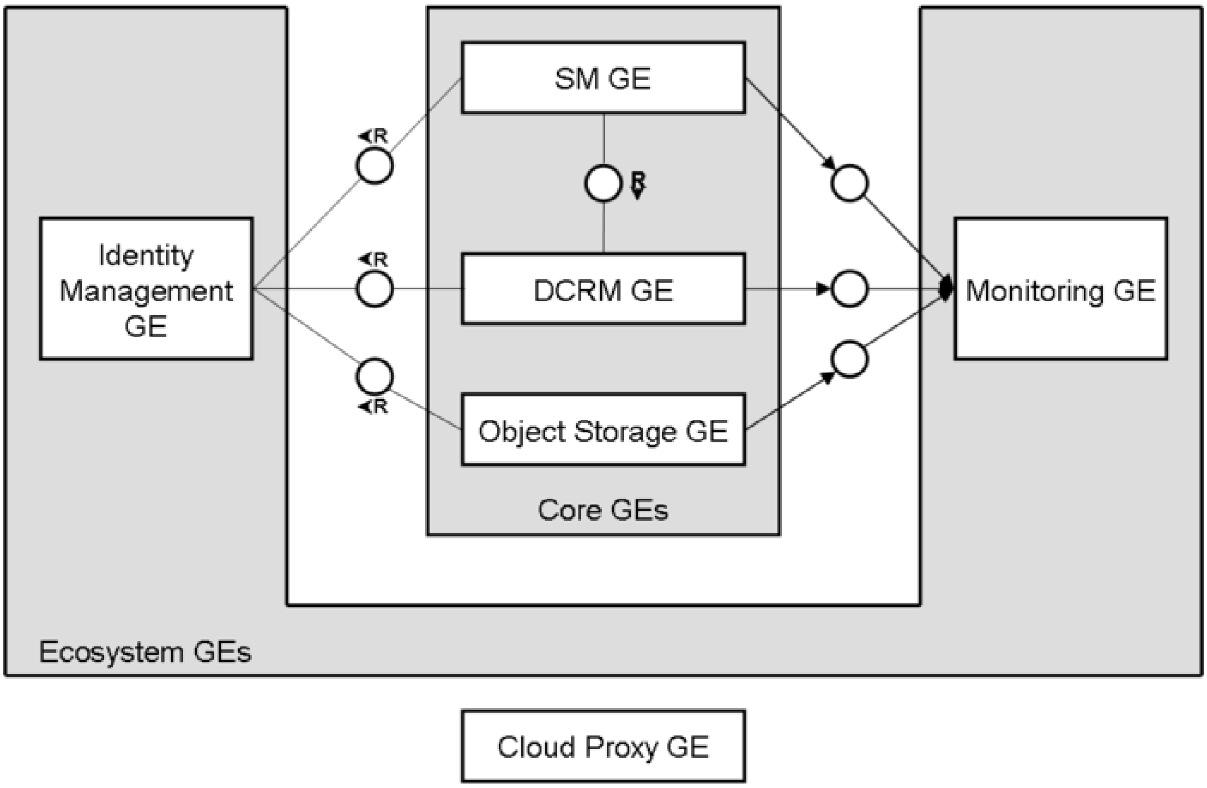
3.1.1 Introduction

The Cloud Chapter offers Generic Enablers that comprise the foundation for designing a modern cloud hosting infrastructure that can be used to develop, deploy and manage Future Internet applications and services, as outlined in Materializing Cloud Hosting in FI-WARE.

In a nutshell, in the first release of FI-WARE, the solution focuses on fundamental cloud capabilities enabling provisioning and life cycle management of virtual machines and associated resources (compute, storage, network, images, etc) hosting FI applications and services, as well as object storage capabilities which can be used directly by FI applications and services via a REST API. In future releases, additional capabilities will be added, notably the support for complex services comprising multiple virtual machines, including monitoring, policy-based elasticity, as well as many others, as outlines in Roadmap of Cloud Hosting.

3.1.2 Architecture Overview

The following diagram shows the main components (Generic Enablers) that comprise the first release of FI-WARE architecture.



The GEs in the above diagram are grouped into **Core GEs**, providing the core hosting capabilities at different abstraction levels (resources, services, objects, etc) and **Ecosystem GEs**, addressing various specific needs across the Core GEs, and establishing the ecosystem that enables the end-to-end capabilities provided by a cloud offering.

The **Core GEs** include:

- **Data Center Resource Maangement (DCRM) GE**, offering provisioning and life cycle management of virtualized resources (compute, storage, network) associated with virtual machines.
- **Object Storage GE**, offering provisioning and life cycle management of object-based storage containers and elements
- **Service Management (SM) GE**, offering provisioning and life cycle management of composite services comprising several resources provided by on of the above GEs. In the first release of FI-WARE, Service Management GE will consume resources provided by Data Center Resource Management GE, via the corresponding APIs.

In the next releases of FI-WARE, the following additional Core GEs will be considered:

- **Cloud Edge Resource Management GE**, enabling end-to-end provisioning and life cycle management of cloud applications which comprise run-time components designed to run on Cloud Edge devices. The hosting capabilities on such Cloud Edge devices are provided by **Cloud Proxy GE**, developed across Cloud Chapter and I2ND Chapter.
- **PaaS Management GE**, offering provisioning and life cycle management of middleware-level containers, such as Web, Database, etc.

The **Ecosystem GEs** include:

- **Monitoring GE**, collecting metrics associated with each of the Core GEs, and offering them to GEs which are interested to consume such metrics. For example, Service Management GE consumes metrics associated with KPIs of the various service components in order to drive auto-scaling decisions. In the future, more advanced metrics-related capabilities will be provided, such as processing (before it is delivered to the consumer), archival and analysis of metrics.
- **Identity Management GE**, providing a unified management of users, roles and tokens, that can be used by other GEs for authentication and authorization purposes. This GE will be provided by the Security Chapter.

In the next releases of FI-WARE, the following additional Ecosystem GEs will be considered:

- **Accounting GE**, allowing to keep records about resource usage, which can be then used for billing purposes
- **Audit GE**, allowing to keep track of all the activities in the cloud for auditing purposes
- **Cloud Broker GE**, allowing to leverage GEs deployed in another Cloud in a federated or hybrid fashion

3.1.2.1 *Inter-dependencies and Interaction Between GEs*

While each GE comprises a set of functions and capabilities that can be used stand-alone, in a typical cloud deployment the different GEs will interact with each other, to provide a complete end-to-end solution. In particular, each of the 'Ecosystem GEs' is expected to interact with the Core GEs, via their APIs, as outlined above. To summarize, the following interaction between GEs is expected:

- Service Management (SM) GE will invoke APIs of Data Center Resource Management (DCRM) GE to perform operations on actual virtualized resources (mainly virtual machines) which comprise the services managed by SM GE. In future releases of FI-WARE, SM GE will offer services that comprise additional types of resources, and hence will also interact with additional Core GEs, such as Cloud Edge Resource Management GE and Object Storage GE.
- All the GEs will use Identity Management GE APIs for authentication and authorization purposes
- Core GEs will publish metrics via the Monitoring GE, GEs will subscribe to metrics provided by Monitoring GE

3.1.2.2 *Guiding Design Principles*

There has been quite an amount of work carried out in defining what cloud computing is and what design principles it should adhere to. There are a number of general principles that have been collated by the Future Internet Architecture Group and some of these are especially relevant. One of the canonical definitions of cloud computing, more so from a technology perspective, is that of the National Institute for Standards and Technologies (NIST).

On-Demand, Self-Service.

A consumer can unilaterally provision computing capabilities, such as server time and network storage, automatically as needed without requiring manual human interaction with each service provider.

Broad Network Access.

Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin- or thick-client platforms (e.g., mobile phones, tablets, laptops, and workstations).

Resource Pooling.

The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources, but may be able to specify location at a higher level of abstraction (e.g., country, state, or data centre). Examples of resources include storage, processing, memory and network bandwidth.

Rapid Elasticity.

Capabilities can be quickly elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

Measured Service / Pay-As-You-Go.

Cloud systems automatically control and optimise resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported. This provides transparency for both the provider and consumer of the utilised service. This finally allows for metered services pricing, also known as pay-as-you-go.

Dependability

To be dependable, a service must exhibit/implement the following attributes as defined by Avizienis:

- **Availability**: readiness for correct service
- **Reliability**: continuity of correct service
- **Safety**: absence of catastrophic consequences on the user(s) and the environment
- **Integrity**: absence of improper system alteration
- **Maintainability**: ability for a process to undergo modifications and repairs
- **Confidentiality**: absence of unauthorised disclosure of information

This definition was formed in 2004, however in the age of the Internet of services, where end-users are service- and not product-oriented, this list needs to be updated to better reflect today's needs. The definition should cover aspects of **transparency**. Transparency in this context is the ability to inspect and introspect a service so that the delivered and guaranteed quality of the service agreement can be verified and observed. For us in Cloud Hosting this means that cloud providers should provide means to access performance information on one's provisionings so that, not only can one see what one has but one can build more useful services atop. This finding is reflected in the EU Future Internet Architecture Working Group findings.

3.1.3 Architecture Description of GEs

- FIWARE.ArchitectureDescription.Cloud.DCRM
- FIWARE.ArchitectureDescription.Cloud.SM
- FIWARE.ArchitectureDescription.Cloud.CloudEdge
- FIWARE.ArchitectureDescription.Cloud.ObjectStorage

The following GEs are partially available for internal needs of Core GEs, and will provider externally visible APIs in future releases:

- FIWARE.ArchitectureDescription.Cloud.Monitoring
- FIWARE.ArchitectureDescription.Cloud.Identity

Out of scope for the first release:

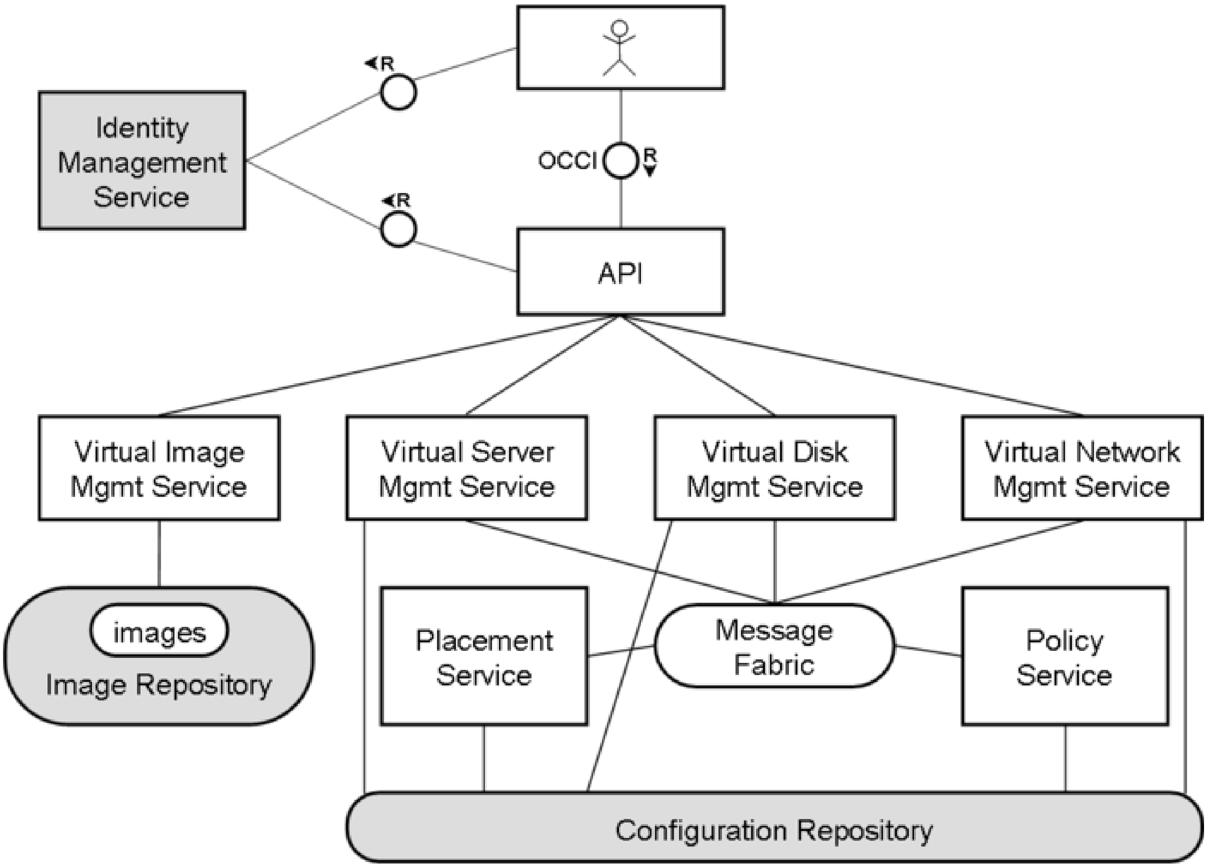
- FIWARE.ArchitectureDescription.Cloud.CloudEdgeRM
- FIWARE.ArchitectureDescription.Cloud.MonitoringAnalytics
- FIWARE.ArchitectureDescription.Cloud.Accounting
- FIWARE.ArchitectureDescription.Cloud.HybridCloudBroker
- FIWARE.ArchitectureDescription.Cloud.PaaSManager

4 FIWARE ArchitectureDescription Cloud DCRM

You can find the content of this chapter as well in the wiki of fi-ware.

4.1 Overview

This specification describes the DataCenter Resource Management GE, which is a key enabler to build a cloud solution. The following diagram shows the main components of the first release of DCRM GE, as well as main interactions.



On the above diagram, the *API* component is the front-end of DCRM, providing the Open Cloud Computing Interface (OCCI, see below). It dispatches the requests to components handling each of the resource types -- virtual servers, virtual disks, virtual networks and virtual images. At the back-end, different aspects of resource management are handled by corresponding internal services, such as policy service and placement service. Note that additional services and components will be added in future releases of DCRM GE.

4.2 Basic Concepts

The key concepts visible to the cloud user are:

- **Virtual server**, comprising a virtualized container that can host an arbitrary Operating System and an arbitrary software stack on top, installed within the virtual server. DCRM GE supports provisioning and life cycle management of Virtual Servers.

- **Virtual disk**, representing a persistent virtual disk that can be potentially attached to an arbitrary virtual server. DCRM GE supports provisioning of virtual disks, as well as their attachment to virtual servers.
- **Virtual network**, representing a logical network abstraction that would typically represent a L2 segment. DCRM GE supports provisioning of virtual networks, as well as attachment of virtual NICs of virtual servers to them.
- **Virtual image** -- a pre-packaged virtual server image. DCRM GE supports life cycle of virtual images, as well as provisioning of virtual servers based on virtual images.
- **Policies** -- a mechanism to control access to, and behavior of, operations on the above entities, in a given context (see more details below).

4.2.1 Policies

Both functional and non-functional characteristics of different operations (typically initiated by a cloud user) are subject to policies, which are set and managed by cloud administrator. Depending on the context in which the operation is executed, policies may influence the decision whether it is at all allowed to perform the operation, and if so, policies influence the way of executing the operations.

Some policies are visible to the cloud user, yet other are transparent to the cloud user and influence only the inner workings of DCRM. Consequently, all policies are classified into:

- User and Administrator Visible: examples of such policies include quotas on cloud resources, such as the number of compute instances that can be acquired simultaneously by a single user account, storage volume that is allocated to a new user account, storage volume that can be acquired in one operation, etc. Another examples of the user visible policies is given by the privileges, which are explicitly associated with the user accounts, resource utilization metering policies, such as the way the usage of a resource is rounded up or down, replication and monitoring policies for acquired cloud resources.
- Administrator-only Visible: examples of these policies include optimization criteria for the resource management, such as energy saving, load balancing, capacity management policies, such as physical hosts and network over-subscription ratio.

Each policy is an object that has the following attributes.

- Policy Name: name that specifies a policy;
- Description: description of a policy;
- Resource Scope: the scope of application of resource types and, possibly, instances;
- User Scope: users and user groups, whose cloud resources might be affected by this policy
- Time Scope: start and end times, defining when a specific policy applies (can be "always", "Each First Sunday of a Month", etc.)
- Priority: relative priority of the policy, allowing to determine relative importance of the service and order of application with respect to other policies.
- One or more conditions: conditions defining applicability of the policy (by default it is "always")
- Actions: management action or actions that should be executed by DCRM in response to detecting conditions for the policy. A default value can be "none". Other

actions may include sending a mail to administrator or user, triggering placement changes, triggering resource reallocation, etc.

In the first release, policies will be configured by administrator using configuration files. In the next releases we will develop a policy framework and an API to manage policies programmatically.

4.3 Example Scenario

The following sequence of operations describes a typical (simple) scenario of provisioning of a virtual server hosted in the Cloud:

- User authenticates with Identity Management GE, receives a token
- User creates ssh key-pair, to be used to authenticate with the guest OS within the virtual server instances
- User retrieves a list of available images and of virtual server flavors
- User requests a new virtual server
- User verifies that the virtual server creation has completed
- User retrieves the IP address allocated for the virtual server
- User connects to the virtual server using ssh

4.4 Main Interactions

DCRM provides a wide variety of operations to provision and manage the life cycle of cloud resources. The most important ones are listed below.

4.4.1 Virtual Images

- ***listVirtualImages*** -- return a list of all available virtual images (visible by the authenticated user)
- ***queryVirtualImages*** -- return a list of available virtual images, filtered by given query criteria
- ***getVirtualImageDetails*** -- return details of a virtual image (type, size, creation details, etc)
- ***uploadVirtualImage*** -- upload a new virtual image into the virtual image repository

4.4.2 Virtual Servers

4.4.2.1 *Provisioning*

- ***createVirtualServer*** -- provision a new virtual server with the given properties (virtual hardware, policy parameters, access, etc). Returns unique ID of the virtual server.
- ***destroyVirtualServer*** -- remove a virtual server

4.4.2.2 *Power Management*

- ***powerOnVirtualServer*** -- turn on a virtual server
- ***powerOffVirtualServer*** -- turn off a virtual server
- ***RestartVirtualServer*** -- restart a virtual server
- ***ShutdownVirtualServer*** -- shut down a virtual server (note: the ability to perform this operation on the fly depends on the capabilities of the underlying virtualization platform)

4.4.2.3 *Reconfiguration*

- ***resizeVirtualServer*** -- change the virtual hardware allocation for a virtual server (note: the types of resources for which the reconfiguration can be done on the fly depends on the capabilities of the underlying virtualization platofrm)

Inventory

- ***getVirtualServerDetails*** -- returns details of a virtual server (virtual hardware specification, state, associated policy parameters, access details, etc)

4.4.3 Virtual Disks

4.4.3.1 *Provisioning*

- ***createVirtualDisk*** -- provision a new virtual disk with the given properties (size, capabilities, etc). Returns unique ID of the virtual disk.
- ***destroyVirtualDisk*** -- remove a virtual disk

4.4.3.2 *Attachment*

- ***attachVirtualDisk*** -- attach a given virtual disk to a given virtual server (note: the ability to perform this operation on the fly depends on the capabilities of the underlying virtualization platform)
- ***detachVirtualDisk*** -- detach a given virtual disk from a given virtual server (note: the ability to perform this operation on the fly depends on the capabilities of the underlying virtualization platform)

4.4.3.3 *Inventory*

- ***getVirtualDiskDetails*** -- return details of a given virtual disk (size, capabilities, attachment details, etc)

4.4.4 Virtual Networks

4.4.4.1 *Provisioning*

- ***createVirtualNetwork*** -- provision a new virtual network with the given properties (e.g., VLAN ID, capabilities, etc). Returns unique ID of the virtual network.
- ***destroyVirtualNetwork*** -- remove a virtual network

4.4.4.2 *Attachment*

- ***attachVirtualServerToNetwork*** -- attach a virtual network interface of a given virtual server to a given virtual network
- ***detachVirtualServerFromNetwork*** -- detach a virtual network interface of a given virtual server from a given virtual network

4.4.4.3 *Inventory*

- ***getVirtualNetworkDetails*** -- return details of a given virtual network (ID, capabilities, attachment details, etc)

4.5 Re-utilised Technologies/Specifications

DCRM Open Specification utilizes Open Cloud Computing Interface. In the reminder of this section we provide a comprehensive review of OCCI with link for additional information included in the description

4.5.1.1 *Open Cloud Computing Interface (OCCI)*

The Open Cloud Computing Interface (OCCI) is a RESTful protocol and API for the management of cloud service resources. It comprises a set of open community-lead specifications delivered through the Open Grid Forum. OCCI was originally initiated to create a remote management API for IaaS model-based services. It has since evolved into a flexible API with a strong focus on integration, portability, interoperability and innovation while still offering a high degree of extensibility.

OCCI is a RESTful protocol and API for the management of cloud service resources. It comprises a set of open community-lead specifications delivered through the Open Grid Forum. OCCI was originally initiated to create a remote management API for IaaS model based Services. It has since evolved into a flexible API with a strong focus on integration, portability, interoperability and innovation while still offering a high degree of extensibility.

OCCI aims to leverage existing Standards Developing Organization (SDO) specifications and integrate those such that where a OCCI specified feature may not be rich enough a more capable one can be brought into play. An excellent example of this is the integration of both CDMI and OVF. In particular to those 2 previously mentioned standards, when combined together provide a profile for open and interoperable infrastructural cloud services.

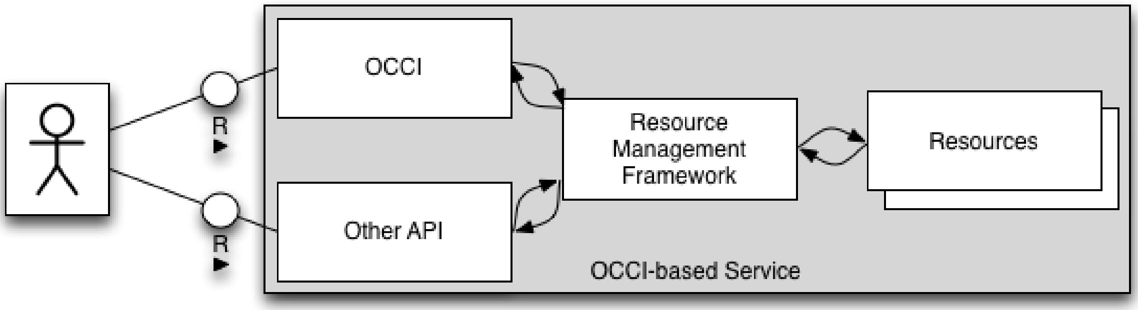
The main design foci of OCCI are:

- **Flexibility**: enabling a dynamic, adaptable model,
- **Simplicity**: do not mandate a large number of requirements for compliance with the specification. Look to provide the lowest common denominator in terms of features and then allow providers supply their own differentiating features that are discoverable and compliant with the OCCI core model,
- **Extensibility**: enable providers to specify and expose their own service features that are discoverable and commonly understood (via core model).

The specification itself currently comprises of 3 modular parts:

- **Core**: This specifies the basic types and presents them through a meta-model. It is this specification that dictates the common functionality and behaviour that all specialisations of it must respect. It specifies how extensions may be defined.
- **Infrastructure**: This specification is an extension of Core (provides a good example of how other parties can create extensions). It defines the types necessary to provide the a basic infrastructure as a service offering.
- **HTTP Rendering**: this document specifies how the OCCI model is communicated both semantically and syntactically using the RESTful architectural-style.

From an architectural point of view OCCI sits on the boundary of a service provider. It does not seek to replace the proprietary protocols/APIs that a service provider may have as legacy.



The main capabilities of OCCI are:

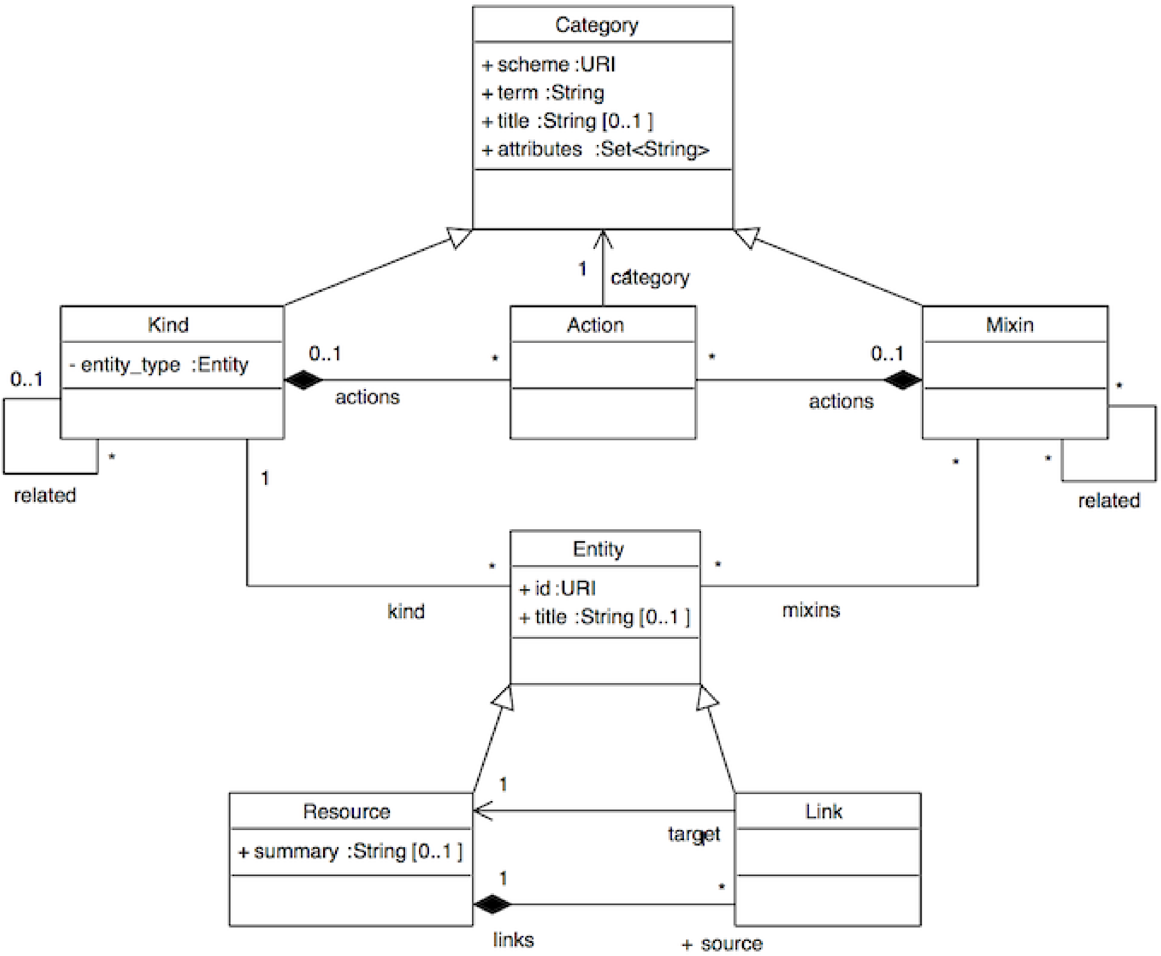
- **Basic type definitions** (attributes, actions, relationships):
    - Compute: defines an entity that processes data, typically implemented as a virtual machine.
    - Storage: defines an entity that stores information and data, typically block-level devices, implemented with technologies like iSCSI and AoE.
    - Network: defines both client (network interface) and service (L2/L3 switch) networking entities, typically implemented with software defined networking frameworks.
- **Discovery system**: Types and their instances' URL schema (provider can dictate their own) is discovered. Extensions are also discoverable through this system.

- **Extension Mechanism**: allows service providers expose their differentiating features. Those features are comprehended by clients through the discovery system. Resource (REST) handling (CRUD) of individual and groups of resource instances
- **Tagging & Grouping of Resources**
- **Dynamic Composition** that allows for the runtime addition of new attributes and functional capabilities
- **Template support** for both operating systems and resource types
- **Independent** of provisioning system (e.g. OpenStack, OpenNebula, vCloud etc.)

This section will briefly introduce the OCCI core and infrastructure model. For those needing further details it is advised to refer to the specific specification.

4.5.1.2 *Core*

The OCCI core model forms the basis of its type system. Types made available by OCCI implementations are defined by the Category entity. There are two specialised forms of the `Category` entity: `Kind` and `Mixin`. `Kind` defines the basic capabilities (attributes and functionality) of a type of resource. `Mixin` defines a means to further modify and extend a particular Kind's capabilities. This follows a common pattern that is seen in programming languages such as Lisp, Ruby and Groovy. Zero or more `Action`s can be associated with both `Kind` and `Mixin`. `Action`s define the executable functionality (e.g. methods) of either. Categories are self-descriptive, they can be discovered through a Query interface. The Query Interface allows for all service provider supported Categories to be discovered and described.



Where `Category` and its subclasses define the basis of a type system, the class `Entity` represents instances of OCCI Categories. `Resource` and `Link` are specialisations of Entity. It is Resource that represents entities that are managed by a management system. `Link` allows for the representation of associations between 2 or more `Resource`s.

The core model briefly presented here forms the basis for the Infrastructure as a Service specification. This specification extends the core model and provides an example of how one can extend the OCCI model to suit particular needs.

4.5.1.3 *Infrastructure*

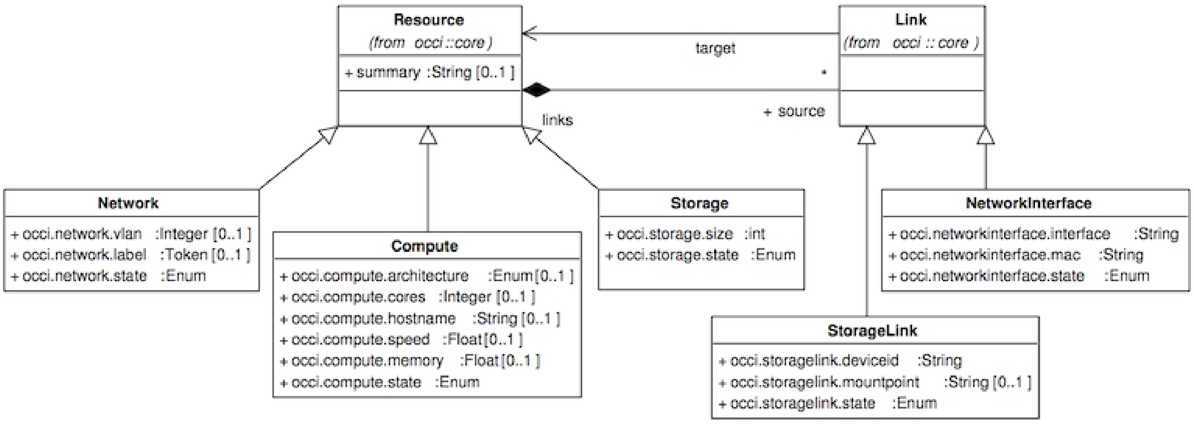
In the OCCI infrastructure specification, three central subclassed `Resource`s are specified.

- `Compute`: Information processing resources.
- `Network`: Interconnection resource and represents a L2 networking resource. This is complemented by the IPNetwork Mixin.
- `Storage`: Information recording resources.

Complementing these specialised `Resource`s are two specialised `Link`s and a means to capture the notion of templates. The `Link`s are:

- `StorageLink`: connects a `Compute` instance to a `Storage` instance. CDMI interoperability is accomplished through this specialised `Link`.

- `NetworkInterface`: connects a `Compute` instance to a `Network` instance. This complemented by an `IPNetworkInterface Mixin`.



There are two types of templates that are specified:

- `ResourceTemplate`: a provider-defined `Mixin` instance that refers to a preset Resource conguration.
- `OsTemplate`: allow clients specify what operating system must be installed on a requested `Compute` resource.

4.5.1.4 *Interacting & Examples*

The OCCI HTTP Rendering specification defines how one can interact with a compliant OCCI implementation using HTTP. This specification is based on a RESTful approach. An example, as defined by the specification, of how a `Compute` (e.g. VM) resource is instantiated (provisioned) is shown below.

The Request has 3 Categories defined:

```
* One to define the type of the resource. In this case 'compute'
* The resource template which defines the size of the VM (Here:
m1.tiny)
* And the operating system which is to be started. In this case an
linux based OS 'cirros'
```

Also an security token is given which let's the service determine the user.

Request:

```
> POST /compute/ HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0)
libcurl/7.21.4 OpenSSL/0.9.8r zlib/1.2.5
> Host: localhost:8787
> Accept: */*
> Category: compute;
scheme="http://schemas.ogf.org/occi/infrastructure#"; class="kind"
> Content-Type: text/occi
> X-Auth-Token: b642acaa44f54f7eb14c6e0bdf162de3
> X-Auth-Project-ID: 1
> Category: m1.tiny;
scheme="http://schemas.openstack.org/template/resource#";
class="mixin"
> Category: cirros-0.3.0-x86_64-blank;
scheme="http://schemas.openstack.org/template/os#"; class="mixin"
>
```

Response:

The reponse indicates that the request was succesfull (201 OK) and also the location of the newly created resource.

```
< HTTP/1.1 201 OK
< Content-Length: 2
< Content-Type: text/plain; charset=UTF-8
< Location: http://localhost:8787/compute/40675abc-c4ca-e6dd-ac7e-
fa057cd5b164
< Server: pyssf OCCI/1.1
<
OK
```

4.5.2 FIware Extensions

For this release of the DCRM, there are a number of extensions that are required. Those extensions are:

1. A `Mixin` to allow clients specify the administrative password of the VM to be provisioned.
    - Category definition: `Category: public_key; scheme="http://schemas.openstack.org/instance/credentials#"; class="mixin"`

  - Related attributes: name: `org.openstack.credentials.publickey.name` type: `string`, name: `org.openstack.credentials.publickey.data` type: `string`
2. A `Mixin` to allow clients specify the public ssh key of the VM to be provisioned.
  - Category definition: `Category: admin_pwd; scheme="http://schemas.openstack.org/instance/credentials#"; class="mixin"`
  - Related attributes: name: `org.openstack.credentials.admin_pwd` type: `string`
3. A `Mixin` to allow clients to specify if the VM to be provisioned should be Trusted Pool Computing enabled.
  - Category definition: `Category: tcp; scheme="http://schemas.fi-ware.eu/occi/infrastructure/compute#"; class="mixin"`
  - Related attributes: name: `eu.fi-ware.compute.tcp` type: `string`

In future releases, further extensions will be added to support additional capabilities, such as placement considerations, policies, etc.

4.6 Basic Design Principles

When applied to DCRM, the general design principles outlined at Cloud Hosting Architecture can be translated into the following key design goals:

- Fully-automated provisioning and life cycle of compute, storage and network resources, requested, managed and released via a standards-based REST API (OCCI)
- High resource utilization, while providing the necessary levels of isolation, availability and performance of provisioned resources
- Ability to dynamically control the amount of allocated resources, as well as to monitor the actual resource usage
- High availability and scalability of the management stack
- Non-disruptive, automated administrative tasks (e.g., infrastructure maintenance)
- Avoid non-authorized access to resources and workloads

4.7 Detailed Specifications

Following is a list of Open Specifications linked to this Generic Enabler. Specifications labeled as "PRELIMINARY" are considered stable but subject to minor changes derived from lessons learned during last interactions of the development of a first reference implementation planned for the current Major Release of FI-WARE. Specifications labeled as "DRAFT" are planned for future Major Releases of FI-WARE but they are provided for the sake of future users.

4.7.1 Open API Specifications

- DCRM Open RESTful API Specification (PRELIMINARY)

5 FIWARE ArchitectureDescription Cloud SM

You can find the content of this chapter as well in the wiki of fi-ware.

5.1 Copyright

5.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

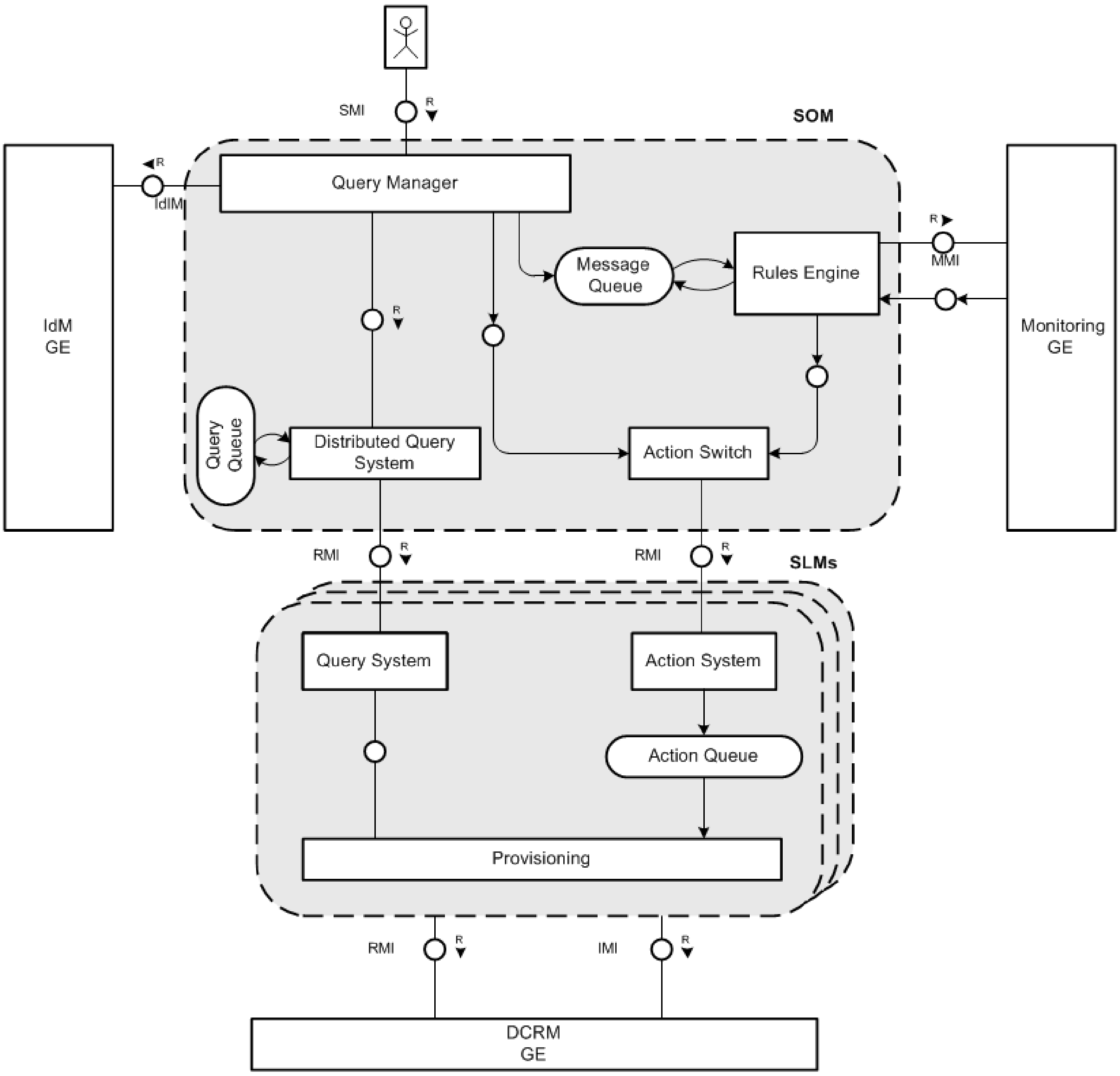
5.3 Overview

This specification describes the Service Manager GE, which is the key enabler to provide an automated control solution of services through scaling up/down and in/out in an automated manner.

The Service Manager helps to save valuable time/resources by automating the management of recovery tasks that need to be done repeatedly. Since it is working at the service level, operations are done only once for the whole service, rather than as many times as the VMs number defined by the given service.

The Service Manager must support the integration with any public or private cloud provider and must help to build federated services without requiring underlying cloud federation.

This GE allows Cloud user to extend the basic functionalities offered by IaaS DCRM GE in order to cope with elasticity management, vApps/Services and Virtual Data Centers. The following diagram shows the main components of Service Manager Generic Enabler (IaaS SM GE).



IaaS SM architecture specification

On the above diagram, the Service Manager Interface (SMI) is the front-end of the IaaS SM GE, providing an [Openstack Compute API] compliant interface. It dispatches the requests received from the cloud user or cloud portal to components handling each of the services management operations/tasks-- deploy servers, deploy services, management of scalability rules. At the back-end, different aspects of service management are handled by the corresponding internal service components, such as elasticity, service management, Virtual Data Center (VDC) management, Organization management. A detailed description of these operations is shown in the next sections. Note that additional service management functionality and components will be added in future releases of IaaS SM GE.

5.3.1 Target Usage

The IaaS Service Management GE introduces a layer on top of IaaS Resource Manager GEs (both DataCenter and Cloud-edge) in order to provide a higher-level of abstraction to Application/Service providers. Thus, the Service Provider does not have to manage the individual placement of virtual machines, storage and networks on physical resources but deal with the definition of the virtual resources it needs to run an application/service, how these virtual resources relate each other and the elasticity rules that should govern the dynamic deployment or deactivation of virtual resources as well as the dynamic assignment of values to resource parameters related to virtual resources (CPU, memory and local storage capacity on VMs, capacity on virtual storage and bandwidth and QoS on virtual networks, for example).

5.4 Main concepts

Following the above FMC diagram of the IaaS Service Manager, in this section we introduce the main concepts related to this GE through the definition of their interfaces and components and finally and example of their use.

5.4.1 Basic Concepts

The Service Manager is responsible for the deployment and instantiation of service applications (controlling the service lifecycle) and dynamically asking for virtualized resources to the Data Center Resource Manager (DCRM). In addition, Service Manager tries to avoid over/under provisioning of resources and reduce over-costs based on SLAs and business rules protection techniques.

The key components visible to the cloud user could be differentiated between the interfaces and the components, together with the explication of the concepts used on it, each of them is described below.

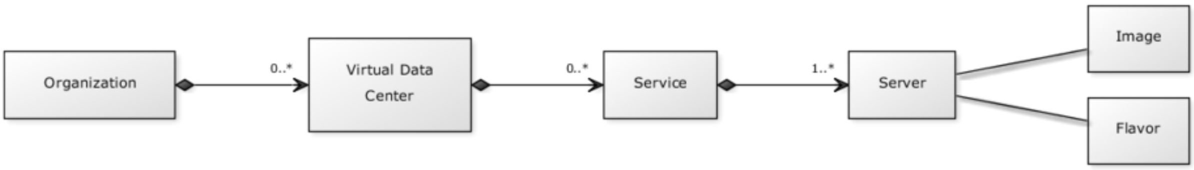
5.4.1.1 *Concepts*

To use the IaaS Service Manager Interface (SMI) effectively, you should understand several key concepts, see the diagram below:

- **Server.** A server is a virtual machine instance in the computing system. Images and flavors are required elements in order to create a server (also including the name to identify it).
- **Virtual Appliances or Services** (vApp). This entity represents virtual infrastructure designed to run on a virtualization platform. Typically a vApp or Service is considered as a server or set of servers running on top of a hypervisor, but the bounds of vApp or service concept is open to the cloud user. Anyway the concept of service here is totally different from the concept of server defined in the Data Center Resource Management (DCRM) GE. Here a Service involves the definition of one or several virtual machines with or without its own elasticity rule. Last but not least, services could be connected themselves through the same virtual data center in which they are running.
- **Virtual Data Centers** (VDC). vApps/Services and/or servers reside in a VDC context. A VDC is a container of virtual infrastructure that has a set of virtual resources (e.g.,

computing capacities, storage capacities) to support the former. In other words, a VDC is a pool of virtual resources that supports the virtual infrastructure it contains.

- **Organizations** (Org). VDCs are owned by organizations. An organization represents any kind of independent unit, which manages its own cloud resources (e.g. enterprises, divisions, groups,...).
- **Virtual Infrastructure**. Any kind of IaaS resource offered to cloud consumers as part of a cloud service that may be managed through SMI. E.g., vApps/services, servers, volumes, firewalls, load balancers...
- **Flavor**. A flavor is a hardware configuration that could be applied to a server. Each flavor has a unique combination of disk space and memory capacity. An example of this combination could be for example 8 virtual CPUs, 16 Gb RAM memory, 10 Gb HDD and 160 Gb of ephemeral disk (a disk that is stored locally on the hypervisor host). It must be available previously in order to allow creating servers. Each flavor has a unique combination of disk space, memory capacity and priority for CPU time.
- **Image**. An image is a collection of files used to create or rebuild a server. FIWARE will provide a number of pre-built OS images but the cloud user may also create his own images using the appropriate functionality (see Main Interactions). These custom images are useful for backup purposes or for producing "gold" server images if you plan to deploy a particular server configuration frequently.



Concepts class diagram

5.4.1.2 *Interfaces*

Service Manager is currently composed of four main interfaces:

- The **Service Manager Interface** (SMI) is the REST interface exposing all features of the Service Manager. The SMI allows service providers to control the service provisioning lifecycle using a service manifest, which is based on the Open Virtualization Format (OVF). This file declares the service components, service requirements, monitoring, SLA targets and elasticity rules (based on the W3C RIF). The SMI implements a standard API compliant with OpenStack Compute API working at the whole application level abstraction. The SMI is composed of three different feature types:
    - Service deployment features, which involve the different operation to deploy and un-deploy services and also retrieve information about them.
    - Service provisioning features, which involve the operations related to the deployment of applications. It means that an applicationcan can be deployed on a single VM or on a set of VMs. In case that this deployment is made over a set of VMs, it is needed to know the way in which these VMs are connected through a network, which are their interrelations, etc.

- The **Resource Manager Interface** (RMI) abstracts the Service Manager from the heterogeneity existing in cloud infrastructure providers. It is a REST client interface (OCCI compliant) and includes primitives and data types dealing with Services or VMs.
- The **Monitoring Manager Interface** (MMI) provides a REST client interface compliant with Openstack Compute API and provides connectivity to the Monitoring GE in order to retrive information about the status of the deployed Services. This information is used by the Rules Engine in order to activate or deactivate the elasticity rules associated to a specific service.
- The **Image Repository Manager Interface** (IMI) provides a REST client interface compliant with CDMI in order to access to the Image Repository for administrative operations associated to object storage and also the operation related to the management of Services and/or Servers Images. The Image Repository it is the place where you will be uploading your images as well as the place from which they will be consumed by the rest of the cloud system.

5.4.1.3 *Components*

The IaaS Service Manager is split into two main modules that could be deployed in the same or different machines:

- **Scalability & Optimization Manager** (SOM) is responsible for the management of monitoring events and scalability rules and the distribution of queries and actions to the different Service Lifecycle Managers (SLM), see below. It is composed of the following components:
    - **Query Manager**, which is the SMI server and is the **Cloud Entry Point** (CEP) of the Service Manager. The CEP is the main entry into the IaaS provider. All other data is discovered, iteratively: pointers to machines, volumes, networks, services, virtual data centers, etc. Query Manager has to separate the incoming messages in Actions, Queries and Elasticity Rules. Action means an asynchronous operation associated to a service. By contrast query is a synchronous operation associated to a service. Finally, Elasticity Rules allow to describe the rules to be applied when we want to scale up/down a server.
    - **Rules Engine**, which provides the management of the elasticity rules based on monitoring data. These rules are associated to a service description and passed from the Query Manager through a shared memory (**Message Queue**). The Rules Engine triggers actions when a specific elasticity rule switches on based on the data provided by the Monitoring GE.
    - **Action Switch**, which takes as input both the asynchronous actions coming from the SMI and the actions generated by some activated elasticity rules. These actions involve specific operations over a Virtual Machine like increase or decrease the memory, the virtual CPU, the disk capacity (scale up/down), or the number of virtual machines associated to a service (scale up/down). Besides, it must know which host it has to send the action requests to.
    - **Distributed Query System**, which receives the definition of a service through the Query Manager and sends the information to the appropriate Query System (see bellow). It uses shared memory in order to store the information related to the service (**Query Queue**) in order to use it afterwards. Since the Query System could be in a different Host, The Distributed Query System must know to which host we have to send the information related to a service.

- **Service Lifecycle Manager** (SLM) is responsible for providing queries and actions over the services through the provisioning component, which connects directly to the Data Center Resource Manager GE.
  - **Query System** receives the queries associated to a specific resource and passes to the Provisioning component.
  - **Action System** receives the actions from the Action Switch and sends it to a shared memory (**Action Queue**) in order to be processed afterwards by the provisioning component in an asynchronous way.
  - **Provisioning**, this component deal with the interface related to the Data Center Resource Management (DCRM). Its purpose is to translate the queries and actions to the appropriate interface in order to be sent to DCRM. In case of actions, Provisioning is responsible of deleting the actions that, it takes from the Action Queue.

5.4.2 Example Scenario

The Service Manager provides a more flexible and easier way to use the FI-WARE platform to deploy services with all the benefits of the IaaS cloud model. As it is stated in both scenarios, now cloud service providers can define the configuration and dynamic behavior of their services in an easy and standard way. The Service Manager hides the complexity of the platform to Service Providers, allowing them to avoid complicated management processes and allowing them to focus on the service definition.

Thus, this section tries to provide a general scenario that illustrates how the IaaS Service Manager can be used. Concretely, we are going to exemplify how the Service Manager architecture works when a service provider decides to use the FI-WARE platform to deploy one of its services.

5.4.2.1 *Deployment Scenario*

As a previous step before the beginning of the deployment process, we need to get the list of Images available in the DCRM through the ***getImageList*** interface. This interface provides the information (including the url) of the images to be selected. In the same way, we need to recover the list of flavors available in the DCRM in order to use them through ***listFlavors*** interface.

In the next steps, before the beginning of the deployment process, the service provider must define the service structure, conditions and characteristics. For understand this please refer to the OpenStack extension API. This information will be sent through the ***CreateServer*** or the ***CreateService*** interfaces, depending if we provide the description based on Image Id and Flavors Id or based on a more extended description of all service's servers or service's virtual machines (OS, hardware characteristics, networks, etc.). This last option means the use of OVF descriptor. At this point, the service provider has to generate the OVF section with all the information related to the VMs involved in the services and pass this information to the DCRM interfaces.

Besides, in case that we have the definition of servers in OVF, Service Manager revises the OVF descriptor in search of useful information for the service lifecycle management. The start-up order of all the VMs of the Service, scalability rules, resource requirements, etc. is some example of the available information within the OVF Descriptor. Service Manager also translates the OVF Descriptor into the needed DCRM templates, which are stored in the IaaS DCRM.

At this point, we have the possibility to stop a specific server through the ***stopServer*** interface, which stops a specific server or VM. We have the possibility to restart it again invoking the ***startServer*** interface. By default, when we create a server or a vApp/Service, the server/servers are started.

5.4.2.2 *Scalability Scenario*

As commented previously, Service Manager will provide service elasticity. Thus, service providers define elasticity rules in the service manifest (OVF descriptor) to assure the right operation of their services and/or the final user experience. Moreover, the service provider has defined elasticity rules, which allows adding a new instance of a concrete virtual machine to avoid the system overload.

We have the possibility to define afterwards the elasticity rule using the ***createRule*** interface, which sends to a specific instance of a server the corresponding rule defined in W3C RIF format. This rule might be changed in the future, if we decide to add more controls over the data or managing new information not previously taken into account, through the ***updateRule*** interface.

Besides, the Monitoring GE provides KPIs values for the service to the Service Manager through the MMI. The Service Manager evaluates, in real time, the elasticity rules of the service with the information provided by the Monitoring GE in the rule engine. When the corresponding KPI exceeds the given threshold, the SM starts the deployment process of a new VM instance through the ***createServer*** interface or converts to a different flavor (scaling the server up and down) through the ***resizeServer*** interface.

If the problem that originated the scaling event finishes, the Rule Engine can decide to undeploy a server through the ***deleteServer*** interface or to scale it down to a less restricted hardware requirements flavor through the ***resizeServer*** interface.

5.4.2.3 *Real Scenario*

Based on the previous information, the idea in this section is to translate it into a more concrete real scenario in which our purpose would be firstly the deployment and configuration of a service based on a set of virtual machines and networks and secondly the scaling up or down of the service according to KPIs.

Imagine that we want to develop a service consisting of an auction site prototype modeled (e.g. eBay.com). It contains all the elements of a typical enterprise application and potentially allows us to benchmark the performance of the platform. This service needs to implement functionalities like selling, browsing and bidding. Like any auction portal, the demand for any application fluctuates; consequently the IaaS SM must identify bottlenecks (idleness) at the various tiers (database, portal, business logic) and deploy/remove resources, ideally automatically.

The steps that we must take in order to deploy the service can be summarized as follow:

1. The user describes the auction service, which involves the definition of the architecture of the service (a web application to connect to the business layer and finally a database layer to store the information) using OVF format.

2. The user generates all the images (virtual machines) involved in the service and stores them into the Object Storage GE. This operation is made through the OpenStack Compute API ***CreateImage*** interface.
3. Using the OpenStack Compute API, the user requests the service deployment. This request includes the OVF descriptor.
4. IaaS SM parses the OVF descriptor in search of useful information for the service lifecycle management (e.g. scalability rules, startup order, needed KPIs, etc.). It also translates the OVF descriptor to OpenStack Images (Glance) previously stored in the Object Storage GE.
5. Using the OCCI interface, the IaaS SM starts the final phase of the deployment (network generation and virtual machine startups).

Once the service is up and running, the scaling process starts the process of scaling the service. IaaS SM manages the horizontal scaling by adding more of the same virtual machine image, configuring the necessary load balancer to deal with the different instances of this image, which allows increasing the number of front-end/backend nodes if the measured KPIs violate some rules defined by the user. The steps in this case would be the following:

1. The monitoring system collects the KPIs information samples of each virtual machine involved in the service.
2. IaaS SM evaluates the elasticity rules to apply scalability if needed
3. Using the OCCI API, IaaS SM requests the deployment of a new node (e.g. front-end component) of the service
4. OpenStack manages the deployment of the new virtual machine of the services, taking the info from the Object Storage GE.
5. IaaS SM configures the new instance of the web service application in order to use it and configure a load balancer in order to redirect traffic from one node to another.

5.5 Main Interactions

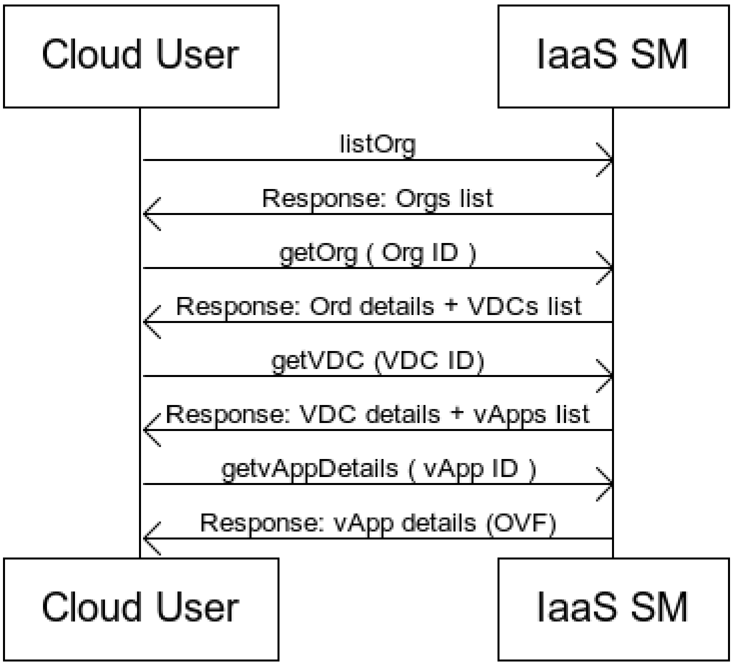
In this section, the SMI operations are described. These operations are classified in the following areas:

- Browsing operations. These operations are used to discover cloud resources and obtain their specific information.
- Provisioning operations. These operations are used to provision new resources in the cloud.
- Management operations. These operations are used to manage the resources previously deployed in the cloud.
- Elasticity & Monitoring operations. These operations are used to manage the elasticity rules and obtain the information of monitoring to work with them.
- Miscellanea operations. Operations not included in the previous areas.

5.5.1 Browsing operations

Browsing API includes all those operations used for discovering and browsing cloud resources. The output of the operations will be returned in XML or JSON format. These operations include the following:

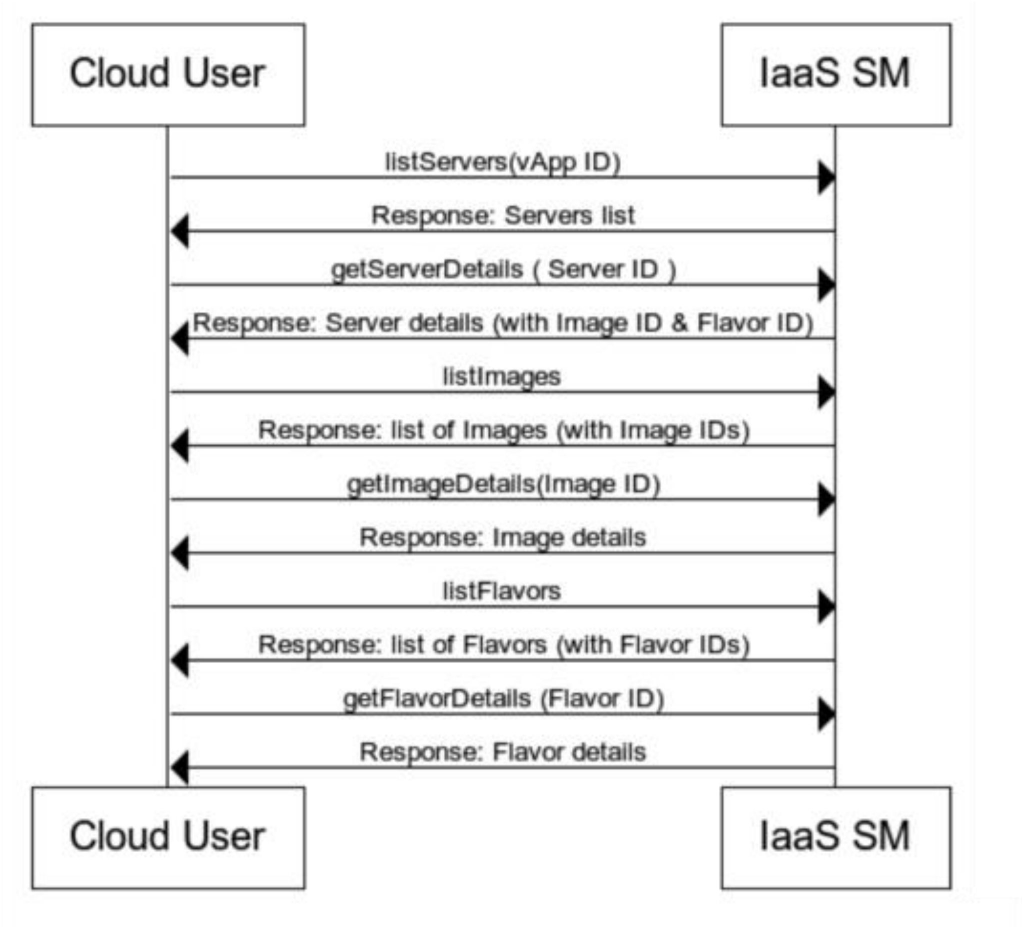
- List of Organization elements available through ***listOrg***, each one representing an organization the user (cloud user) belongs to. If the API consumer is not authenticated in the platform, this element shall not appear. Each of the elements of the list must be the ID of the Organization and the name of it. This return will be in XML or JSON format.
- If we want to recover the details of each Organization we have to use ***getOrg*** with the parameter of Organization ID, this returns with the list of VDC IDs belonged within this Organization.
- Using the VDC IDs provided by ***getOrg***, it is possible to obtain the representation of a VDC through the ***getVDC*** interface which returns a name attribute, a VDC ID and a description of the VDC. The information must include the Organization ID, which references the parent organization for this VDC and the list of references to vApps/Services instantiated for this VDC identified by their vApp/Service IDs. Besides, ***getVDC*** returns the capacity restrictions for this VDC (Storage and Compute), the list of networks available for this VDC identified by their Network ID.
- Using the vApp/Service IDs, we can use the ***getServiceDetails*** interface, which returns the description of a previously provisioned vApp/Service, returning a vApp/Service element as output in [OVF format](). Children vApps/Services and Servers can be also browsed. This vApp/Service element will include the list of Servers identified through their Server ID.



Browsing operations 1

- There is the possibility to use the ***listServers*** interface with parameter vApp/Service ID to get the list of servers associated to a vApp/Service, including Image ID, flavor ID, Networks ID and their IP associated.

- Using the previously obtained server ID, we can use ***getServerDetails***, which obtains the description for a previously provisioned Server, returning a Server element as output. Including server ID, image ID, flavor ID, Networks ID and its IP addresses (IPv4 and/or IPv6).
- ***listImages*** queries the public Image catalogue provided by the IaaS DCRM, returning a description of the stored Images in the Image Repository and identified by image ID.
- ***getImageDetails*** , with parameter Image ID, will return the specific information for the Image identified by this parameter.
- ***listFlavors*** returns all the available flavors identified by their flavor ID.
- ***getFlavorDetails***, with parameter flavor ID, will return the specific information for the Image identified by this parameter. This return will be in XML or JSON format



Browsing operations 2

5.5.2 Provisioning operations

This includes capabilities for instantiating and configuring Virtual Infrastructure (vApps/Services, VDCs, Orgs and Servers). Provisioning API includes all those operations for resource provisioning like the following.

- vApp/Service provisioning through ***createService***, which instantiates a new vApp/Service in target VDC based on an OVF description and several deployment parameters, which define several aspects regarding the deployment of the vApp/Service like network connectivity, and so on. This describes how the vApp/Service is going to be provisioned. Need to know the VDC ID.
- vApp/Service unprovisioning through ***deleteService***, which removes an existing vApp/Service, including all its children, releasing its consumed resources. It would need the vApp/Service ID.
- Server provisioning through ***createServer*** specifying the flavor id, image id and name. The Servers creation will be always associated to a vApp/Service and then we need to pass the vApp/Service ID like other parameter. It will return the description of the Server including also the Server ID, user ID, admin password, progress attribute (0-100% completion) and status (BUILD, ACTIVE, ERROR).
- Server unprovisioning through ***deleteServer*** with parameter Server ID deletes a cloud server from the system.
- Once provisioned, vApp/Service can be updated invoking the ***updateService*** interface with the new OVF description like parameter.
- Once provisioned, Server configuration may be altered in terms of its flavour through ***resizeServer*** with parameter Server ID and Flavor ID. This operation, in essence, scales the server up and down. The original server is saved for a period of time to allow rollback if there is a problem. The server status will pass from ACTIVE to RESIZE and finally to VERIFY_RESIZE.
- To confirm the resize, we have the ***confirmResize*** with parameter Server ID, which delete the original server.
- If we detect a problem during the resize we can call ***revertResize*** with parameter Server ID to roll back to the original server.
- Server cloning. Once provisioned, a VM may be cloned, creating a new Server with the same configuration, same virtual disks and same network connectivity, through the ***createImage***. It will need the Server ID in order to create the Image.

Regarding the Organization, it requires a special treatment. The creation of an Org resource may mean the establishment of a contractual relationship between cloud provider and cloud consumer. This relationship cannot be made by just an invocation of a programmatic interface due to the legal issues of accepting service level agreements (SLA) and End-user license agreement (EULA) implicitly. On the other hand, there are some scenarios where this constraint is not required, as when the cloud cloud consumer and cloud provider are in a trusted environment and no contractual relation is needed. This interpretation of the organization move to establish a close relationship between them and the accounting and billing mechanism to be implemented in FI-WARE are not covered in the current release of FI-WARE Cloud. Anyway the operations that should be available are the following:

- Cloud consumer may require the instantiation of a new organization by invoking the ***createOrg*** interface, in which we need to pass the information about the Organization (name and acronym) and the identification of the different contracts (Contract ID).
- Due to finalization of contractual relationship, cloud consumer may require the suppression of its Org instance by invoking the ***deleteOrg*** interface, with the parameter Org ID. Note that the elimination of a Organization will produce the suppression of all their VDCs, vApps/Services and Servers deployed and therefore the liberation of all its resources.
- The update operation should be available through invoking the ***updateOrg*** interface, with the new information about the Organization (name, acronym and or contract IDs) together with the Org ID parameter in order to identify it.

5.5.3 Management operations

Management API includes operations for managing cloud resources. These operations may be categorized as follows.

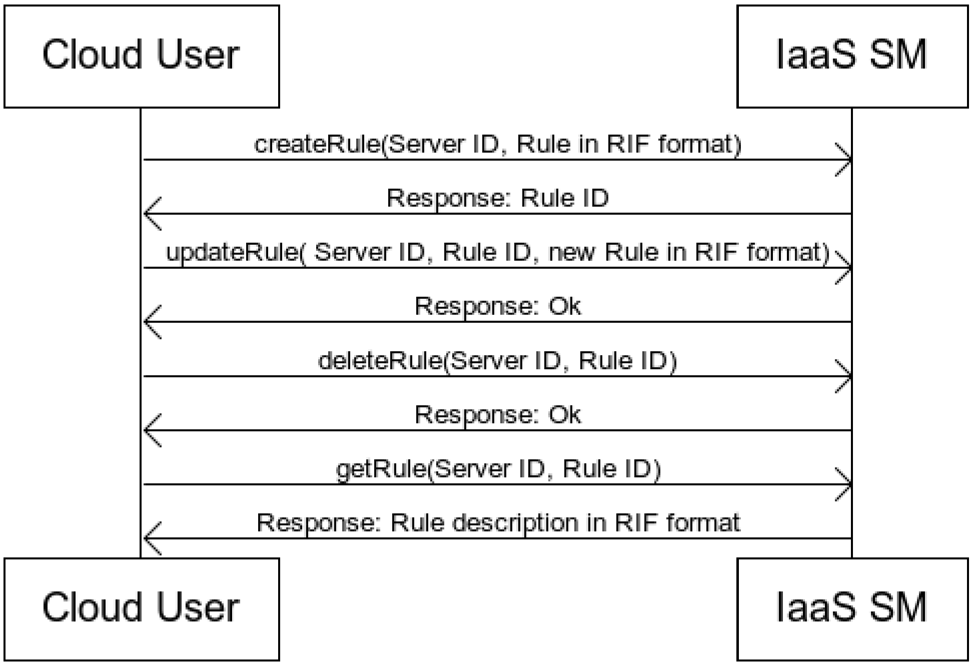
- vApp/Service activation and deactivation through ***activateService*** interface with parameter vApp/Service ID and a boolean parameter (true or false). vApp/Service deactivation consists of releasing vApp/Service resources and storing them for a later use. The Server included in a vApp/Service will be stopped and the memory and compute resource will be freed. The vApp/Service activation will produce an activation of all the servers associated to this service. The vApp/Service may be activated again invoking the activation operation with the boolean parameter set to true.
- Stop a Server through ***stopServer*** interface with parameter Server ID, will stop the server.
- Start a Server through ***startServer*** with parameter Server ID. By default all deployed servers will be ACTIVE at the beginning.
- Reboot a server through ***rebootServer*** with parameter Server ID and the type of reboot (soft or hard). A soft reboot the operating system is signalled to restart. A hard reboot is equivalent to a switch off the server.
- We can change the name and IPs associated to a Server through the ***updateServer*** interface with parameters Server ID and details of name and IPs.
- The Image stored can be deleted from the Image Repository through ***deleteImage*** interface, which need only the Image ID to do it.
- Once provisioned, a Server may be cloned, creating a new virtual machine with the same configuration, same virtual disks and same network connectivity. This operation is invoked through the ***cloneServer*** interface with Server ID parameter.

5.5.4 Elasticity & Monitoring operations

The elasticity rules associated to a Server are associated to the description of the vApp/Service, anyway we can have the possibility to create new rules in RIF format. The interfaces that we can use are the following.

- We can assign a new rule to a Server if it has not be specified in the OVF or we want to add a new one through the ***createRule*** interface specifying the elasticity rule and the Server ID as parameters. It produces a new rule associated to a Server identified by Rule ID.

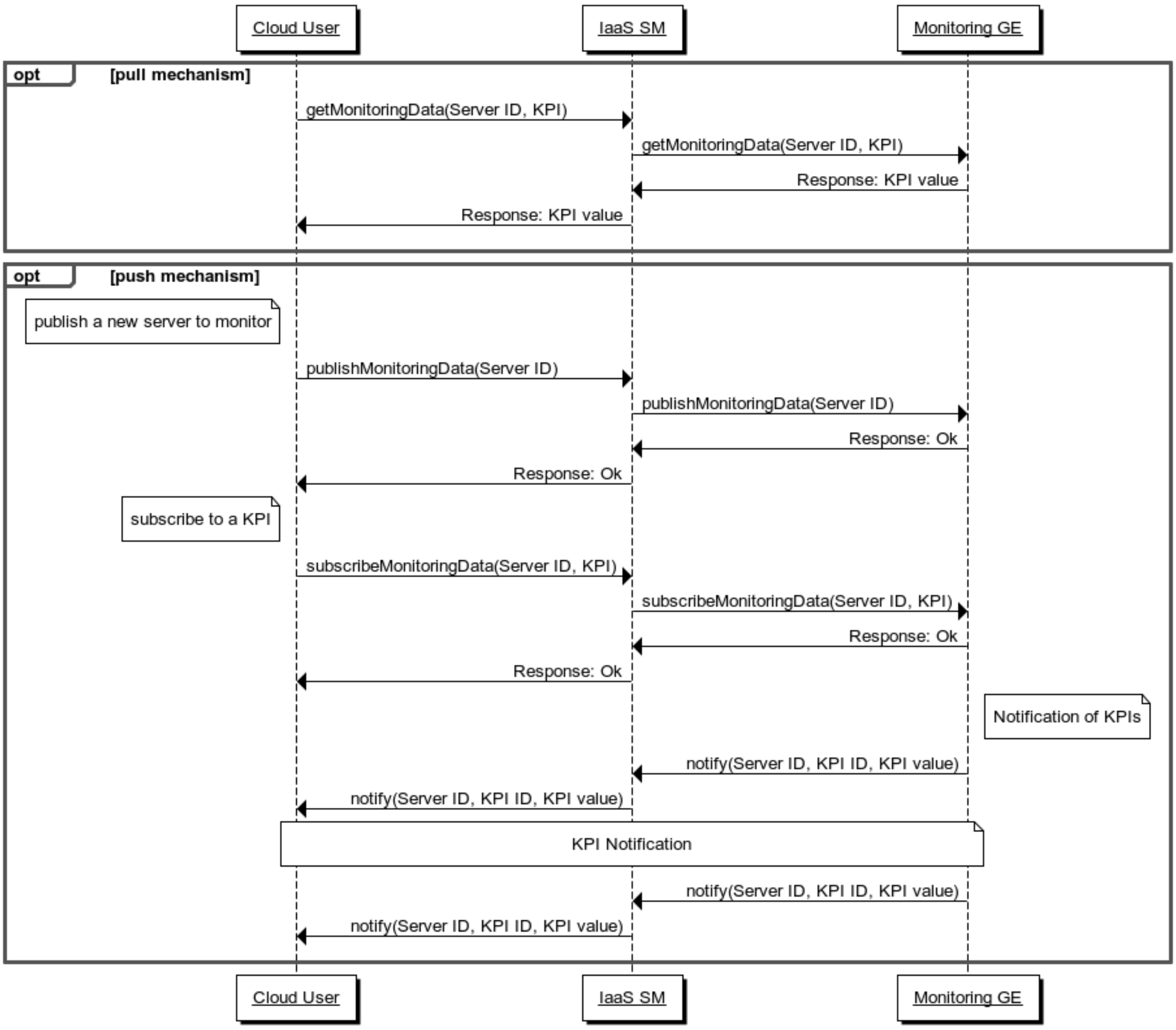
- We can update a specific rule through ***updateRule*** interface with parameters new elasticity rule, service ID and Rule ID of the rule to be updated.
- Invoking ***deteleRule*** interface with parameter Server ID and Rule ID allows to delete a specific rule.
- Invoking the ***getRule*** interface with parameter Server ID and Rule ID, we can obtain the description of the elasticity rule in RIF format.



Elasticity rules operations

Regarding monitoring data, the IaaS Service Manager has to implement a way to access the KPIs of the probes (see the Monitoring GE for more information). It has to implement a pull mechanism in order to periodically ask the Monitoring GE and a push mechanism in which it subscribes to the Monitoring GE.

- Using the ***getMonitoringData*** interface, the Service Manager can ask the Monitoring GE for a specific or complete list of KPIs of a Server identified by its Server ID.
- Using the ***publishMonitoringData*** interface, the Service Manager informs to Monitoring GE that it wants to monitor a specific Server (identified by its Server ID) so that the Monitoring GE triggers the deployment of a probe on it.
- Using the ***subscribeMonitoringData*** interface, the Service Manager informs to Monitoring GE that it wants to monitor a specific Server and KPI. This operation will generate a subscribe/notify mechanism in the Monitoring GE (see Monitoring GE description for more details).



Monitoring operations

5.5.5 Miscellanea

In this section the operations not included in previous categories are described. It includes the following miscellanea operations.

- API consumer verification. This includes the ability for IaaS SM to validate the user against the IdM. This will be done through ***validateToken*** interface with parameter token to be validated.
- Get credentials from IdM to assign to a VM. This operation request the admin credentials to the IdM in order to incorporate in the server administrator user and password. The interface will be ***getCredentials*** with no parameter and will return a credential to be used in the server user/password configuration.
- At any moment in time the administrator password of a server can be changed through the ***changePassword*** interface with parameter password.

5.6 Basic Design Principles

5.6.1 Design Principles

IaaS Service Manager has to provide the following technical requirements:

- The cloud works in a distributed environment. The design does not depend on any global, critical or centralized service and the solution provided has to be highly available and scalable.
- IaaS Service Manager allows running a single instance of a service on a server, being accessed by multiple users.
- The Cloud Users can either deploy a single instance of a service or use a single instance of a service previously deployed.
- A single instance of a service could be composed by either a simple server over a virtual machine or a group of servers executing the same or different virtual machines. The system should also be able to deploy full services from OVF file format.
- IaaS Service Manager should manage Virtual Data Center resource usage limits. Each virtual data center will have defined some amount of resources which will be assigned to the services deployed on it.
- IaaS Service Manager supports standard REST interfaces, with all the information and capabilities of the system being exposed via this interface.
- Cloud Users can deploy services onto a physical server without knowing the infrastructure providers by the cloud providers in which the virtual machines will be finally deployed. IaaS Service Manager should be able to work transparently with multiple hypervisor technologies and provide an abstraction layer to the cloud user.
- IaaS Service Manager and its interface are capable of being implemented and used in different operating systems environments.
- IaaS Service Manager has to support federation capabilities, which means that it has to be able to to deploy VMs and move them between different cloud data centers. A data center is a centralized repository, either physical or virtual, for the storage, management, and dissemination of data and information organized around a particular body of knowledge or pertaining to a particular business. Each datacenter will be managed by a different IaaS Service Manager instance.
- IaaS Service Manager should aggregate data from different datacenters when asked about global information and should switch distribute actions to the right data centers.
- IaaS Service Management should offer an interface for the clients to create service management rules, with infrastructure management capabilities in a standard way.
- IaaS Service Manager should be able to create and manage virtual machine templates, as well as full OVA files.
- The system should manage monitoring information from Monitoring GE.
- The system has to support both synchronous and asynchronous notifications.
- The system has to implement mechanisms to secure the communications and operations through authentication and authorization processes.

5.6.2 Resolution of Technical Issues

When applied to SM, the general design principles outlined at Cloud Hosting Architecture can be translated into the following key design goals:

- Fully-automated provisioning, elasticity and life cycle of service, requested, managed and released via a standards-based REST API (Openstack Compute API compliant).
- Provide the necessary levels of isolation, availability and performance of provisioned services.
- Ability to dynamically control the amount of allocated resources in the DCRM, to scale them up and down, as well as monitor the deployed services.
- High availability, virtualized, cross-platform and scalability of IaaS SM.
- Non-disruptive, automated administrative tasks (e.g., service maintenance).
- Avoid non-authorized access to services.

Regarding the general design principles not covered at Cloud Hosting Architecture, they can be translated into the following key design goals:

- Support multitenancy architecture, in which a single instance of a software application serves multiple customers.
- Provide the necessary functionality to work with services, which could be composed for one or several virtual machines.
- OVF representation of services and OVA support.
- Ability to work with multiple hypervisor technologies.
- Ability to work in a federation environment through the separation of SOM from SLM.
- Provide monitoring information of the services deployed.
- Provide both synchronous and asynchronous notifications.

5.7 Detailed Specifications

5.7.1 Open API Specifications

Following is a list of Open Specifications linked to this Generic Enabler. Specifications labeled as "PRELIMINARY" are considered stable but subject to minor changes derived from lessons learned during last interactions of the development of a first reference implementation planned for the current Major Release of FI-WARE. Specifications labeled as "DRAFT" are planned for future Major Releases of FI-WARE but they are provided for the sake of future users.

- SM Open RESTful API Specification (PRELIMINARY)
- SM Open RESTful Extension API Specification (PRELIMINARY)

6 FIWARE ArchitectureDescription Cloud CloudEdge

You can find the content of this chapter as well in the wiki of fi-ware.

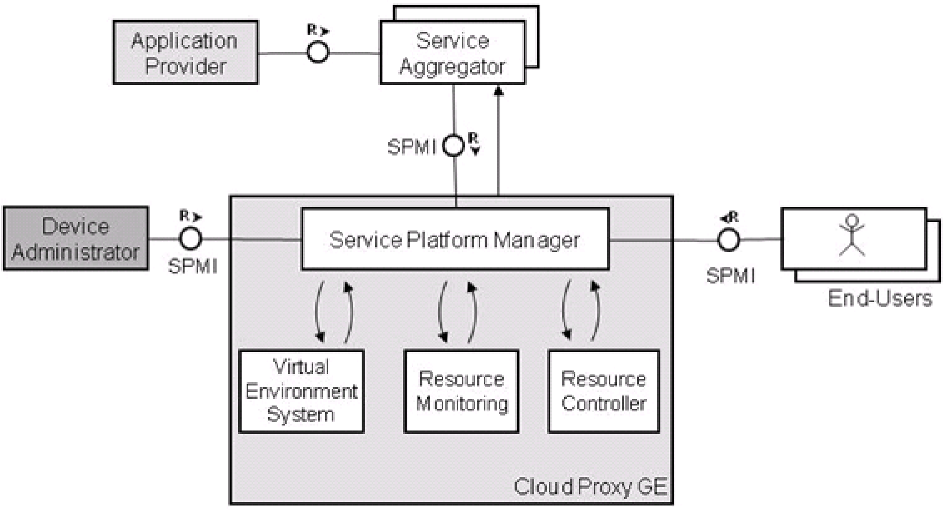
6.1 Overview

This specification describes the Cloud Edge GE, which is located beside the cloud, acting as the cloud agent in the end consumer's private network.

The Cloud Edge consists of equipment called "Cloud Proxy". Its main function is to offer local Services hosting capabilities as a complement of the standard Service hosting capabilities provided by traditional Cloud infrastructure.

Services hosted in such Cloud Proxy can benefit of this privileged position inside the private network area of a consumer to provide new enhanced Services. It can leverage on the proximity between the consumer and the Service itself to offer Services that require strong connectivity. It can also offer access to private capabilities that are hosted or accessible via the cloud proxy (for example sensors or private home network storage).

The following diagram shows the main components of the Cloud Proxy and the interactions with the different potential actors.



The Cloud Proxy offers a single public interface to manage the local service hosting capabilities: the Service Platform Management Interface (SPMI).

6.2 Main Concepts

6.2.1 Basic Concepts

To use the Service Platform Management interface, you should understand the following concept:

- Virtual Appliance: this entity represents the kind of Service that the cloud proxy is able to host: it is an operating system and application package together that can to run on top of the virtualized system supported by the Cloud Proxy.
- Image: this corresponds to a set of files that compose a virtual appliance and the associate metadata that describe requirement and configuration needed for installing this virtual Appliance
- Instance: in our context, it represents the virtual machine that runs the Service. It is an instantiation of a Virtual Appliance.

6.2.2 Actors and Roles

We consider four different types of actors that can interact directly or indirectly with the Cloud Proxy.

- Application Provider (AP): entity that creates applications for any users and deploys them on the Cloud.
- Service Aggregator (SA): entity that is in charge to manage a catalogue of applications that are compatible with a set of cloud proxies. Its role is to make sure that the proposed applications are sufficiently safe and secure to be deployed in any private consumer environment. If it gets all the required agreements and authorizations, it can deploy specific applications on a set of Cloud Proxies.
- Device Administrator (DA): root administrator of the Cloud Proxy Device. The DA has complete control over the cloud proxy, this includes:
    - The management of the administration rights of all the other users that can connect to the Cloud Proxy Service Platform Management system
    - The full control of any virtual appliances hosted on the cloud proxy
- End-User (EU): any user (person) that subscribes and consumes a service that runs on the Cloud Proxy. Register EUs have the ability to selects those applications that need to be installed and executed on their local Cloud Proxy.

6.2.3 Platform Components

The Cloud Proxy GE is composed of four main components: the Service Platform Manager, the Virtual Environment System, the Resource Monitoring and the Resource Controller.

The Service Platform Manager Interface is the REST interface that supports all features offer by the Cloud Proxy. It is the single point of connection for any client (DA, SA or EU) that needs to control and manage any Virtual Appliances. This module is also in charge of managing the users that are allowed to connect and manage the set of images available on the platform.

The Virtual Environment System is the module in charge of running the system-level virtualized commands. In our case, we select LXC (Linux Container) as this virtualization system. This choice is govern by the fact that the cloud proxy is targeted to run in any hardware environment, from PCs to small embedded systems (ex broadband access Gateway). Compared to other virtualization framework (ex KVM, XEN, VMWARE) LXC fits perfectly this requirement because it is light (very low overhead in term of memory and CPU), fast (ability to start or stop any Virtual Appliance in very few seconds), and not require any specific hardware (i.e. no specific processor instructions).

6.3 Examples of deployment scenarios

The service hosting capabilities are managed either directly by the end user consumer or by a third party that can deploy a catalogue of application on set of cloud proxies.

6.3.1 Individual on-demand deployment scenario

Summary: While browsing from his laptop at home, an end-user decides to install an application on his local Cloud Proxy. In this example, the installed application is a small web portal accessible from the home network.

1. A user connect to a web Application and the application propose to install an application into its private cloud proxy
2. The user accepts and provides to the Web App the description of the Cloud proxy environment. In response to that information, the Web application provides the URL of the compatible virtual appliance that could run on the consumer’s cloud proxy
3. The user registers this virtual appliance into the cloud proxy. The virtual appliance is downloaded on the system. If this virtual appliance is certified by a third party, the Cloud Proxy checks the validity of the certificate.
4. After reviewing the specific usage’s condition of this application, the user creates a virtual machine based on this virtual appliance and starts it.
5. As soon as the local Service is started on the cloud proxy, the web browser redirects the user to the local web provided by the new running application.

6.3.2 Large scale deployment scenario

Summary: a service aggregator deploys a catalogue of virtual appliances on a set of cloud proxies.

Prior to any transactions, the DA and the SA needs to find a formal authorisation that allows the Service Manager to use the Cloud Proxy owned by the DA.

1. An AP requests from the SA to deploy a specific service on a set of Cloud Proxies.
2. The SA checks in its database what are the Cloud Proxies currently available, and among them, select only the ones that can support the application of the AP.
3. As soon as the set of compatible Cloud Proxies is selected, the SA can start to deploy this application on those cloud proxies.
4. Then, each end-user can browse the catalogue of its own cloud proxy and decide to install it or not.

5. Once an instance of Virtual Appliance is created, any authorized user can start, stop or remove the created Service.

6.4 Main Interactions

In this section, the SPMI operations are described. These operations are classified in the following area:

- Platform features operation: these operations are used to provide generic information about the platform itself and the resources that can be shared or offered to virtual applications.
- Images Features operation: These operations are related to the management of images that are available on the cloud proxy.
- Instances Features operation: These operations are used to manage Instances that runs on cloud proxy.
- Users Features operation: These operation are used to manage the user's authentication and authorisation.
- Monitoring Feature operation: These operations are used to provide information about the state and the behaviour of any Instance.

6.4.1 Platform Features

- Platform version: Provide the current version of the SPM.
- Platform Description: Provide the general information that describes the platform in term of product, hardware and firmware. Used for by any client that needs to provide the right image for a specified Cloud Proxy.

6.4.2 User Features

- User Create: Create an account for a user. Any user that wants to interact directly with the Services Hosting Platform (install, uninstall a Virtual Appliance) need to be registered. This is performed by the DA, or any local administrator.
- User Attributes Update: Allow the authorized client to change attributes of a user account.
- User Delete: Delete a user account.

6.4.3 Images Features

- Image Register: Registrar a particular Image into the system so that it is available into the local application's catalogue.
- Image Detail: Provide detailed description of an image and resources it needs to run on the system.
- Image List: Provide the list of all the available images for a given user.
- Image Delete: Delete all the files related to an image and remove the application from the local catalogue.

6.4.4 Instances Features

- Instance Install: create a Virtual Appliance using a registered image
- Instance Detail: provide detailed description of a specified Virtual Appliance
- Instance Uninstall: delete a Virtual Appliance and free all the resources used by this instance.
- Instance List: Provide all available Virtual Appliances created on the Cloud Proxy.
- Instance Action: Perform a set of actions (start, stop, freeze, unfreeze, reboot) on Virtual Appliances.

6.4.5 Monitoring Features

- List metrics: List all the metrics and the associated type of statistics that are available on the platform.
- Get metrics statistics: Provide the collected measured values of a set of metrics for a specified instance.

7 FIWARE ArchitectureDescription Cloud ObjectStorage

You can find the content of this chapter as well in the wiki of fi-ware.

7.1 Copyright

7.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

7.3 Overview

Object Storage is one of the Generic Enablers within the Project. It offers persistent storage for digital objects, important cloud-based functionality that has been specifically requested by Use Cases. Objects can be files, databases or other datasets which need to be archived. These objects can be structured into a hierarchical way. The idea here is too have different objects sorted into groups. Within the context of Object Storage those are referred to as containers.

Containers and objects can have Metadata attached which give a more detailed overview of what the data represents. Similar to files a filesystem - objects in a Object storage belong to certain users (accounts). The following sections will give more details on these topics and will introduce the CDMI standard which is used within this Generic Enabler.

The users of the Object Storage Generic Enabler will be both the FI-WARE Cloud Instance Provider and the FI-WARE Cloud Instance User.

- Provider usage: The Provider can assume two roles: one as consumer of the service and another as its manager. From a consumer perspective, the Provider will use this system in order to store certain types of data. A good example of this is in the storage of monitoring, reporting and auditing data. That data could then be made available to the Users or not depending on the wants of the Provider. The Object storage service could also be used as a virtual machine staging area. This has two aspects, the Provider and User aspects. In the case of the Provider, a User will upload a virtual machine image to the object storage service and once received, the Provider will make this virtual machine image available for instantiation in order to satisfy a particular customized virtual machine request (the case here is that the virtual machine images that the Provider offers are not sufficient and the User wishes to supply their own). From a management perspective, the Provider will expect that the system will require as little maintenance as is possible. This entails that any:
    - stale data be purged,
    - deactivated accounts be removed,
    - corrupt data is replaced with a valid replica,
    - Issues are escalated to an automated service that will attempt to resolve them (if they cannot be resolved then notifications to the Provider should be sent),
    - relevant statistics should be available to support inspection of the system and the User's utilisation of the system,

  - additional requirements for hardware (storage capability) can be easily added to the system without any drop in service. This will allow the storage capacity to grow over time.

- User usage: The User will use the object storage service as a means to distribute static content rather than incur the additional load of serving static content from an application. Taking this approach allows the Provider to optimize the distribution of those files. The Provider can also use this as a building block to offer further content distribution network capabilities. The User could also use the object storage service as a means to supply a customized virtual machine that only they have access to (the storage is isolated by user). This would follow in a similar fashion to how customized virtual machine images are supplied on Amazon EC2.

7.4 Basic Concepts

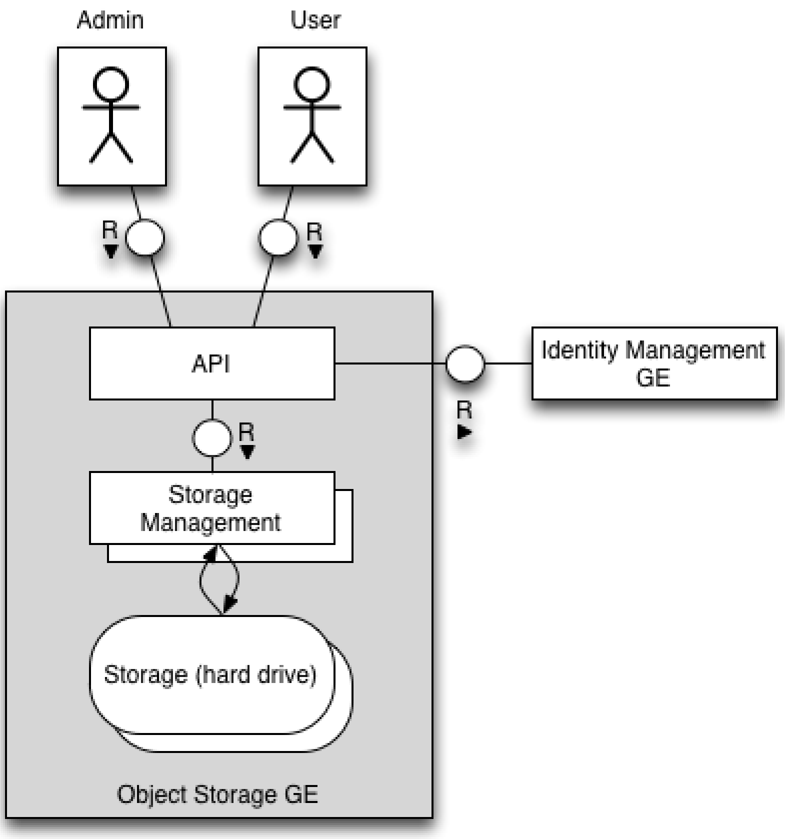
Implementations of the Object Storage Generic Enabler (GE) should provide a highly available, distributed and eventually consistent object store. The object store is a collection of objects that are structured in a simple hierarchy. The object store presents itself as a service that has multi-tenant capabilities such that the service can be offered to many users and organisations and that their data is safely partitioned. The object storage service does not have a traditional POSIX-type file system and as it has a simple hierarchical structure it really has little notion of true directory semantics. In fact, the Object Storage service allows for levels of hierarchy, where a container can reside within another container so allowing for more flexible organisation of data. Notably the Object Storage GE adopts the Cloud Data Management Interface (CDMI) specification.

The key abstract entities identified that need to be considered by this GE are:

- **Object**: opaque piece of data with associated meta-data. This directly maps to the CDMI object.
- **Container**: collection of objects with associated meta-data. This directly maps to the CDMI container.
- **Policy**: meta data associated with an object or container that dictates the use of the data by the object storage service provider. Policies will be expressed through the meta-data facilities offered by CDMI
- **Account**: a collection of containers assigned to a user. As the Object Storage GE uses the CDMI specification this will map to a container. In CDMI, a container can contain many other containers. A container cannot have more than one parent container.
- **User**: the actor accessing and managing the above entities through the GE's API. At a minimum the actors of end-user (human or external service) and administrator are considered. The security information related to User is managed by the Identity Management GE.

Access to and management of Object Storage entities is performed through the defined API. As standardised and open interfaces to GEs is an important aspect to consider in the specification of all GEs, the Cloud Data Management Interface (CDMI) has been chosen as the API specification for the Object Storage GE.

The overall *internal* architecture of the Object Storage GE is shown below:



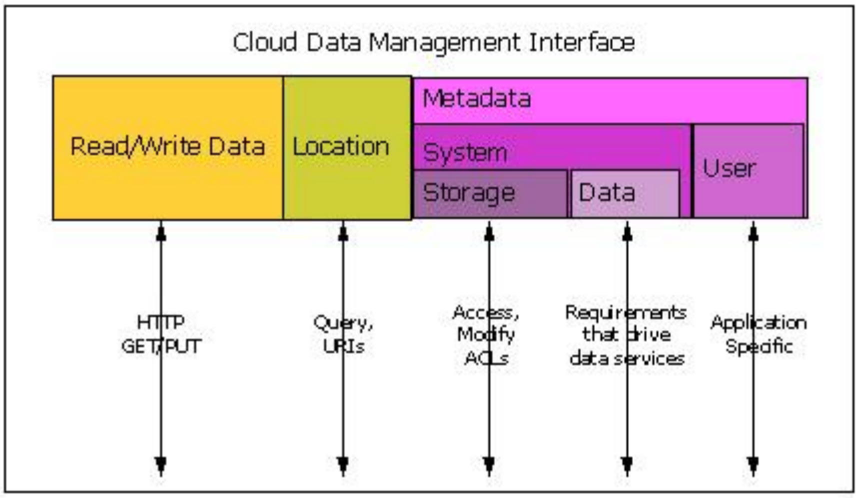
The main elements in the functional block diagram are as follows:

- **Admin**, **User**: These are Actors interacting with the service. Both might have different privileges associated to their accounts.
- **Identity Management GE**: This is entity which handels the privileges of the users.
- **API**: this entity is what exposes the interface of the Object Storage GE and allows the administrator and user user-types interact and manage their service instances.
- **Storage Management**: this is the entity that manages the resources associated with a user's service instance. Here entities such as containers, meta-data, objects and policies are managed.
- **Storage**: this is the storage device where the objects are physically persisted

7.4.1 Cloud Data Management Interface (CDMI)

CDMI is the de-facto standard for manipulating data in the cloud. Developed by the Storage Networking Industry Association (SNIA), it has now been designated by ISO as a formal international standard.

CDMI considers the management of both cloud storage systems and those of enterprise systems. It implements the SNIA's Storage Industry Resource Domain Model (SIRDM).



At the core of CDMI are the basic management operations of Create, Retrieve, Update and Delete. This functionality is exposed via a RESTful API that:

- Enables clients to discover capabilities of the object storage offering
- Manage containers and the objects that are placed within them
- Assigns and manipulates metadata to containers and objects

Meta-data is core to enabling richer management of data within CDMI. Meta-data can be associated with both system-managed objects and of course user-managed objects. Indeed, according to the specification "metadata is interpreted by the cloud offering as data requirements that control the operation of underlying data services for that data." This focus of CDMI makes it as a very flexible and suitable specification for the Object Storage GE, especially as providing quality of service and experience is a core focus in FI-ware. It is through CDMI's meta-data facilities that various data handling policies and QoS parameters can be specified and how the abstract entity of Policy can be represented.

The core entities in the CDMI model are:

- **Capabilities**: allows a client to discover features of the CDMI standard implemented by a provider. This is required for basic CDMI compliance.
- **Object**: opaque piece of data with associated meta-data. This is required for basic CDMI compliance. This CDMI entity will allow for the representation of Object.
- **Container**: collection of objects with associated meta-data. This is required for basic CDMI compliance. This CDMI entity will allow for the representation of Container and Account.
- **Domain**: represent the concept of administrative ownership of stored data within a CDMI storage system. This is an optional entity and is not required to be implemented for basic CDMI compliance. For the first release of the Object Storage GE this feature will not be supported.
- **Queue**: is a first-in, first-out (FIFO) access when storing and retrieving data. This is an optional entity and is not required to be implemented for basic CDMI compliance. For the first release of the Object Storage GE this feature will not be supported.

7.5 Main Interactions

For details on how a client interacts with the CDMI interface, please refer to the CDMI specification document. However below are two of the most basic but typical and perhaps important interactions with the interface.

7.5.1 Example: Create a Container

```
> PUT /dump/ HTTP/1.1
> Host: os.fi-ware.eu

< HTTP/1.1 201 Created
```

This creates a container at the specified location. The service responds with a HTTP 201, indicating success.

7.5.2 Example: Persisting an Object

```
> PUT /dump/simpleobject.json HTTP/1.1
> Host: os.fi-ware.eu
> Content-Type: application/json
> Content-Length: 29
>
> {"message": "Hello FIware!"}

< HTTP/1.1 201 Created
```

This creates an object which contains JSON content (the supplied content is in the body of the HTTP PUT). The service responds with a HTTP 201, indicating success.

Please note CDMI is an evolving standard. At the time of writing the approved version is CDMI 1.0.2. The actual raw CDMI calls detail the precise version of CDMI that the GE supports.

8 Data/Context Management Architecture

You can find the content of this chapter as well in the wiki of fi-ware.

8.1 Introduction

Fi-WARE will enable smarter, more customized/personalized and context-aware applications and services by the means of a set of assets able to gather, exchange, process and analyze massive data in a fast and efficient way.

Nowadays, several well-known free Internet services are based on business models that exploit massive data provided by end users. This data is exploited in advertising or offered to 3rd parties so that they can build innovative applications. Twitter, Facebook, Amazon, Google and many others are examples of this.

The Data/Context Management FI-WARE chapter aims at providing outperforming and platform-like GEs that will ease development and provision of innovative Applications that require management, processing and exploitation of context information as well as data streams in real-time and at massive scale. Combined with enablers coming from the Apps Chapters, Application Providers will be able to build innovative business models such as the ones described above and beyond.

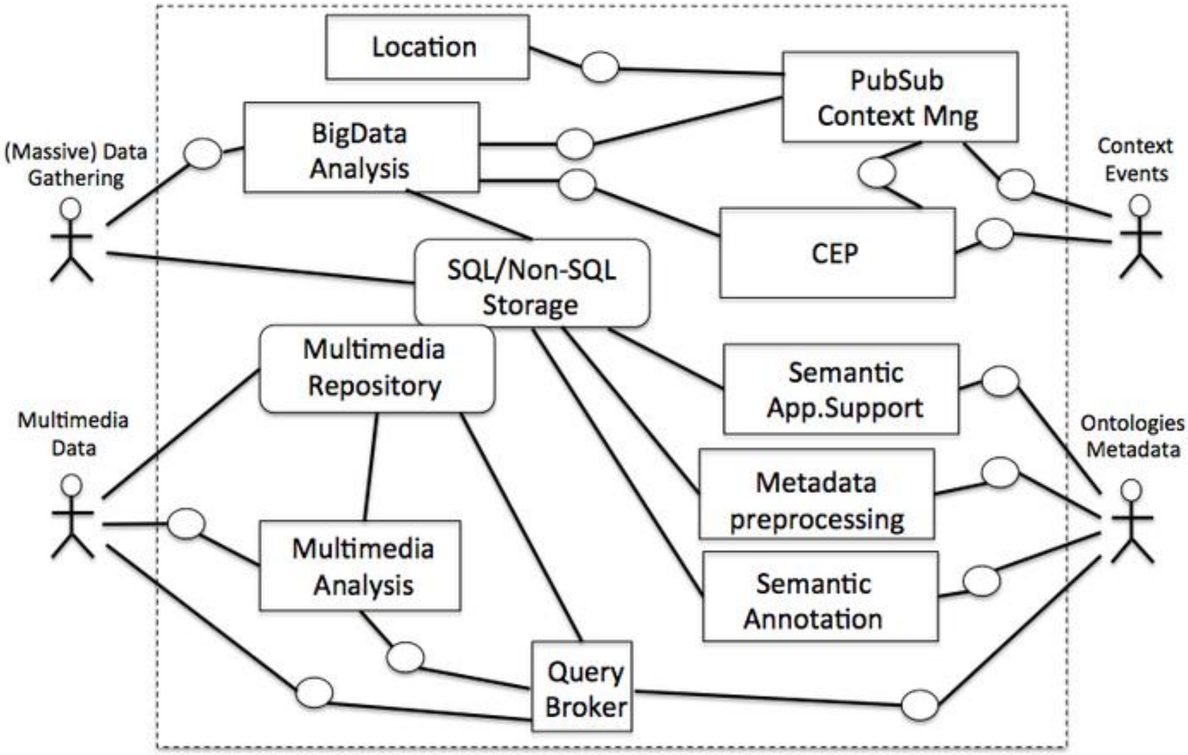
FI-WARE Data/Context Management GEs will enable to:

- Record, subscribe for being notified about and query for context information coming from different sources.
- Model changes in context as events that can be processed to detect complex situations that will lead to generation of actions or the generation of new context information (therefore, also treatable as events).
- Processing large amounts of context information in an aggregated way, using map&reduce techniques, in order to generate knowledge that may also lead to execution of actions and/or creation of new context information.
- Process data streams (particularly, multimedia video streams) coming from different sources in order to generate new data streams as well as context information that can be further exploited.
- Process metadata that may be linked to context information, using standard semantic support technologies.
- Manage some context information, such Location information, in a standardized way.

A cornerstone concept within this chapter is the structural definition of Data Elements enclosing its "Data Type", a number of "Data Element attributes" (which enclose the following: Name, Type, Value) and, optionally, a set of "Metadata Elements" (which have also in turn Data-like attributes: Name, Type, Value). However, this precise definition remains unbound to any specific type of representation and is able to represent "Context Elements" and "Events" as "Data Element" structures. More comprehensive information is available at Fi-WARE Data/Context Chapter vision.

8.2 Architecture Overview

The following diagram shows the main components (Generic Enablers) that comprise the first release of FI-WARE Data/Context chapter architecture.



8.2.1 Guiding Design Principles: Data, Context, Event and Event Object Concepts in FI-WARE

Data in FI-WARE refers to information that is produced, generated, collected or observed that may be relevant for processing, carrying out further analysis and knowledge extraction. A cornerstone concept in FI-WARE is that data elements are not bound to a specific format representation.

The structure associated to a **data element** is represented in Figure 1.

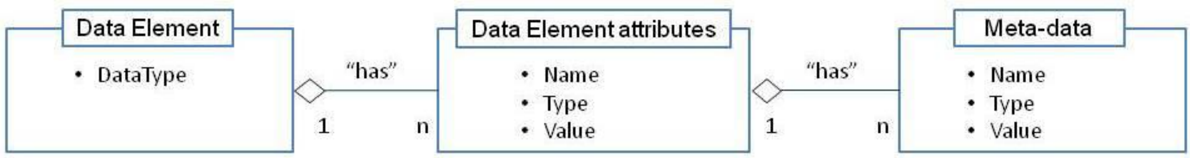


Figure 1: Data Element Structure

Data has associated a **data type** and a **value**. FI-WARE will support a set of built-in **basic data types** similar to those existing in most programming languages.

A **data element** refers to data whose value is defined as consisting of a sequence of one or more <name, type, value> triplets referred as **data element attributes**, where the type and value of each attribute is either mapped to a basic data type and a basic data value or mapped to the data type and value of another data element. Note that each data element has an associated data type in this formalism. This data type determines what concrete sequence of attributes characterizes the data element.

There may be **meta-data** (also referred as semantic data) linked to attributes in a data element. However, existence of meta-data linked to a data element attribute is optional.

Applications may assign an identifier to data elements in order to store them in a given **Data Storage**, e.g. a Data Base. Such identifier will not be considered part of the structure of the data element and the way it can be generated is out of the scope of this specification. Note that a given application may decide to use the value of some attribute linked to a data element as its identifier in a given Data Storage but, again, there is no identifier associated to the representation of a data element.

Context in FI-WARE is represented through **context elements**. A context element extends the concept of **data element** by associating an EntityId and EntityType to it, uniquely identifying the entity (which in turn may map to a group of entities) in the FI-WARE system to which the context element information refers. In addition, there may be some attributes as well as meta-data associated to attributes that we may define as mandatory for context elements as compared to data elements.

The structure of a context element is represented in Figure 2.

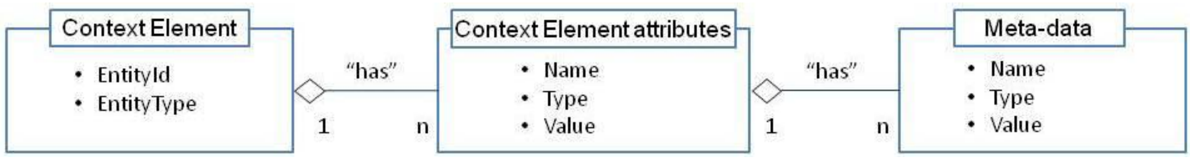


Figure 2: Context Element Structure Model

An **event** is an occurrence within a particular system or domain; it is something that has happened, or is contemplated as having happened in that domain. Events typically lead to creation of some data or context element, thus enabling that information describing or related to events can be handled by applications or event-aware FI-WARE GEs (e.g., the Publish/Subscribe Broker GE, when handling update/notifications, or the CEP GE). As an example, a sensor device may be measuring the temperature and pressure of a given boiler, sending a context element every five minutes associated to that entity (the boiler) that includes the value of these to attributes (temperature and pressure) or just the one that has changed. The creation and sending of the context element is an event, i.e., something that has occurred (the sensor device has sent new measures). As another example, a mobile handset may export attributes like "Operating System" or "Screen size". A given application may query for the value of these two attributes in order to adapt the content to be delivered to the device. As a result, the mobile handset creates and replies a context element back to the application. This response may be considered as well an event, i.e., something that has occurred (the mobile handset has replied to a request issued by an application).

Events get visible in the system by updates in Data/context elements and , therefore, it is common to refer to data/context elements related to events simply as "events". For convenience, we also may use the terms “data event” and “context event”. A “data event” refers to an event leading to creation of a data element, while a “context event” refers to an event leading to creation of a context element.

The word **event object** is used to mean a programming entity that represents such an occurrence (event) in a computing system [EPIA]. Events are represented as event objects within computing systems to distinguish them from other types of objects and to perform operations on them, also known as **event processing**.

In FI-WARE, event objects are created internally to some GEs like the Complex Event Processing GE or the Publish/Subscribe Broker GE. These event objects are defined as a data element (or a context element) representing an event to which a number of standard event object properties (similar to a header) are associated internally. These standard event object properties support certain event processing functions. The concrete set of standard event object properties in FI-WARE is still to be defined but we may anticipate that one of these properties would be the time at which the event object is detected by the GE (arrives to the GE). This will, for example, allow to support functions that can operate on events that exceed certain age in the system. Tools will be provided enabling applications or admin users to assign values to those event object properties based on values of data element attributes (e.g., source of the event or actual capture time). An event object may wrap different characteristics of the data element (i.e., DataType) or the context element (i.e., EntityId and EntityType).

8.2.2 Inter-dependencies and Interaction Between GEs

In a first stage, the functionalities set provided by each one of the GEs within the Data/Context architecture are considered in a standalone manner to ensure those key individual functionalities are delivered to other chapters and third parties exploiting FI-WARE core-platform. However, some tasks have been already started to identify scenarios involving smart combinations of Data/Context GEs and thus providing synergic features to other chapters and customer applications beyond FI-WARE framework. For instance, some three scenarios have been identified when combining BigData and CEP and some two scenarios are being discussed at this time regarding CEP and Publish/Subscribe GEs.

8.3 Architecture description of GEs

- FIWARE.ArchitectureDescription.Data.BigData
- FIWARE.ArchitectureDescription.Data.PubSub
- FIWARE.ArchitectureDescription.Data.CEP
- FIWARE.ArchitectureDescription.Data.Location
- FIWARE.ArchitectureDescription.Data.MetadataPreprocessing
- FIWARE.ArchitectureDescription.Data.CompressedDomainVideoAnalysis
- FIWARE.ArchitectureDescription.Data.QueryBroker
- FIWARE.ArchitectureDescription.Data.SemanticAnnotation
- FIWARE.ArchitectureDescription.Data.SemanticSupport

8.4 Interfaces to other Chapters

8.4.1 Security

Gathering, processing and exchanging information is subject to privacy concerns. Therefore, Data/Context GEs are expected to use Security APIs and interfaces providing Identity management, data authentication/encryption and confidentiality scope.

8.4.2 Internet of Things

In a first approach, several Data/Context GEs will be exploited by IoT chapter, providing a high differentiation and thus added-value of FI-WARE core platform for IoT businesses. For instance, Data/Chapter will provide a NGSI-capable PublishSubscribe engine coping with the management of notifications from/to users, applications and devices. Additionally, other Data/Context assets such as the CEP, enabling the advanced composition of events, or the BigData analysis are expected to be highly useful for this chapter.

Secondly, Data/Context framework knowledge and thus its added-value can be significantly improved by collecting contextual and metadata information of previously stored Data Elements in the BigData, Metadata management and Context Awareness Management platforms.

8.4.3 Applications and Services Ecosystem and Delivery Framework

In addition to Data/Context assets role in the overall Fi-WARE core platform architecture, several assets will be offered as standalone services with its own policies and business models (i.e "BigData as a Service"). Therefore, a set of synergic features resulting from the combination of Data/Context GEs, but also specific standalone services/assets, will be exposed in the FI-WARE business framework portfolio throughout this chapter APIs and interfaces.

8.4.4 Cloud Hosting

Data/Context assets deployment and operations will take benefit of the dynamic resources allocation and monitoring provided by the APIs and Interfaces in this chapter.

8.4.5 Interface to Networks and Devices

Some relevant contextual and metadata information relevant for third party organizations using FI-WARE such as availability, reachability and location of users, applications resources and devices can be actually provided by I2ND platform.

9 FIWARE ArchitectureDescription Data BigData

You can find the content of this chapter as well in the wiki of fi-ware.

9.1 Copyright

9.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

9.3 Overview

9.3.1 Introduction

Big Data Analysis is the process of using new tools to provide insights in to data previously considered too big or complex, given the current state of technology. The Big Data Analysis GE allows an end user or developer to unlock the information within the data. The components used in this GE are:

- Data Stream Analysis
- A high-performance distributed file system compliant with Apache Hadoop HDFS that provides a query interface for querying files
- A NoSQL document-orientated storage solution compatible with the MongoDB.

9.3.1.1 *Data Stream Analysis*

The Data Stream Analysis platform brings together a cluster of commodity computer servers or nodes to divide up and process a pre-programmed task.

Each node in the cluster has a Worker process that divides up the task amongst the available resources within the server. These tasks are programmed in modules designed to solve a specific problem. Modules are built as shared libraries allowing for changes to be made to the module so that it can be reloaded at run-time without interrupting the operation of the platform. These modules contain the necessary logic to process a given data stream and to represent the data internally to the platform. The analytical process is made up of a series of MapReduce steps, as outlined in Basic Concepts, to produce the desired result.

Communication between the nodes is handled via the network layer that ensures the data needed to process each task is available on every node. The network layer also compresses and optimizes the data flow to reduce latency when transferring data.

Data is uploaded into the cluster using the *delilah* client tool, provided by the platform, in addition this tool can be used to launch processing tasks and download result data. This tool is available in two forms, a command line client and Qt based graphical interface.

There is also a *delilah* client library (C++ API), also provided by the platform, which can be used for communicating between an application and the Data Stream Analysis platform (for streaming operations).

9.3.1.2 *High-performance distributed file-system*

This generic enabler requires the use of a high-performance distributed file system based on Apache's Hadoop HDFS. This framework allows for the distributed processing of large data sets (> 0.5 Terabytes) across clusters of computers using a simple programming model. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. In addition we will make use of an SQL-like interface what allows data exploration of data stored within the file-system.

9.3.1.3 *NoSQL document-orientated storage*

For data sets smaller than 0.5 Terabytes the GE will make use of a document orientated storage system based on NoSQL storage techniques. See the NoSQL section for information on how this differs with respect to Relational Database technologies. Due to there being no standard interface to interact with NoSQL database the interface is to be compatible with MongoDB. Additional interfaces to other NoSQL systems will be considered as needed. This component will be used to provide a high-availability store for querying data.

9.3.2 Target Usage

Big Data Crunching (also known as Big Data Batch Processing) is the technology used to process huge amounts of previously stored data in order to get relevant **insights** in scenarios where latency is not a highly relevant parameter. These insights take the form of newly generated data, which will be at disposal of applications using the same mechanisms through which initially stored data is available.

On the other hand, Big Data Stream Processing could be defined as the technology to process continuous unbounded and large streams of data extracting relevant insights on the go. This technology could be applied to scenarios where it is not necessary to store all incoming data or it has to be processed “on the go”, immediately after it becomes available. Additionally, this technology would be more suitable to big-data problems where low latency in generation of insights is expected. In this particular case, insights would be continuously generated, parallel to incoming data, allowing continuous estimations and predictions.

Finally, several Big Data Stream Processing technologies have been defined targeted to real-time generation of insights from a continuous stream of data received at a reasonable input rate.

Lately, a number of commercial solutions for the crunching problem have appeared; most of them based on open-source projects like Hadoop. On the other hand, several Real-Time Stream Processing engines can be found starting from those specialized in particular scenarios, like real-time user modeling for web-advertisement, to those intended to be more generic, like the ones based on Complex Event Processing techniques (see the Complex Event Processing section of High Level Vision). The approach taken in these two scenarios is radically different, not offering a single elegant solution that can be the most efficient and flexible one for Big Data Crunching scenario while at the same time is able to cope with Big Data Streaming scenarios.

The Big Data Analysis Support GE offers a continuous solution for both Big Data crunching and Big Data Streaming. A key characteristic of this GE is that it would present a unified set of tools and APIs allowing developers to program the analysis on large amount of data and extract relevant insights in both scenarios. Using this API, developers will be able to program Intelligent Services like the ones described in the Intelligent Services section of the High

Level Vision. These Intelligent Services will be plugged in the Big Data Analysis GE using a number of tools and APIs that this GE will support.

Input to the Big Data Analysis GE will be provided in two forms: as stored data so that analysis is carried out in batch mode or as a continuous stream of data so that analysis is carried out on-the-fly.

The first is adequate when latency is not a relevant parameter or additional data (not previously collected) is required for the process (i.e. access to auxiliary data on external databases, crawling of external sites, etc). The second is better suited in applications where lower latency is expected.

Algorithms developed using the API provided by the Big Data Analysis GE in oder to process data will be interchangeable between the batch and stream modes of operation. In other words, the API available for programming Intelligent Services will be the same in both modes.

In both cases, the focus of this enabler is in the "big data" consideration, that is, developers will be able to plug "intelligence" to the data-processing (batch or stream) without worrying about the parallelization/distribution or size/scalability of the problem. In the batch processing case, this means that the enabler should be able to scale with the size of the data-set and the complexity of the applied algorithms. On the other hand, in the stream mode, the enabler has to scale with both input rate and the size of the continuous updated analytics (usually called "state"). Note that other GEs in FI-WARE are more focused on real-time response of a continuous stream of events not making emphasis in the big-data consideration (see the Complex Event Processing section of High Level Vision).

9.3.3 Example Scenario

Imagine you are receiving a high volume stream of data that contains, amongst other things, a customer reference number (IMSI), a terminal ID (IMEI) and the ID of the cell tower they are currently connected to (CellID). As each mobile terminal moves throughout an operators area of coverage the stream will contain new entries with the IMSI, IMEI and CellID as they change between cell towers. This data stream can be joined / matched with the actual location (latitude, longuitde) of the cell tower to determine the approximate location of a given subscriber or terminal. This information is then stored in MongoDB, creating a profile for the subscriber that identifies where they live and work. This information can then be joined with an analysis of the movements of mobile phones that can detect traffic problems to notify people who travel a known route home that there has been an accident on the motorway/freeway so they can seek alternate route before leaving the office.

9.4 Basic Concepts

The two core technologies employed by the Big Data analysis GE are MapReduce and NoSQL. This section explains the basic concepts behind each one.

9.4.1 MapReduce

The core asset, SAMSON Platform, in the Big Data Analysis GE is based on improvements to the Map Reduce framework. MapReduce (MR) is a paradigm evolved from functional programming and applied to distributed systems. It was presented in 2004 by Google [BDA1]. It is meant for processing problems whose solution can be expressed in commutative and associative functions. In essence, MR offers an abstraction for processing large datasets on a set of machines, configured in a cluster. With this abstraction, the platform can easily solve the synchronization problem, freeing the developer thus of thinking about that issue. All data of these datasets is stored, processed and distributed in the form of key-value pairs, where both the key and the value can be of any data type.

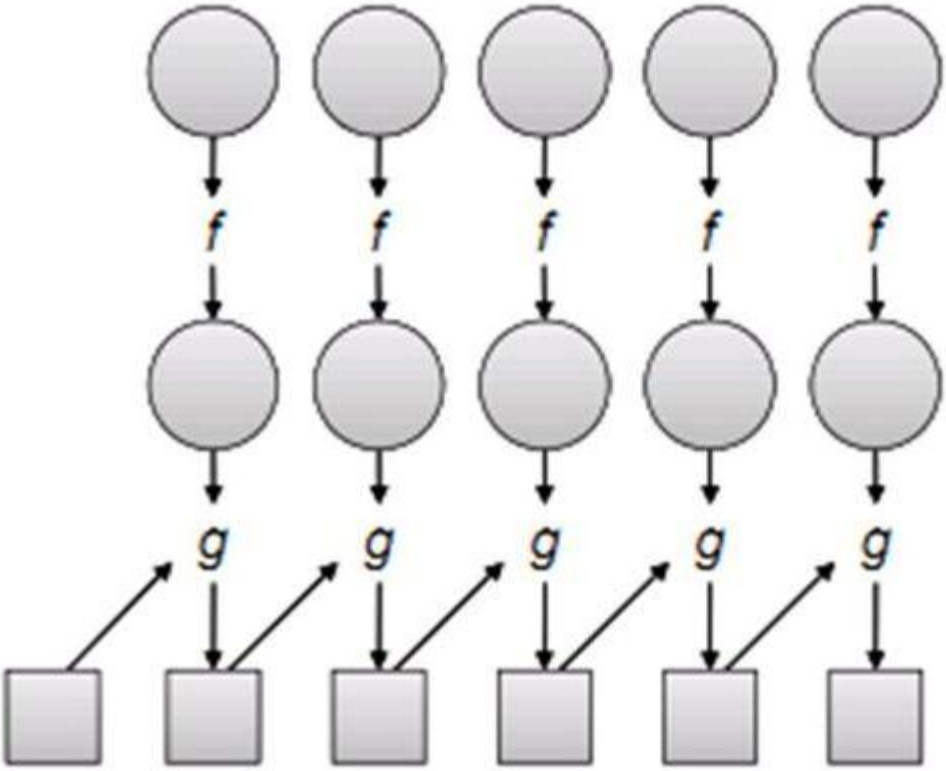


Figure BDA-1 – Functional programming diagram, with map (f) and fold (g) functions

From the field of functional programming, it is proved that any problem whose solution can be expressed in terms of commutative and associative functions, can be expressed in two types of functions: map (named also map in the MR paradigm) and fold (named reduce in the MR paradigm). Any job must be expressed as a sequence of these functions. These functions have a restriction: they operate on some input data, and produce a result without side effects, i.e. without modifying neither the input data nor any global state. This restriction is the key point to allow an easy parallelization.

Given a list of elements, map takes as an argument a function f (that takes a single argument) and applies it to all elements in a list (the top part of the Figure BDA-1), returning a list or results. The second step, fold, accumulates a new result by iterating through the elements in the result list. It takes three parameters: a base value, a list, and a function, g. Typically, map and fold are used in combination. The output of one function is the input of the next one (as functional programming avoids state and mutable data, all the computation must progress by passing results from one function to the next one), and this type of functions can be cascaded until finishing the job.

In the map type of function, a user-specified computation is applied over all input records in a dataset. As the result depends only on the input data, the task can be split among any number of instances (the mappers), each of them working on a subset of the input data, and can be distributed among any number of machines. These operations occur in parallel. Every key-value pair in the input data is processed, and they can produce none, one or multiple key-value pairs, with the same or different information. They yield intermediate output that is then dumped to the reduce functions.

The reduce phase has the function to aggregate the results disseminated in the map phase. In order to do so, all the results from all the mappers are sorted by the key element of the key-value pair, and the operation is distributed among a number of instances (the reducers, also running in parallel among the available machines). The platform guarantees that all the key-value pairs with the same key are presented to the same reducer. This phase has so the possibility to aggregate the information emitted in the map phase.

The job to be processed can be divided in any number of implementations of these two-phase cycles. The platform provides the framework to execute these operations distributed in parallel in a number of CPUs. The only point of synchronization is at the output of the map phase, were all the key-values must be available to be sorted and redistributed. This way, the developer has only to care about the implementation (according to the limitations of the paradigm) of the map and reduce functions, and the platform hides the complexity of data distribution and synchronization. Basically, the developer can access the combined resources (CPU, disk, memory) of the whole cluster, in a transparent way. The utility of the paradigm arises when dealing with big data problems, where a single machine has not enough memory to handle all the data, or its local disk would not be big and fast enough to cope with all the data.

The entire process can be presented in a simple, typical example: word frequency computing in a large set of documents. A simple word count algorithm in MapReduce is shown in Figure BDA 2. This algorithm counts the number of occurrences of every word in a text collection. Input key-value pairs take the form of (docid, doc) pairs stored on the distributed file system, where the former is a unique identifier for the document, and the latter is the content of the document. The mapper takes an input key-value pair, tokenizes the document, and emits an intermediate key-value pair for every word. The key would be a string (the word itself) while the value is the count of the occurrences of the word (an integer). In an initial approximation, it will be a “1” (denoting that we’ve seen the word once). The MapReduce execution framework guarantees that all values associated with the same key are brought together in the reducer. Therefore, the reducer simply needs to sum up all counts (ones) associated with each word, and to emit final key-value pairs with the word as the key, and the count as the value.

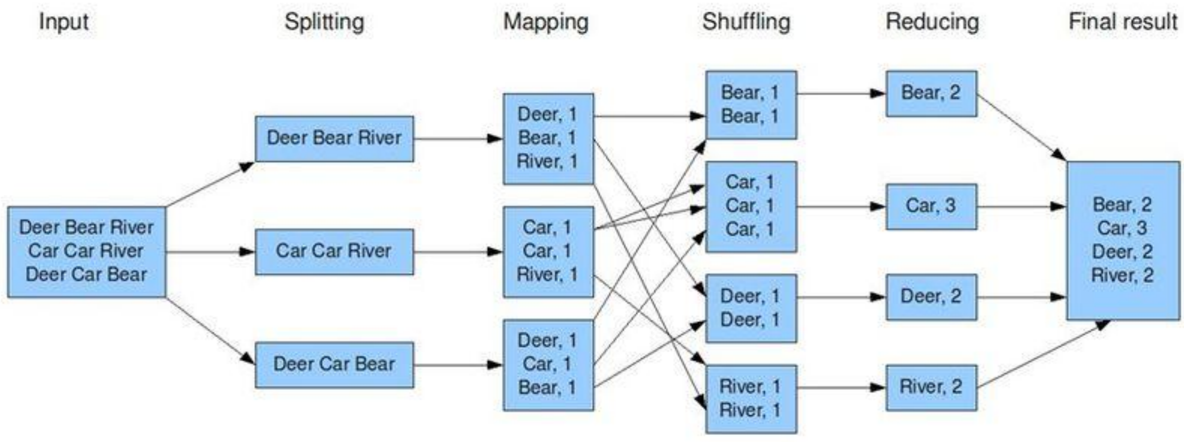


Figure BDA-2 – Word count algorithm implementation in MapReduce

This paradigm has had a number of different implementations: the already presented by Google, with a patent [BDA2] , the open source project Apache Hadoop [BDA3], that is the most prominent and widely used implementation, and a number of implementations of the same concept: Sector/Sphere [BDA4][BDA5] Microsoft has also developed a framework for parallel computing, Dryad [BDA6], which is a superset of MapReduce.

These implementations have been developed to solve a number of problems (task scheduling, scalability, fault tolerance...). One such problem is how to ensure that every task will have the input data available as soon as it is needed, without making network and disk input/output the system bottleneck (a difficulty inherent in big-data problems).

Most of these implementations (Google, Hadoop, Sphere, Dryad...) rely on a distributed file-system [BDA7][BDA8] for data management.

Data files are split in large chunks (e.g. 64MB), and these chunks are stored and replicated to a number of data nodes. Tables keep track on how data files are split and where the replica for each chunk resides. When scheduling a task, the distributed file system can be queried to determine the node that has the required data to fulfill the task. The node that has the data (or one nearby) is selected to execute the operation, reducing network traffic.

The main problem of this model is the increased latency. Data can be distributed and processed in a very large number of machines, and synchronization is provided by the platform in a transparent way to the developer. But this ease of use has a price: no reduce operation can start until all the map operations have finished and their results are placed on the distributed file-system. These limitations increase the response time, and this response time limits the type of solutions where a “standard” MR solution can be applied when requiring time-critical responses.

Due to these limitations we are looking for a solution that is able to integrate data from an external source such as Hadoop HDFS or MongoDB with real-time stream big data processing. As stated above existing solutions struggle to provide this integration and responsiveness needed in real world applications.

9.4.2 NoSQL

Coined in the late 90's the term NoSQL represents database storage technologies that eschew relational database storage systems such as Oracle or MySQL. NoSQL emerged from a need to overcome the limitations of the relational model when working with large quantities of data, typically in unstructured form. Initially, as a reaction to these limitations, NoSQL was considered, as the name might be interpreted to be an opposition movement to using SQL based storage systems. However as it's seen that SQL and NoSQL systems often co-exist and complement each other the term "NoSQL" has morphed to mean "Not only SQL".

With a change in usage focus new applications, in particular those for the web, are no longer read orientated rather they are tending to read/write if not write heavy. Traditional SQL based systems struggle with this when demand scales up often enough the underlying data store cannot do the same, without incurring downtime. These systems are based on the ACID ("Atomic, Consistent, Isolated, Durable") principle:

- Atomic - either a transaction succeeds or not
- Consistent - data needs to be in a consistent state
- Isolated - one transaction cannot interfere with another
- Durable - data persists once committed even after a restart or a power-loss

In systems that need to scale out it's not always possible to guarantee that the data being read is consistent or durable. For example when shopping during times of high demand, say Christmas, via the web for a particular item, it is more important that the web site remains responsive, so as not to dissuade customers, rather than the inventory count for every item is kept up to date. Over time item counts will get refreshed as more hardware is brought on stream to be able to cope with the demand.

NoSQL systems are designed around on Brewers CAP Theorem [BDA10][BDA11], that says if a distributed system wants Consistency, Availability and Partition Tolerance, it can only pick two. Rather than NoSQL striving for ACID compliance, NoSQL systems are said to aim for eventual consistency (BASE - Basic Availability, Soft and Eventual Consistency [BDA12]). Such that over time the data within the system becomes consistent via consolidation, in the same way accountants close their books at the end of an accounting period to provide an accurate state of accounts.

The different types of NoSQL database are

- Column Store - Data storage is orientated to the column rather than the row as it is with traditional DBMS engines, favouring aggregate operations on columns typically used in data warehousing. Example implementations: Hadoop HBase and Google's BigTable.
- Key Value Store - A schema-less storage system, data is stored in key-value pairs. Data is accessed via a hash table using the unique key. Example implementation:[BDA9]
- Document Store - Similar to Key Value storage, document storage works with semi-structured data that contain a collection of key-value pairs. Unlike key-value storage these documents can contain child elements that store relevant knowledge to that particular document. Unlike in traditional DBMS engines, document orientated storage does not require that every document contain all the fields if no information is there for that particular document. Example implementations: [BDA13][BDA15]
- Graph Database - Using a graph structure, data about an entity is stored within a node, relationships between the nodes are defined in the edges that interconnect the nodes. This allows for lookups which utilize associative datasets as the information that relates to any given node is already present, eliminating the need to perform any joins. Example implementations: [BDA15]

Given that the structure of the data that is to be stored is not known, the preferred solution is to use a document storage engine. This will allow the Big Data Analysis GE to retrieve and store most types of data without compromising its format.

9.5 Big Data Analysis Generic Architecture

From the users perspective there are 3 main areas of interaction with the Big Data Platform. These are:

- Data Stream
- Module Programming
- Analytical scripting

The Data Streaming platform is designed to be executed on a cluster of computing nodes. Typically using commodity hardware, each node should have the same amount of physical RAM as disk storage. As can been seen in Figure BDA-3 below, the cluster of nodes communicate via a common network, typically dedicated to the cluster. One node is nominated to be the controller. This node keeps track of the state of the operations running in the platform. It journals operations performed in a task allowing to rollback any changes. The delilah client tool connects to the controller for monitoring and control of the Data Streaming platform. In each node a worker task is present that does the actual work. This task manages the operations/tasks and distribution of the data between each node. The worker task determines whether the data it currently holds should be kept in memory, allowing the node to make best use of the resources it has.

The data processed by the platform can come from a number of sources: a data stream (e.g. company network traffic), flat files stored locally or in a distributed file-system (e.g. Apache Hadoop HDFS) or from an external document store database.

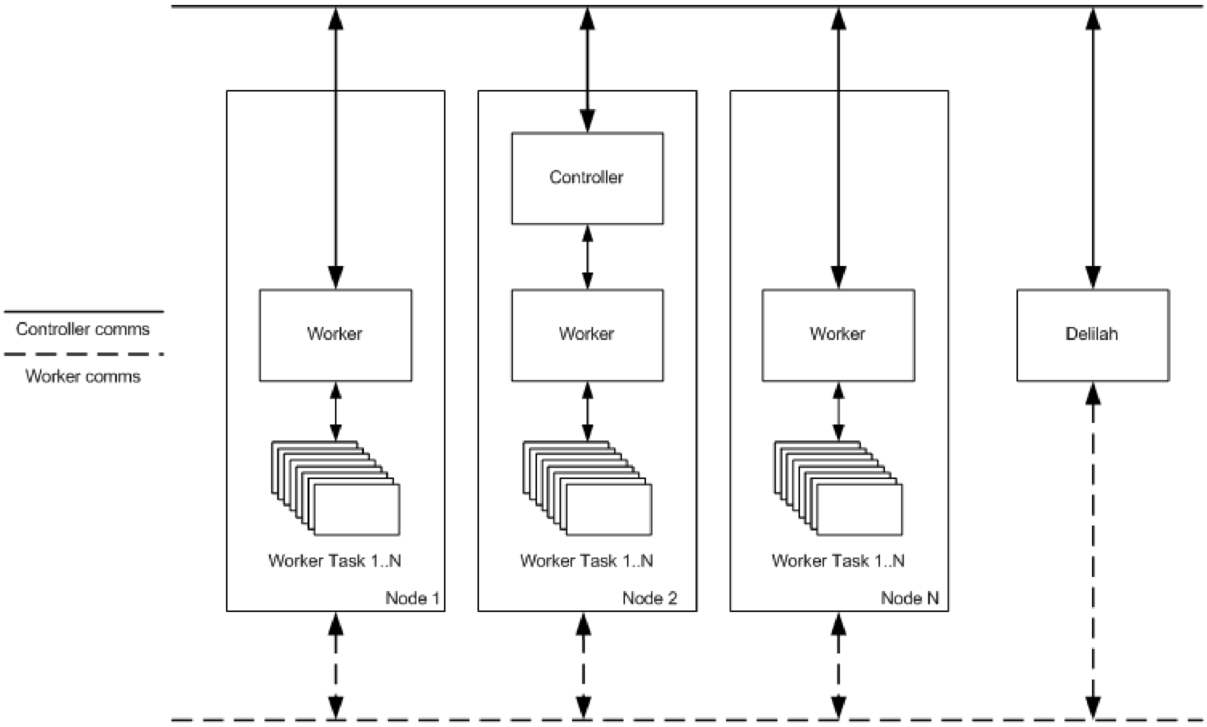


Figure BDA-3 - Platform overview

When receiving data, the platform uses pre-programmed modules to process the data in order to obtain the desired output. This data can come from a live source, such as network probes from the cellular phone network or from a database source such as a table that contains a list of clients and their account details. An overview of this can be seen in Figure BDA-4.

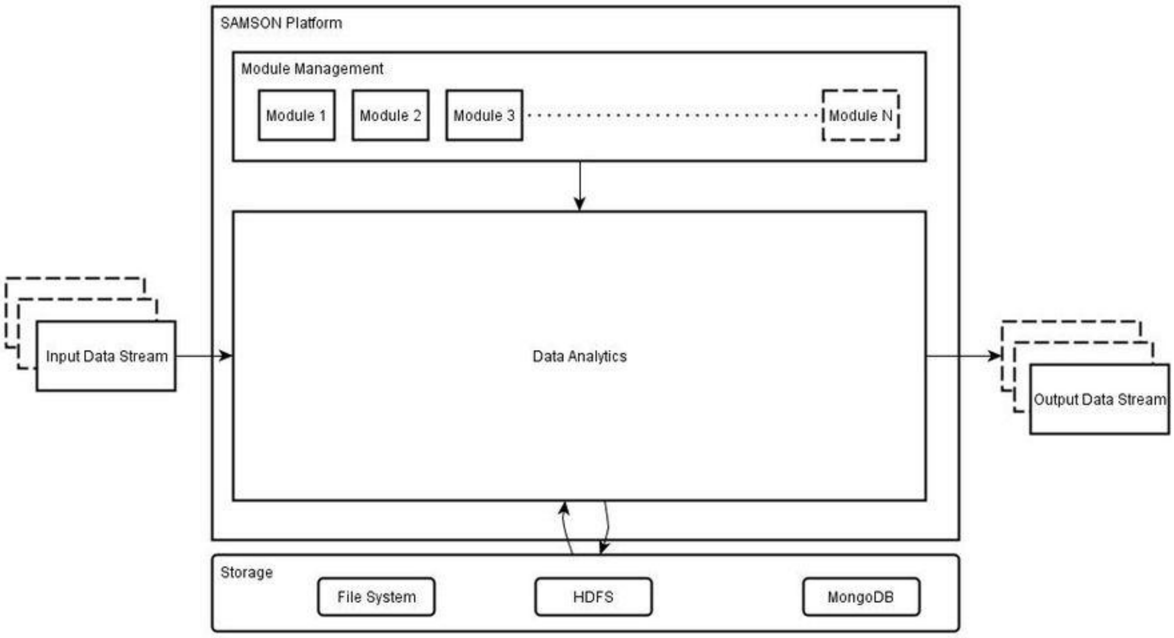


Figure BDA-4 - Overall architecture view

The sequence of operations or tasks is programmed via a script (see Analytical scripting). That script contains the list of queues / data sources that are to be used and the operations that are to be performed on the queues. Integration with an external source such as a distributed file system or database occurs within an operation. The script contains the necessary details (i.e. location of the data, format etc.) needed to load the data into the platform. This external data is loaded into a queue which in turn be utilized by any defined operation within the platform. The resulting output or state from each operation is then placed in another queue for the next step in the task list to use. This process is repeated until the data has finished being processed. At that point the data can be kept in a queue, stored in a database or streamed out for another GE to process the result. Likewise with input data from the file-system or a database, the script can direct the output from processing back to the file system or to the database.

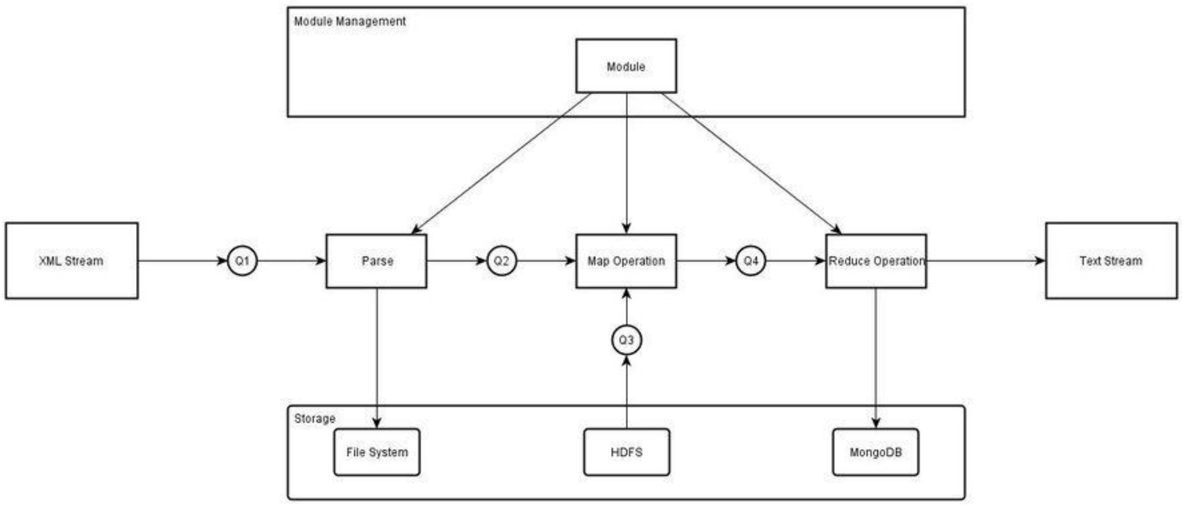


Figure BDA-5 - Example analytical flow

For example imagine we want to identify what websites are popular for our customer base to be able to suggest, in a future marketing campaign, websites that might be of interest. In Figure BDA-5, We receive a stream of XML data that contains the web URLs being browsed along with the IP address and the time being spent on each page. For this analytical process all we need to know is the URL and the IP address. Before we can do anything with this data it first must be loaded in to a queue. Once there we use a parse operation that understands the XML DTD and can extract the URL and IP address. This information is stored in a second queue for later processing, at the same time for auditing purposes we place a copy of the URL and IP in a flat file on disk. The next phase in the process is to join the IP address we received in the stream with the account details. In this contrived example we periodically load the customer details and RADIUS logs from an external source, such as HDFS. In SAMSON, we join these two sources of information together to convert the IP address into a customer number allowing us to uniquely identify a customer. The result of this join is placed in to another queue for reduction. The last phase in the operation list is to take the incoming customer and URL list and to count, for a given customer, the number of occurrences of each URL. This information can then be streamed out to an application that takes the data and stores it in a database for later processing in the upcoming marketing campaign.

9.6 Main Interactions

9.6.1 Modules and Interfaces

This section covers the description of the software modules and interfaces of the BigData Analysis GE.

9.6.2 Data Stream Interfaces

As mentioned above the Big Data Analysis GE is designed to provide analysis of ***streams*** of data as well as pre-loaded data sets that are stored either in the file system or in a database system.

9.6.2.1 *Inject data into the platform*

When needing to process data from an external source there are two ways this can be done. Firstly using a program that can connect to an external data source or receive data from the source and then insert the incoming data into a ***queue/buffer***. This program can be developed specifically for the incoming data source or can be injected via a generic program that uses a named pipe to forward on the data. The second way to load data is via a step in the analytical script that loads the data from the file-system or database into the platform.

In each case the ***format*** of the data needs to be known beforehand so that it can be parsed and the relevant information can be extracted for later processing. The same is true for ***streams*** of binary data as well as fixed format/structured text data such as CSV, XML or JSON. This however is dealt with in module programming

9.6.2.2 *Extract data from the platform*

Data from any stage in the analytical process can be sent to an external source, e.g. another GE that is able to process the output, or stored locally in a database (either MongoDB or HDFS). As with injecting data there are two ways to extract the data. Using a program to stream out data from a known queue or adding a step in the analytical script that exports the data to files or the database. At the current time it is expected that the user of the platform will provide the necessary tools to visualize the generated results. The platform may provide a GUI interface to enable data visualization in the future.

9.6.3 Analysis Module Programming

The process of developing a module is summarized by the following diagram:

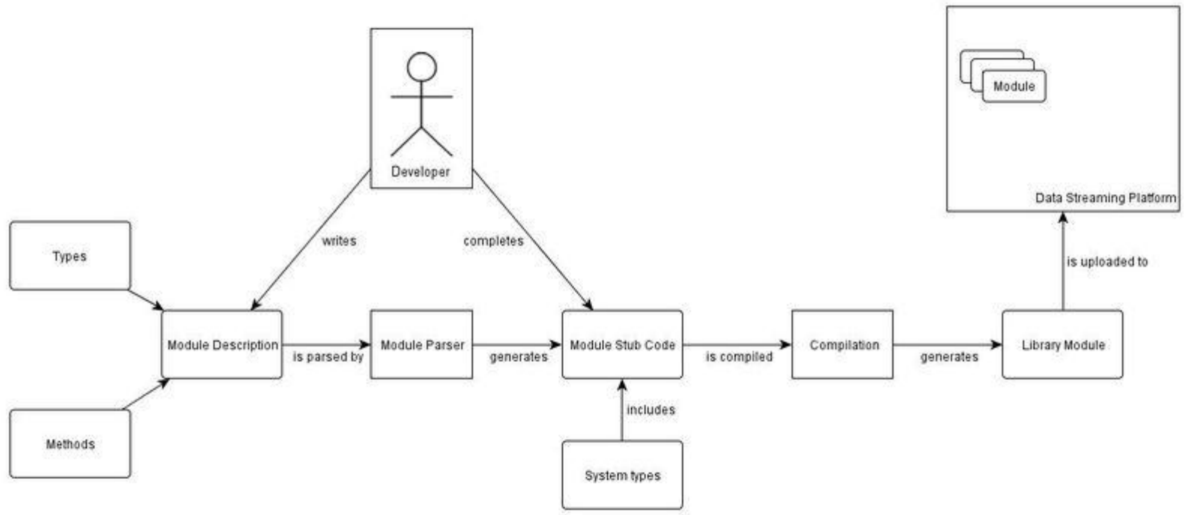


Figure BDA 6 - Developing a module

The developer defines the ***types*** and ***methods*** for the module in a file. This file is then parsed using a ***module parser*** to generate dummy stub code. The developer then writes the implementation of the ***methods*** needed to analyse the data, including, if needed types from other modules such as the ***system module***. Once complete this code is ***compiled*** to generate a ***module library*** which is uploaded into the ***Data streaming platform***.

9.6.3.1 *System Module*

This module defines the basic data-types that are available for use within other modules. These definitions include the functions needed to serialize and parse these types. The types provided by the ***system*** module match those that can be found in C or C++ along with some addition custom types for existing problem domains:

- Integer
- String
- Floating Point
- Date/Time

9.6.3.2 *Module Development*

Extending the analytical functionality of the Big Data Analysis GE requires the development of modules in C++. A module is made up of data types to handle the data to be processed as well as ***operations*** to process that data. The modules are linked as a ***dynamic library*** allowing updates to the platform at run time without needing to restart the platform.

New modules are initialized using a ***boot-strap*** program that generates the necessary files needed to start developing a module. This program performs some basic checks to see if that module already exists within the platform since each ***module*** must use a unique name across the ***platform***.

Configuration File

Once a new module has been initialized, the next step in developing the module is a high level definition of the ***types*** and ***operations*** being developed. These definitions are places in a ***Module*** file for later parsing.

Data structures can be built by building a collection of existing types, either from the ***system*** module or from other modules that have been developed. For example, imagine we have a parser that extracts the URL and the client IP that requested the URL from our web logs. We could use something like this to represent the data types:

```
data webLog
{
   // Website requested by the user
   system.String URL;
   // User IP address
   web.IP ipAddress;
   // Timestamp for the request
   system.TimeUnix timeStamp;
}
```

In the above example we are using ***system.String***, a basic String type, to store the URL accessed by the client, ***web.IP*** type (something that we have defined elsewhere in this example module) to store the IP address and ***system.TimeUnix*** to store the timestamp of the URL request.

Once we have our types defined we need to define the operations that work on those types. The different operations that can be defined are:

- ***generator*** - to generate test data
- ***map*** - to separate the data that matches a given criteria
- ***parser*** - to parse data from an input source or ***queue***
- ***parserOut*** - Send the parsed data to an external source
- ***reduce*** - aggregate the data emitted from the map phase

Along with the type of operation to be performed we need to specify the data types being worked for input or output. For example we would define our example URL parser as:

```


parser parse_web_logs
{
    out web.webLog
}
```

Note that there is no input data type since it's format is unknown prior to being parsed. In the above ***operation*** definition we assign that the web.webLog type (as defined earlier) as the output from this ***operation***.

Once the ***module*** definition file is complete it needs to be processed using the ***module parser*** to generate the necessary code stubs to implement the logic to parse the data. A ***header*** file is generated for each operation defined in the ***module*** file. So with our example *parse_web_logs* a *parse_web_logs.h* is created with the necessary code need start adding the logic to ***parse*** the web logs.

Operation code

The code to perform each particular operation is placed in the header file generated by the ***module parser***. Once complete, the code is compiled using the generated ***makefile*** and installed into the platform.

9.6.3.3 *Analytical scripting*

Once you have a programmed module ready to deploy the next step is to define the sequence of operations needed to achieve a given task. Currently this is done via a script that is either programmed into the module directly or by uploading into the SAMSON platform. The script contains all the necessary commands to process the data streamed or loaded into the Big Data Analysis GE.

The script contains a list of ***operations***, their parameters (inputs and outputs) and the ***queues*** used to exchange or store data. In ***operations*** that store data to an external store, such as MongoDB, ***variables/properties*** can be set that control how an operation behaves. For example these properties can be used to specify the MongoDB server to connect to, the port and the ***collection*** where data is to be read from or stored.

When ready the script is loaded into the SAMSON platform via a command line utility, activating the streaming process. Once data starts arriving in the ***input queues*** the operations continue to process this data whilst it is being sent to the GE.

Alternatively this script can be built into the module and executed from within ***the command tool***:

```
script init_stream_operations
{
    code
    {
        # Queues;
        alias [input]   01.input;
        alias [words]   99.hit.01.input;

        # Operations;
        alias [parse_web]   01.parse_web;

        # Definitions;
        add_stream_operation  [parse_web]  web.parse_web_logs
[input] [words];
    }

    helpLine "Init stream operations"
}
```

9.6.3.4 *Command Line interface*

The client line client is the primary user interface for executing solutions within the SAMSON platform. It can be run from a machine inside the cluster, or from any machine outside of the cluster (as long as the client can connect to the cluster).

9.6.3.5 *General Commands*

- ls_operations - Provide a list of the currently programmed operations
- ls_datas - Provide a list of the available data types
- reload - Reload the modules
- set - Set a variable
- unset - Unset a variable
- ls_local - Show a list of current directory with relevant information about local data-sets
- rm_local - Remove a local directory and all its contents
- ls_operation_rates - Get a list of statistics about the currently running operations
- ls_modules - List the available modules
- trace - Enable / disable tracing

9.6.3.6 *Stream Processing Commands*

- run_stream_operation - Execute a named operation on given queues
- init_stream - Initialize stream processing from a script
- push - Upload/push the contents of a local file to a queue
- pop - Download/pop the contents of queue to a local file. Use samsonCat to view the contents.
- add_stream_operation - Add an operation to automatically process data from input queues to output queues
- rm_stream_operation - Delete/remove a previously defined operation
- set_stream_operation_property - Define a parameter/property for use within an operation, e.g. the database server, name and login details.
- rm_queue - Delete/remove a queue
- cp_queue - Copy a queue to a new queue name
- ls_queues - Provide a list of defined queues
- ps_stream - Provide a list of currently running stream processes
- ls_stream_operations - Provide a list of currently running stream operations

9.6.3.7 *RESTful API*

In addition to the command line interface a RESTful API is available. This API provides the ability to query the following parts of the Big Data Analysis:

Obtain the current status of the platform

Determine if the platform is running or not and provide information regarding the current configuration of the nodes

Obtain a list of the current active operations

Provide a list of the current operations in effect on the platform with the number of items processed along with the current throughput rates.

Obtain a list of the available modules

Provide a list of the modules installed into the platform with the methods that have been exposed.

Obtain a list of the current queues

Provide a list of the queues defined in the platform with amount of data contained within each queue

Obtain the output state of a given operation

Provide the ability to query an operation to obtain the result for some criteria, e.g. last cell tower used by a mobile phone.

Define a sequence of operations

Provide the ability to define a sequence of operations that can be used to analyse data from different sources

9.7 Basic Design Principles

- The Big Data GE is designed to deploy analytical solutions against a cluster of commodity hardware without needing to know how to distribute the work.
- The SAMSON platform is designed to accept and process high volumes of data so that new insights can be gained from the data source
- The SAMSON platform is designed to store analytical results in an external store such as a database system
- The GE is designed to be extended to address new problem domains, allowing for the reuse of logic from existing solutions that have been developed in the process.
- The GE is designed to be as agnostic as possible towards the data it needs to process so as to provide a flexible analytical platform.

9.8 References

BDA1 MapReduce: Simplified Data Processing on Large Clusters

BDA2 System and method for efficient large-scale data processing

BDA3 http://hadoop.apache.org/

BDA4 http://sector.sourceforge.net/

BDA5 Sector and Sphere: the design and implementation of a high-performance data cloud

BDA6 http://research.microsoft.com/en-us/projects/Dryad/

BDA7 The Google File System

BDA8 HDFS Architecture Guide

BDA9 Dynamo: Amazon's Highly Available Key-value Store

BDA10 Cap Theorem

BDA11 Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services

BDA12 Eventually Consistent - Dr Werner Vogels

BDA13 http://couchdb.apache.org/

BDA14 http://www.mongodb.org/

BDA15 http://neo4j.org/

10 FIWARE ArchitectureDescription Data PubSub

You can find the content of this chapter as well in the wiki of fi-ware.

10.1 Copyright

10.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

10.3 Overview

10.3.1 Introduction to the (Publish/Subscribe) Context Broker GE

Generally speaking, the Context Broker GE will enable publication of context information by entities, referred as Context Producers, so that published context information becomes available to other entities, referred as Context Consumers, which are interested in processing the published context information. Applications or even other GEs in the FI-WARE platform may play the role of Context Producers, Context Consumers or both. Events in FI-WARE based systems refer to something that has happened, or is contemplated as having happened. As such, they map to updates or notification on updates on context information that can be handled by applications or FI-WARE GEs.

A fundamental principle supported by the Context Broker GE is that of achieving a total decoupling between Context Producers and Context Consumers. On one hand, this means that Context Producers publish context information without knowing which Context Consumers will consume published context information; therefore they don't need to be connected to them. On the other hand, Context Consumers consume context information of their interest, without this meaning they know which Context Producer has published a particular event: they are just interested in the context information itself but not in who generated it.

Another fundamental principle is that of agnosticism with respect to the information model, i.e. the data types associated to attributes of entities in the system or even the way those entities are classified and identified.

10.3.2 Target usage

The Publish/Subscribe Context Broker GE, or Context Broker GE for short, is intended to play a key role in development of context-aware applications. It can be used to provide and consume information about any sort of entities that may be relevant to an application, including users, group of users, representation of physical objects in the real world or even more abstract entities such as application processes. Exploiting that information, generally referred as context information, applications can sense and react to changes on the state of entities thus providing an enhanced user experience or support smarter processes.

The producer/consumer decoupling and the information agnosticism principles supported by this GE make it an excellent bridge enabling external applications to manage context information related to the Internet of the Things in a simpler way, hiding the complexity of gathering measures from IoT resources (sensors) that might be distributed or involving multiple low-level communication protocols.

A rather interesting and promising usage of the Context Broker GE is that of making an instance available to multiple applications. That way, some applications may provide information that others may exploit without knowing which are they a priori. As an example, applications sponsored by city authorities may export data about the city following an open data approach. Data would be published through an instance of the Context Broker GE so that third-party applications may consume and exploit it in order to bring innovative services to the citizens and, why not, make some money out of it. As another example, a FI-WARE Instance Provider may provide a Context Broker GE “as a Service” so that applications consuming context information may pay for using context information. The FI-WARE Instance Provider may in turn implement some revenue-share system with application providers that publish data. This could be the basis for implementing a true marketplace of data.

10.3.3 Example Scenarios

The number of potential context sources permanently connected through 3G links, e.g. mobile user terminals, embedded sensors, microphones and cameras, is expected to increase significantly. By processing and inferring this raw information, a variety of useful information will be available in future communication and content management systems. It is likely for smart spaces to grow from smart homes/offices to urban smart spaces in which plenty of artifacts and people are interconnected over distance. This will enable all sorts of innovative interactive pervasive applications as perceived by Weiser [1]

Few examples of how the publish/subscribe to certain information may improve the user experience and enrich a service are given below.

The Figure 1 below shows a context-aware advertising service (shown in high left corner, described in [3]) sending an invitation and a coupon to a customer in proximity to a boutique. Also the goods are chosen for that customer based on her/his preferences and previous acquisition. Therefore advertisement messages traffic is significantly reducing targeting only potential clients, and the clients' user experience is not suffering from a "broadcast" advertisement messages with zero or very low value. This scenario is possible because the customer or a service provider on her/his behalf subscribes to a content (advertisement message and coupon) in certain conditions (close to the boutique and matching the preferences).

Another example might be context-aware content exchange also shown in Figure 1 (high right corner), where a customer sees only the content places by her social friend (a friend from SN) and this content is related to her location as well as originally that content was "placed" in that location. She has subscribed to be informed with recommendations related to her location based on her preferences and from her friend from Social Networks (SNs).

The bottom part of the Figure 1 shows a logic architecture of the Context - Publish/Subscribe Broker GE with its main components and the basic information it handles to enrich traditional services of Mobile Advertisement and Content Share.

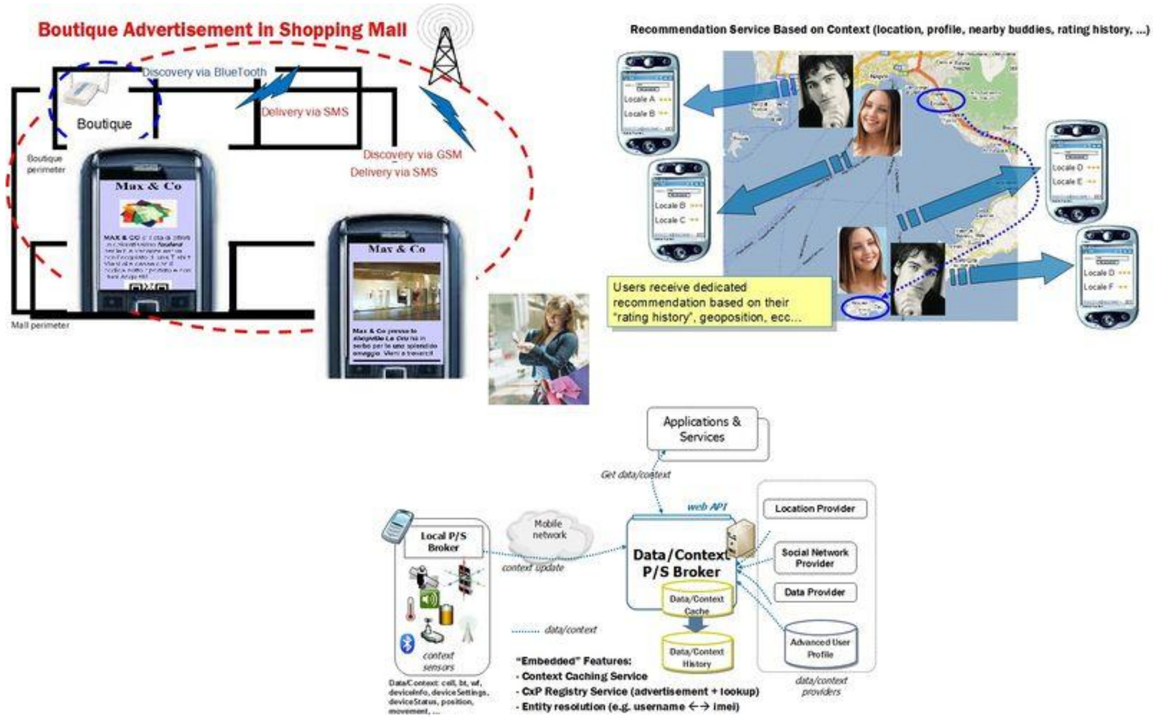


Figure: Context-aware Applications and Service enabled by the Publish/Subscribe GE

10.4 Basic Concepts

10.4.1 Context Elements

Aligned with the standard OMA NGSI specification [4], **Context Information** in FI-WARE is represented through generic data structures referred as **Context Elements**. A Context Element refers to information that is produced, generated, collected or observed that may be relevant for processing, carrying out further analysis and knowledge extraction. It has associated a **value** defined as consisting of a sequence of one or more <name, type, value> triplets referred as **data element attributes**. FI-WARE will support a set of built-in **basic data types** as well as the possibility to define structured data types similarly to how they are defined in most programming languages.

Context data in OMA NGSI is represented through data structures called context elements which have associated:

- An EntityId and EntityType, uniquely identifying the entity to which context data refers.
- A sequence of one or more data element attributes (<name, type, value> triplets)
- Optional meta-data linked to attributes (also <name, type, value> triplets)

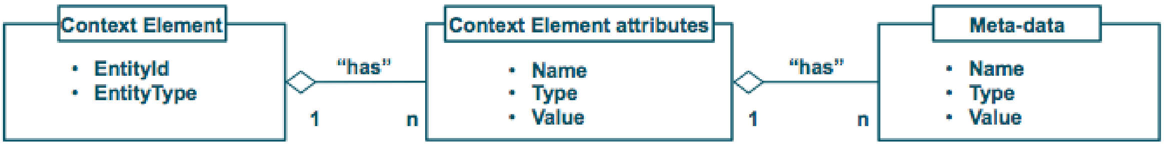
A Context Element typically provides information about attributes associated to a particular entity, being a physical thing or some concept relevant to an application. As an example, a context element may contain values of:

- the “last measured temperature”, “square meters” and “wall color” attributes associated to a “room” in a building

- attributes “speed”, “geolocation”, “current established route” of a “car”
- attributes “message geolocation”, “message contents” of a “user message”
- attributes “creation time”, “priority” of an “alarm”
- attributes “last launch time”, “last shutdown” of a “process”

Context Elements are generated as a result of some event in the system (e.g., the value assigned to an attribute of an entity has changed) or as a result to a query on attributes of well-known entities. Context Elements typically contain an **EntityId** and a **EntityType** uniquely identifying the entity about which context information is provided. Finally, there may be **meta-data** (also referred as semantic data) linked to attributes in a context element. However, existence of meta-data linked to a context element attribute is optional.

The EntityId is a string, and can be used to designate “anything”, not necessarely “things” in the “real world” but also application entities.



A cornerstone concept in FI-WARE is that context elements are not bound to any specific representation formalism. As an example, they can be represented as:

- an XML document (SensorML, ContextML, or whatever)
- a binary buffer being transferred
- as an entry in RDBMS table (or a number of entries in different tables),
- as a number of entries in a RDF Repository
- as entries in a noSQL data base like MongoDB.

A key advantage of this conceptual model for context elements is its compliance with IoT formats (e.g., SensorML) while enabling further extensions to them.

10.4.2 Basic Entities of the GE Model

10.4.2.1 *Context Broker*

As already mentioned the *Context Broker* (**CB**) is the main component of the architecture. It works as a handler and aggregator of context data and as an interface between architecture components. Primarily the CB has to control context flow among all attached components; in order to do that, the CB has to be able to respond to every Context Source or Context Consumer, supporting a high-performance and large-scale management of context information. This requires supporting highly efficient Context Cache and Context History management mechanisms. Besides this, the CB has to know every Context Provider in the architecture (this last feature is supported through an announcement process detailed in next sections).

10.4.2.2 *Context Provider*

A *Context Provider* (**CP**) is a kind of Context Producer that provides context information in synchronous mode; that means that a Context Consumer or even the Context Broker can invoke the CP in order to acquire context information. A CP provides context information (as a collection of context elements) only further to a specific invocation; it never sends data to another platform component in asynchronous mode. Moreover, a CP can produce new context information inferred from the computation of input parameters; hence it is responsible for reasoning of high-level context and for sensor data fusion. Every CP registers its availability and capabilities by sending appropriate announcements to the CB and exposes interfaces to provide context information to the CB and to Context Consumers.

10.4.2.3 *Context Source*

A *Context Source* (**CS**) is a kind of Context Producer that updates context information, about one or more context scopes, in asynchronous mode. A CS sends context information (as a collection of context elements) according to its internal logic and never due to an invocation from another middleware component and does not expose the same interfaces of CP to the CB and to Context Consumers.

Compared to the pull based CP-CB communication, the communication between CS and CB is in push mode.

10.4.2.4 *Context Consumer*

A *Context Consumer* (**CC**) is an entity (e.g. a context based application) that uses context information. A CC can retrieve context information sending a request to the CB or invoking directly a CP over a specific interface. Another way for the CC to obtain information is using implicit request to the CB, it means that the CC subscribes to a specific context update event and the CB notifies it when events occur.

10.4.2.5 *Entity*

In this Context Management Framework every exchange of context information is referred to a specific entity, which can be in its order a complex group of more than one entity. An entity is the subject (e.g. user or group of users), which context data refer to, and it is represented by a type and an identifier. Every Context Provider supports one or more entity type and this information are published to the Context Broker during an announcement process described later.

A type is an object that denotes a set of entities; for example entity types are:

- Human users
- Mobile devices
- Mobile users
- A group of entities within an application

An entity identifier is a value specifies a particular entity within the set of entities belonging to the same type; for example entity identifiers can be values of the following types:

- *username* – for human users;
- *imei* – for mobile devices;

- *mobile phone number* – GSM phone number for mobile users;
- *SIP uri* – for SIP accounts;
- *groupid* – for groups of other entities;

Every human user of the context management platform could be related to many entities in addition to the obvious type whose members are identified by an username. This means that a process that provides identity resolution is necessary. Considering for example a CP that provides geographical cell based location for mobile devices; if the location information is obtained from the computation of parameters provided by mobile devices, this CP supports the entity type associated to mobile users. When the CB receives a request about location of a human user, therefore using a username as identifier, the CB could not invoke the provider previously described because it does not support this entity type, but if the user has a mobile device, information about his location is probably available in the system. If the CB could retrieve all entities related to this user, it could invoke the provider using, if it is possible, right entities. This feature could be provided using a detailed database collecting all information about users; it means that the CB could refer to this DB in order to retrieve all entities associated to a specific user. In this way the example described previously could work because, when the CB receives the request, it invokes the database and retrieves the entity of type mobile user, identified with a mobile phone number, that represents the user; afterwards, the CB could invoke the location provider using the obtained entity type and identifier and could send a response with location data to the requester.

10.4.2.6 *Context scopes*

Every context information set within the Context Management Framework is defined as a "scope", which is a set of closely related context attribute. Every context attribute has a name and belongs to only one scope. Using scope as context exchange unit is very useful because attributes in that scope are always requested, updated, provided and stored at the same time; it means that creation and update of data bound to attributes within a scope are always atomic and that context information in a scope are always consistent. Scopes themselves can be atomic or aggregated, as union of different atomic context scopes.

For example take into account the scope position referring to the geographic position in which an entity is. This scope could be composed of attributes latitude, longitude and accuracy (intended as error on location) and these are always handled at the same time. Updating for example the latitude value without updating longitude, if it has also changed, and vice versa is obviously not correct.

10.4.3 Features and Functionalities

10.4.3.1 *Context caching*

Every context information (scope) received by the Context Broker (from a Context Source or as a result of a request to a Context Provider) is stored in a context cache. If another Context Consumer requests the same scope to the Context Broker, it can be retrieved from the cache, if it is not expired (see next Chapter 3.4), without need to invoke the same Context Provider again and therefore speeding up the process of context delivery.

10.4.3.2 *Context validity*

Every scope that is exchanged is tagged with its timestamp and expiry time. The expiry time tag states the validity of the scope. After this time, the information is considered not to be valid any more, and should be deleted. The setting of the expiration time is in charge of the Context Source or Context Provider that generates the context information and the Context Broker can only change it to synchronize to its clock.

When the Context Broker is asked for a scope, it first looks for it in its cache. If the information is found, the expiry time is checked. If the expiration time is reached, the Context Broker removes it from the context cache and requests it from respective Context Provider again.

10.4.3.3 *Context history*

Every context scope exchanged between the Context Broker and Context Providers or Context Sources is logged in the context history. Differently is for the context cache, which stores only currently valid information. The context history makes the past context information of an entity available, without reference to current validity. Context reasoning techniques could be applied to the context history in order to correlate contexts and deduce further context information, e.g. about situations, user intentions (sub-goals) and goals.

10.4.4 Fi-WARE NGSI Specification

Most of this GE's API operations regarding context information retrieval and notification are inspired on the OMA (Open Mobile Alliance) NGSI Context Management specifications.

However, the Fi-WARE team has identified potential updates to the standard to guarantee its correct exploitation in this context, solve some ambiguities and extend its capabilities according to the FI-WARE vision. Therefore, we will speak onwards about the FI-WARE NGSI specification, which is still under discussion and, hence some contents in the APIs description included in the present document will vary to be aligned with the final FI-WARE NGSI that will be specified.

10.5 Main Interactions using the FI-WARE NGSI Restful API

10.5.1 OMA NGSI Basics

OMA NGSI (Next Generation Service Interface) Operations are grouped into two major interfaces:

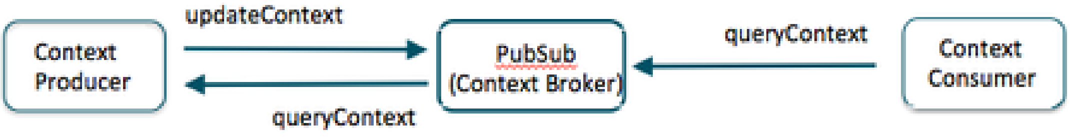
- NGSI-10
    - updateContext
    - queryContext
    - subscribeContext / unsubscribeContext / updateContextSubscription
    - notifyContext
- NGSI-9
    - registerContext
    - discoverContextAvailability

- subscribeContextAvailability / unsubscribeContextAvailability / updateContextAvailabilitySubscription
- notifyContextAvailability

The FI-WARE Publish/Subscribe Context Broker GE is an evolution/modification proposal of the OMA NGSI-10 which aims to maximize the role of NGSI for massive data collection, where a myriad of entities (e.g., IoT resources from the IoT world) are providing context elements occurrence/updates involving low level protocols.

10.5.2 Basic Interactions and related Entities

The following diagram depicts the basic interactions of the Publish/Subscribe Context Broker GE (from now on referred simply as "Context Broker") with its natural counterparts, that is the Context Producers (either Context Sources or Context Providers) and the Context Consumers.



The basic interactions are:

- Context Producers playing the role of Context Sources publish data/context elements by invoking the updateContext operation on a Context Broker.
- Some Context Producers may also play the role of Context Providers and export a queryContext operation Context Brokers may invoke at any given time to query on values of a designated set of attributes linked to a given set of entities
- Context Consumers can retrieve data/context elements by invoking the queryContext operation on a Context Broker
- Context data is kept persistent by Context Brokers and ready to be queried while not exceeding a given expiration time. This is a distinguishing feature of the OMA Context Management model as compared to some Event Brokering standards.

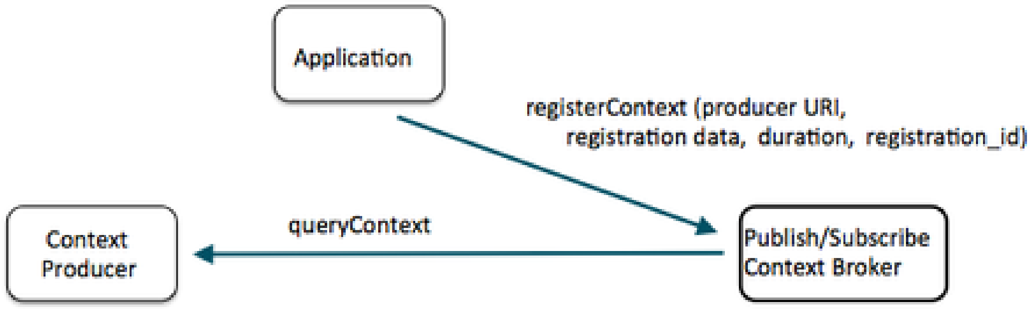
10.5.3 Registration of query-able Context Producers (Context Providers)

Some Context Producers, referred as Context Providers, export a queryContext operation Context Brokers may invoke at any given time to query on values of a designated set of attributes linked to a given set of entities. Context Brokers need to know which Context Providers exist and what are the entities they can provide information about. Third applications provide this information by means of invoking the registerContext operation that Context Brokers export.

The following diagram depicts the interactions supported by the Context Broker GE for this purpose

- Third applications register information about a given Context Provider by means of invoking the registerContext operation that the Context Broker exports. Information about what entities the Context Provider can inform about is provided in the request.

- Context Brokers can retrieve data/context elements by invoking the queryContext operation on a Context Provider

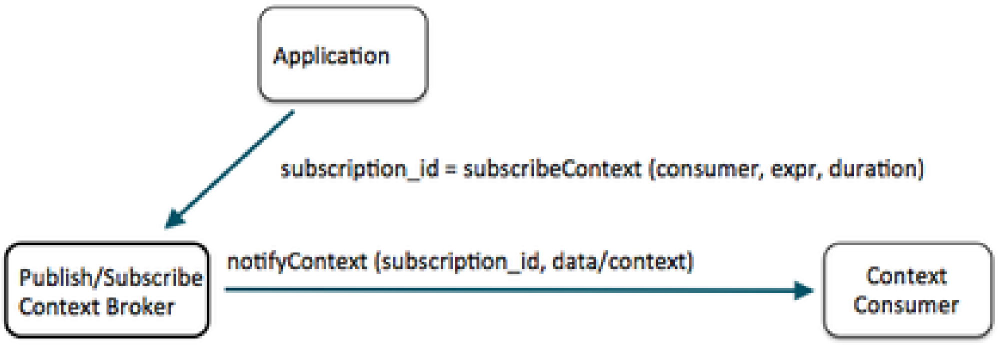


10.5.4 Interactions to subscribe Context Consumers to specific notifications

Some Context Consumers can be subscribed to reception of data/context elements which comply with certain conditions, using the subscribeContext operation a ContextBroker exports. Such subscriptions may have a duration.

Subscribed consumers spontaneously receive data/context elements compliant with that subscription through the notifyContext operation they export.

Note that the Application which subscribes a particular Context Consumer may or may not be the/a Context Consumer itself.



10.5.5 Extended Operations: Registering Entities & Attributes availability

The registerContext operation in Context Brokers can be used not only for registering ContextProducers on which queryContext operations can be invoked but also to register existence of entities in the system and availability of attributes.

Besides, Context Brokers may export operations to discover entities or even attributes and attribute domains that have been registered in the system.

10.5.6 Extended Operations: Applications subscription to Entities/Attributes registration

Some applications can be subscribed to registration of entities or availability of attributes and attribute domains which comply with certain conditions, using the subscribeContext Availability operation a ContextBroker may export. Such subscriptions may have a duration.

Subscribed applications spontaneously receive updates on new entities, attributes or attribute domains compliant with that subscription through the notifyContextAvailability operation they export

Note that the subscriber and subscribed applications may not be the same

10.6 Main interactions using ContextML/CQL

10.6.1.1 *ContextML Basics*

In order to allow a heterogeneous distribution of information, the raw context data needs to be enclosed in a common format understood by the CB and all other architectural components. Every component in the Context Management Framework that can provide context information has to expose common interfaces for the invocations. A light and efficient solution could be REST-like interfaces over HTTP protocol, allowing components to access any functionality (parameters or methods) simply invoking a specific URL. It should be compliant with the following pattern:

'http://<SERVER>/<MODULE>/<INTERFACE>/<OPERATION>/<RESOURCE>?[<OTHER_PARAMETERS>]'

The returned data are formatted according to the Context Management Language (ContextML) proposed for this architecture. '**ContextML'** [5] is an XML-based language designed for use in the aforementioned context awareness architecture as a common language for exchanging context data between architecture components. It defines a language for context representation and communication between components that should be supported by all components in the architecture. The language has commands to enable CPs to register themselves with the Context Broker and enables potential Context Consumers to discover the context information they need. Context information could refer to different context scopes.

ContextML allows the following features:

- Representation of context data
- Announcement of Context Providers toward Context Broker
- Description of Context Providers published to the Context Broker
- Description of context scopes available on Context Broker
- Representation of generic response (ACK/NACK)

The ContextML schema is composed by:

- '**ctxEls'**: contains one or more context elements
- '**txAdvs'**: contains the announcement of Context Provider features toward the Context Broker
- '**scopeEls'**: contains information about scopes actually available to the Context Broker
- '**ctxPrvEls'**: contains information about Context Providers actually published to the Context Broker
- '**ctxResp'**: contains a generic response from a component

10.6.1.2 *Context Data*

Any context information given by a provider refers to an entity and a specific scope. When a context provider is queried, it replies with a ContextML document which contains the following elements:

- **ContextProvider**: a unique identifier for the provider of the data
- **Entity**: the identifier and the type of the entity which the data are related to;
- **Scope**: the scope which the context data belongs to;
- **Timestamp** and **expires**: respectively, the time in which the response was created, and the expiration time of the data part;
- **DataPart**: part of the document which contains actual context data which are represented by a list of a features and relative values through the *<par>* element (“parameter”). They can be grouped through the *<parS>* element (“parameter struct”) and/or *<parA>* element (“parameter array”) if necessary.

The below Figure 11 context data provided from a CP that supports the *civilAddress* scope.


```
<contextML>
  <ctxEls>
    <ctxEl>
      <contextProvider id="LP" v="1.1.0" />
      <entity type="username" id="userId" />
      <scope>civilAddress</scope>
      <timestamp>2011-02-27T12:20:11+01:00</timestamp>
      <expires>2011-02-27T13:20:11+01:00</expires>
      <dataPart>
        <parS n="civilAddress">
          <par n="room">1037</par>
          <par n="corridor">North</par>
          <par n="floor">2</par>
          <par n="building">B</par>
          <par n="street">Via G. Reiss Romoli 274</par>
          <par n="postalCode">10148</par>
          <par n="city">Torino</par>
          <par n="subdivision">TO</par>
          <par n="country">Italy</par>
        </parS>
      </dataPart>
    </ctxEl>
  </ctxEls>
</contextML>
```


Figure 12: *civilAddres* Scope Example

10.6.1.3 *ContextML Naming Conventions*

The following naming conventions are applied to *scope names*, *entity types*, and to ContextML *parameters* (<*par*>), *arrays* (<*parA*>) and *parameters structs* (<*parS*>)

- names should be **lower case**, with the capital letters if composed by more than one word
    - example: *cell*, *longitude*, *netType*
- special chars like *_:/ must be avoided
- MAC addresses or Bluetooth IDs should be written without ':' separator, using capital letters
    - example: MAC address *00:22:6B:86:85:E3* should be represented: *00226B8685E3*

10.6.2 ContextML API

A description of available methods and examples could be find in ContextML API.

10.6.3 ContextQL (CQL)

ContextQL or CQL [8] is an XML-based language allowing to subscribe to the Context Broker (and in future to Publish/Subscribe Broker) by scope conditions and rules consisting of more then one conditions. The applications may request or subscribe to the broker for the real-time context and for history data placing certain matching conditions and rules directly into a (subscription) requests. ContextQL is based on ContextML described above for the data representation and communication between the components within the Pub/Sub GE architecture (a response to a CQL query is a ContextML document). The ContextML objects within filters and conditions are elements of the the ContextQL matching or conditional rules.

10.6.3.1 *Context Query*

A context query allows to send to the Pub/Sub broker a complex request consisting of many rules with conditions and matching parameters over the data available to the broker in real-time (inclding the context cache) and in the history. A query may contain the following elements:

- *action* – an action to undertake as response to the query The type of the actions is determined by the response of the broker
- *entity* – a subject or an object (an entity) or set of entities to which the query refers to
- *scope* – scope to which a query refers to
- *timerange* – period of time interval) to which a query refers to. This parameter (expressed by ttwo attributes *from* and *to* that indicates the begin and the end of the range respectively) indicates if data to be considered within context cache or in the context history on in both
- *conds* – set of conditions that express a filter of the query

The following actions can be represented in CQL:

- **SELECT** – allow to request to broker the context information regarding certain entity and certain scope matching certain conditions. A wildchart e.g. *entityType*|* or *username*|* is allowed
- **SELECT** with the option **LIST** – allows to retrieve a list of all entities of a certain type that satisfying in their context to certain conditions
- **SELECT** with the option **COUNT** – allows to count all the entities which context satisfy certain conditions
- **SUBSCRIBE** – subscribes to the broker for a certain scope matching certain conditions. The requests such as *entityType*|* are permitted. The subscription is limited to certain time or period indicated in the subscription request and might be shortened by the broker down to refusal of the subscription. Therefore a subscription shall be periodically renewed. Any accepted subscription is associated by the broker to an unique *subId* that shall be used by the application submitting the subscription request. An unsubscribe request can be implemented by a subscription with a certain subId of an existing subscription setting the validity period to zero.

The following conditions can be expressed in CQL:

- **ONCLOCK** – conditions that shall be checked in a certain period of time returning a result. This is an asynchronous request therefor can be executed only in SUBSCRIBE requests
- **ONCHANGE** – conditions that will be respected when one of matching parameters will be changed . This is an asynchronous request therefor can be executed only in SUBSCRIBE requests
- **ONVALUE** – cconditions that shall match certain parameters to observe. This might be both a synchronous and an asynchronous requests therefore could be executed as both SELECT and SUBSCSRIBE actions

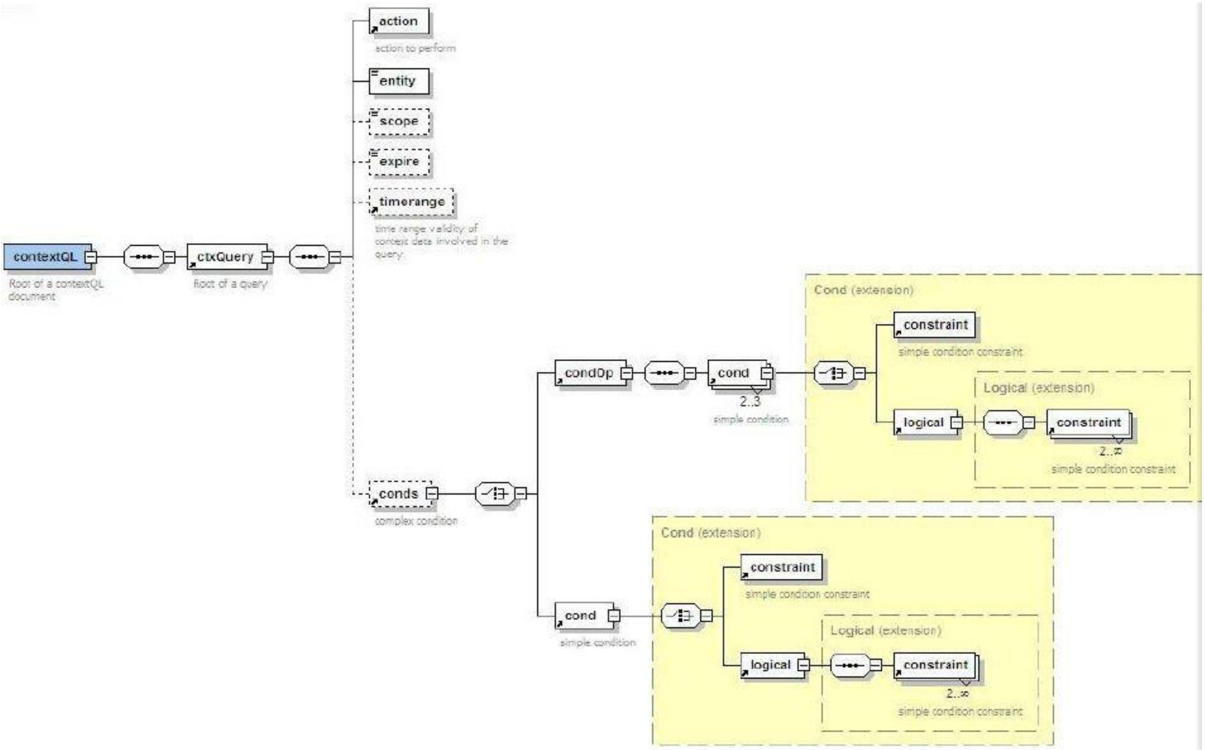


Figure 13: XSD schema of a ContextML query

The *conds* tag may contain one or more conditions for any condition type. If there are more then conditions elements they shall be linked *condOp*. The following table indicated the conditions combinations of different types that cab be handled by the broker.

	C*	H*	V*
AND	NO	YES	YES
OR	YES	YES	YES

***C**:ONCLOCK, **H**:ONCHANGE, **V**:ONVALUE

Table 1: Combinations of possible conditions in the broker

For example a subscription request to *position* scope for 5 minutes and every time the position is retrieved by GPS will be accepted.

A single condition may contain one or more *tag constraints*, in this case the conditions are linked by a logical operator *tag logical* and limited to one only depth level.

Every constraint element has at maximum 4 attributes and its evaluation depends on the applied conditions:

- *param* – iodentifies parameters to which refers a condition and its value shall be the same context type to match e.g. *scope.par*, *scope.parS.par*, *scope.parA[n].par*. This attribute does not exist if the condition ONCLOCK

- *op* – identifies operator to apply to a parameter. This attribute exist only in the conditions ONVALUE. Currently defined attributes are of arithmetic and string-based types, which are listed in the below Table 2

GT	Major of	NCONT	Not contain
NGT	Not major of	STW	Begin with
LT	Minor of	NSTW	Not begin with
NLT	Non minor of	ENW	End with
EQ	Equal to	NENW	Not end with
NEQ	Not equal to	EX	Exist
CONT	Contain	NEX	Not exist

Table 2: ContextQL operators

- *value* – identifies a value matched in the condition. This attribute exists only if condition is **ONVALUE** or **ONCLOCK** (in this case indicates the number of seconds when the condition will be verified). In case of **ONVALUE** condition this attribute doesn't exist for some operations such as e.g. **EX** and **NEX**
- *delta* – used only in conditions **ONVALUE** and if matching parameter have value within certain interval. Identifies a tolerance threshold in condition matching e.g. *param=position.latitude*, *op=EQ*, *value=45*, *delta=0.2*, where the constraint matching for latitude values included within 44.8 e 45.2.

10.6.4 CQL API

A description of Context Query Language with some examples could be find in CQL API

10.7 Basic Design Principles

10.7.1 Conceptual Decoupling

Context and data distribution is the process through which information is distributed and shared between multiple data and context producing and consuming entities in a context(data)-aware system. For efficient data/context management, including context/data distribution, it is imperative to consider communication schemes with respect to the decoupling they provide. Various forms of decoupling are:

- Space Decoupling: The interacting parties do not need to know each other. The publishers (providers) publish information through an event/information service and the subscribers (consumers) receive information indirectly through that service. The publishers and subscribers do not usually hold references to each other and neither do they know how many subscribers/publishers are participating in the interaction.
- Time Decoupling: The interacting parties do not need to be actively participating in the interaction at the same time i.e., the publisher might publish some information while the subscriber is disconnected and the subscriber might get notified about the availability of some information while the original publisher is disconnected.

- Synchronization Decoupling: Publishers are not blocked while producing information, and subscribers can get asynchronously notified (through call-backs) of the availability of information while performing some concurrent activity i.e. the publishing and consumption of information does not happen in the main flow of control of the interacting parties.

This decoupling is important to cater for because decoupling of production and consumption of information increases scalability by removing all explicit dependencies between the interacting participants. Removing these dependencies strongly reduces coordination requirements between the different entities and makes the resulting communication infrastructure well adapted to distributed environments. This advantage becomes more beneficial when mobile entities exist in a distributed system (owing to their limited resources, intermittent connectivity etc).

10.7.2 References

[1]	Weiser, M., "The computer for the 21st century", Human-computer interaction: toward the year 2000, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.
[2]	Lamorte L., Licciardi C. A., Marengo M., Salmeri A., Mohr P., Raffa G., Roffia L., Pettinari M. & Salmon Cinotti T., 2007, “A platform for enabling context aware telecommunication services”, 3rd Workshop on Context Awareness for Proactive Systems, University of Surrey, UK, June 2007.
[3]	Moltchanov B., Knappmeyer M., Licciardi C. A., Baker N., 2008, “Context-Aware Content Shariing and Castiing”, ICIN2008, Bordeaux, France, UK, October 2008.
[4]	Open Mobile Alliance (OMA) Next Generation Services Interface (NGSI) Specification http://www.openmobilealliance.org/Technical/release_program/docs/NGSI/V1_0-20101207-C/OMA-TS-NGSI_Context_Management-V1_0-20100803-C.pdf
[5]	Moltchanov B., Knappmeyer M., Liaquat Kiani S., Fra' C., Baker N., 2010, “ContextML: A Light-Weight Context Representation and Context Management Schema”, IEEE International Symposium on Wireless Pervasive Computing, Modena, Italy, May 2010.
[6]	Sumi, Y., Etani, T., Fels, S., Simonet, N., Kobayashi, K. & Mase, K., 1998, “C-map: Building a context-aware mobile assistant for exhibition tours”, Community Computing and Support Systems, Social Interaction in Networked Communities, Springer-Verlag, UK, pp.137–154.
[7]	Cheverst, K., Davies, N., Mitchell, K., Friday, A. & Efs-tratiou, C., 2000, “Developing a context-aware electronic tourist guide: some issues and experiences”, Proceedings of the SIGCHI conference on Human Factors in Computing Systems, ACM Press, New York, USA, pp.17–24.
[8]	Moltchanov B., Fra' C., Valla, M., Licciardi C. A., 2011, “Context Management Framework and Context Representation for MNO”, Activity Context Workshop / AAAI2012, San Francisco, USA, August 2012.

[9]	Gu, T., Pung, H.K. & Zhang, D.Q., 2004, “A middleware for building context-aware mobile services”, Proceedings of IEEE Vehicular Technology Conference (VTC), Milan, Italy
[10]	Fahy, P. & Clarke, S., 2004, “CASS – a middleware for mobile context-aware applications”, Workshop on Context Awareness, MobiSys 2004.
[11]	MobiLife, an Integrated Project in European Union’s IST 6th Framework Programme, http://www.ist-mobilife.org
[12]	Service Platform for Innovative Communication Environment (SPICE), An Integrated Project in European Union’s IST 6th Framework Programme, http://www.ist-spice.org
[13]	Open Platform for User-centric service Creation and Execution (OPUCE), An Integrated Project in European Union’s IST 6th Framework Programme.
[14]	Context-aware Content Casting (C-CAST), A Research Project in European Union's ICT 7th Framework Programme.

11 FIWARE ArchitectureDescription Data CEP

You can find the content of this chapter as well in the wiki of fi-ware.

11.1 Copyright

11.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

11.3 Overview

11.3.1 Introduction to the CEP GE

The IBM Proactive Technology On Line – Proton - is a scalable integrated platform to support the development, deployment, and maintenance of Complex Event Processing (CEP) applications.

CEP analyses event data in real-time, generates immediate insight and enables instant response to changing conditions. Some functional requirements this technology addresses include event-based routing, observation, monitoring and event correlation. The technology and implementations of CEP provide means to expressively and flexibly define and maintain the event processing logic of the application, and in runtime it is designed to meet all the functional and nonfunctional requirements without taking a toll on the application performance, removing one issue from the application developer’s and system managers concerns.

Entities connected to CEP (application entities or some other GEs like the Publish/Subscribe Broker GE) can play two different roles: the role of Event Producer or the role of Event Consumers. Note that nothing precludes that a given entity plays both roles. Event Producers are the source of events for event processing. Following are some examples of event producers:

- External applications reporting events on user activities such as "user placed of new order", and on operation activities such as "delivery has been shipped".
- Sensors reporting on a new measurement. Such a sensor generated event can be consumed directly by the CEP GE. Another alternative is that the sensor event is gathered and processed through the IoT GEs, which publish context events to the Publish/Subscribe GE, having the CEP be a context consumer of the Publish/Subscribe GE.

They can provide events in two modes:

- "Push" mode: The Event Producers push events into CEP by means of invoking a standard operation CEP exports.
- ”Pull” mode: The Event Producer exports a standard operation that CEP can invoke to retrieve events.

Event Consumers are the sink point of events. Following are some examples of event consumers:

- Dashboard: a type of event consumer that displays alarms defined when certain conditions hold on events related to some user community or produced by a number of devices.
- Handling process: a type of event consumer that consumes meaningful events (such as opportunities or threats) and performs a concrete action.
- The Publish/Subscribe Broker GE: a type of event consumer that forwards the events it consumes to all interested applications based on a subscription model.

CEP implements event processing functions based on the design and execution of Event Processing Networks (EPN). Processing nodes that make up this network are called Event Processing Agents (EPAs) as described in the book “Event Processing in Action” [EPIA]. The network describes the flow of events originating at event producers and flowing through various event processing agents to eventually reach event consumers, see the figure below for an illustration. Here we see that events from Producer 1 are processed by Agent 1. Events derived by Agent 1 are of interest to Consumer 1 but are also processed by Agent 3 together with events derived by Agent 2. Note that the intermediary processing between producers and consumers in every installation is made up of several functions and often the same function is applied to different events for different purposes at different stages of the processing. The EPN approach allows to deal with this in an efficient manner, because a given agent may receive events from different sources. At runtime, this approach also allows for a flexible allocation of agents in physical computing nodes as the entire event processing application can be executed as a single runtime artifact, such as Agent 1 and Agent 2 in Node 1 in the figure below, or as multiple runtime artifacts according to the individual agents that make up the network, such as Agent 1 and Agent 3 running within different nodes. Thus scale, performance and optimization requirements may be addressed by design.

The reasons for running pieces of the network in different nodes or environments vary, for example:

- Distributing the processing power
- Distributing for geographical reasons – process as close to the source as possible for lower networking
- Optimized and specialized processors that deal with specific event processing logic

Another benefit in representing event processing applications as networks is that entire networks can be nested as agents in other networks allowing for reuse and composition of existing event processing applications.

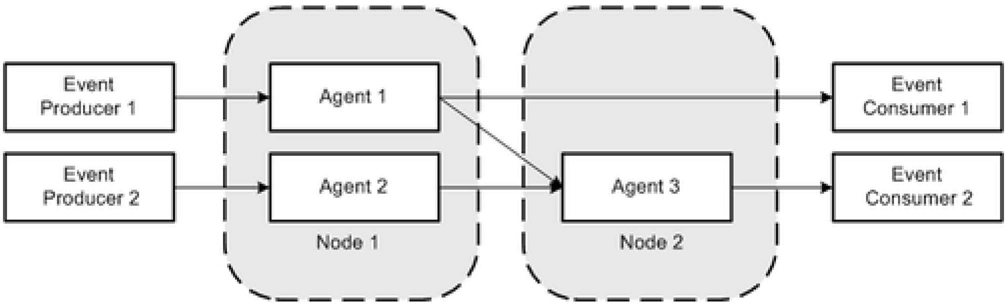


Illustration of an Event Processing Network made of producers, agents and consumers

The event processing agents and their assembly into a network is where most of the functions of CEP are implemented. The behavior of an event processing agent is specified using a rule-oriented language that is inspired by the ECA (Event-Condition-Action) concept and may better be described as Pattern-Condition-Action. Rules in this language will consist in three parts:

- A pattern detection that makes a rule of relevance
- A set of conditions (logical tests) formulated on events as well as external data
- A set of actions to be carried out when all the established conditions are satisfied

Following is an indication of the capabilities to be support in each part of the rule language.

11.3.1.1 *Pattern Detection*

In the pattern detection part the application developer may program patterns over selected events within an event processing context (such as a time window or segmentation) and only if the pattern is matched the rule is of relevance and according to conditions, the action part is executed. Examples for such patterns are:

- Sequence, meaning events need to occur in a specified order for the pattern to be matched. E.g., follow customer transactions, and detect if the same customer bough and later sold the same stock within the time window.
- Aggregate, compute some aggregation functions on a set of incoming events. E.g., compute the percentage of the sensors events that arrived with a fail status out of all the sensors events arrived in the time window. Alert if the percentage of the failed sensors is higher than 10 percent.
- Absent, meaning no event holding some condition arrived within the time window for the pattern to match. E.g., alert if within the time window no sensor events arriving from specific source have arrived. This may indicate that the source is down.
- All, meaning that all the events specified should arrive for the pattern to match. E.g., wait to get status events from all the 4 locations, where each status event arrives with the quantity of reservations. Alert if the total reservations are higher than some threshold.

Event Processing Context [EPIA] is defined as a named specification of conditions that groups event instances so that they can be processed in a related way. It assigns each event instance to one or more context partitions. A context may have one or more context dimensions and can give rise to one or more context partitions. Context dimension tells us whether the context is for a temporal, spatial, state-oriented, or segmentation-oriented context, or whether it is a composite context that is to say one made up of other context specifications. Context partition is a set into which event instances have been classified.

11.3.1.2 *Conditions*

The application developer may add the following kind of conditions in a given rule:

- Simple conditions, which are established as predicates defined over single events of a certain type

- Complex conditions, which are established as logical operations on predicates defined over a set of events of a certain type

11.3.1.3 *Actions*

The application developer of CEP may specify what should be done when a rule is detected. This can include generation of derived events to be sent to the producers and actions to be performed by the producers. These actions definitions include the parameters needed for their execution.

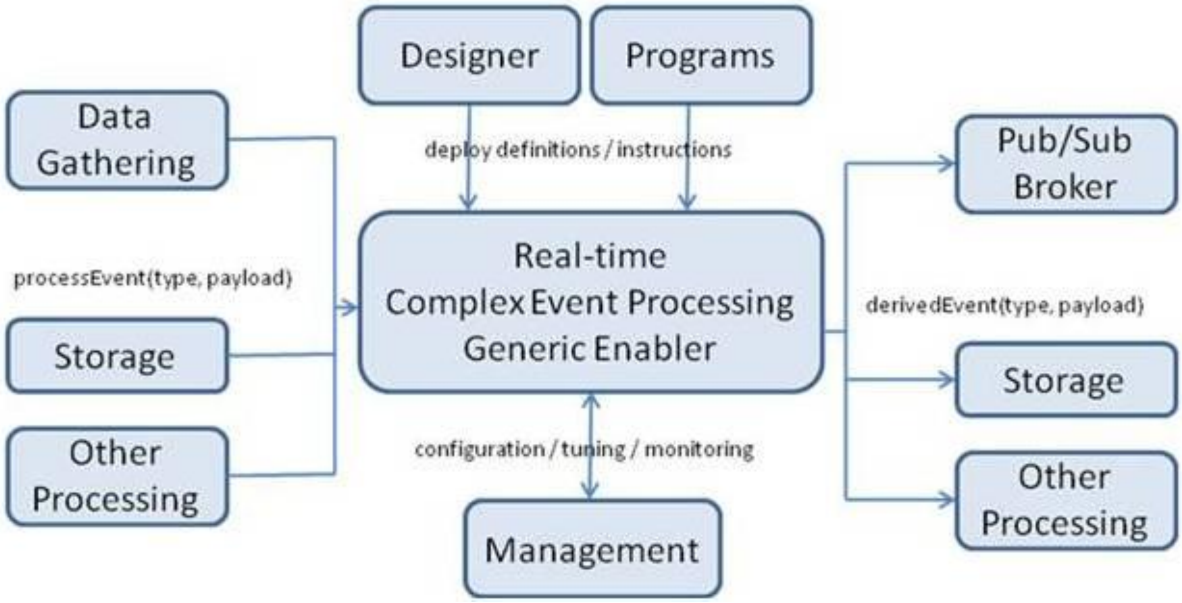
11.3.2 Target Usage

Complex Event Processing (CEP) is the analysis of event patterns in real-time to generate immediate insight and enable instant response to changing conditions. When the need is to respond to a specific event, the Pub/Sub context GE is sufficient. You should consider using the CEP GE when there is a need to detect pattern over the incoming events occurring within some processing context (see the pattern examples in the previous section). Some functional requirements this technology addresses include event-based routing, observation, monitoring and event correlation. The technology and implementations of CEP provide means to expressively and flexibly define and maintain the event processing logic of the application, and in runtime it is designed to meet all the functional and non-functional requirements without taking a toll on the application performance, removing one issue from the application developer’s and system managers concerns.

For the primary user of the real-time processing generic enabler, namely the consumer of the information generated, the Complex Event Processing GE (CEP GE) addresses the user’s concerns of receiving the relevant events at the relevant time with the relevant data in a consumable format. Relevant- meaning of relevance to the consumer\subscriber to react or make use of the event appropriately. The figure below depicts this role through a pseudo API *derivedEvent(type,payload)* by which, at the very least, an event object is received with the name of the event, derived out of the processing of other events, and its payload.

The designer of the event processing logic is responsible for creating event specifications and definitions (including where to receive them) from the data gathered by the Massive Data Gathering Generic Enabler. The designer should also be able to discover and understand existing event definitions. Therefore FI-WARE, in providing an implementation of a Real-time CEP GE, will also provide the tools for the designer. In addition, APIs will be provided to allow generation of event definitions and instructions for operations on these events programmatically, such as by an application or by other tools for other programming models that require Complex Event Processing such as the orchestration of several applications into a composed application using some event processing. In the figure below these roles are described as Designer and Programs making use of the pseudo API*deploy definitions/instructions*.

Finally, the CEP GE addresses the needs of an event system manager and operator, could be either real people or management components, by allowing for configurations (such as security adjustments), exposing processing performance, handling problems, and monitoring the system’s health, represented in the figure below as Management role making use of the pseudo API *configuration/tuning/monitoring*.



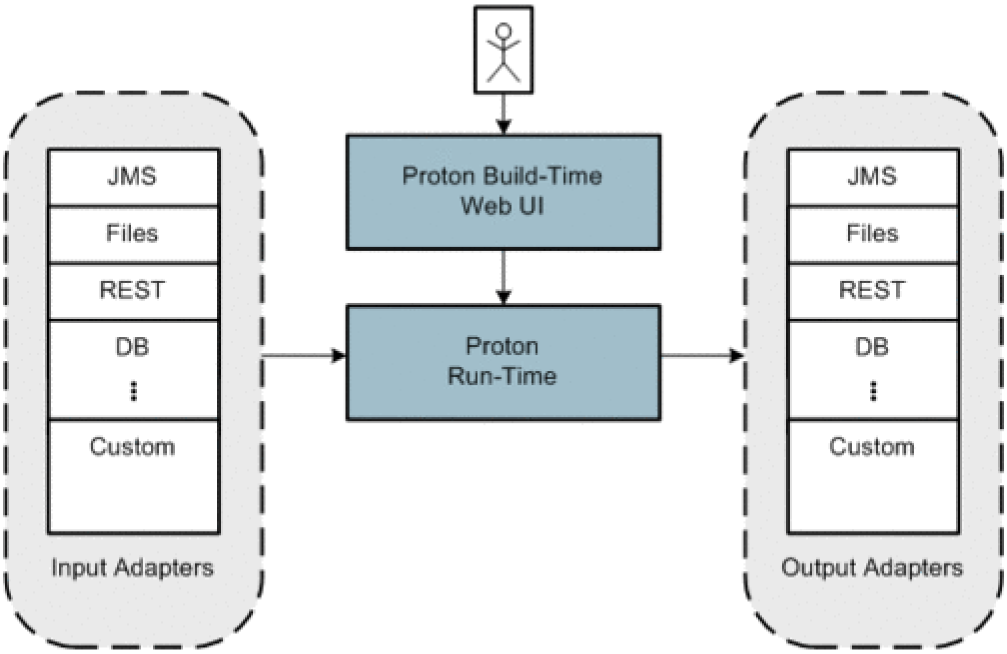
Interactions with and APIs of the Real-time CEP Generic Enabler

11.4 Basic Concepts

CEP has three main interfaces with its environment as can be seen in the figure below:

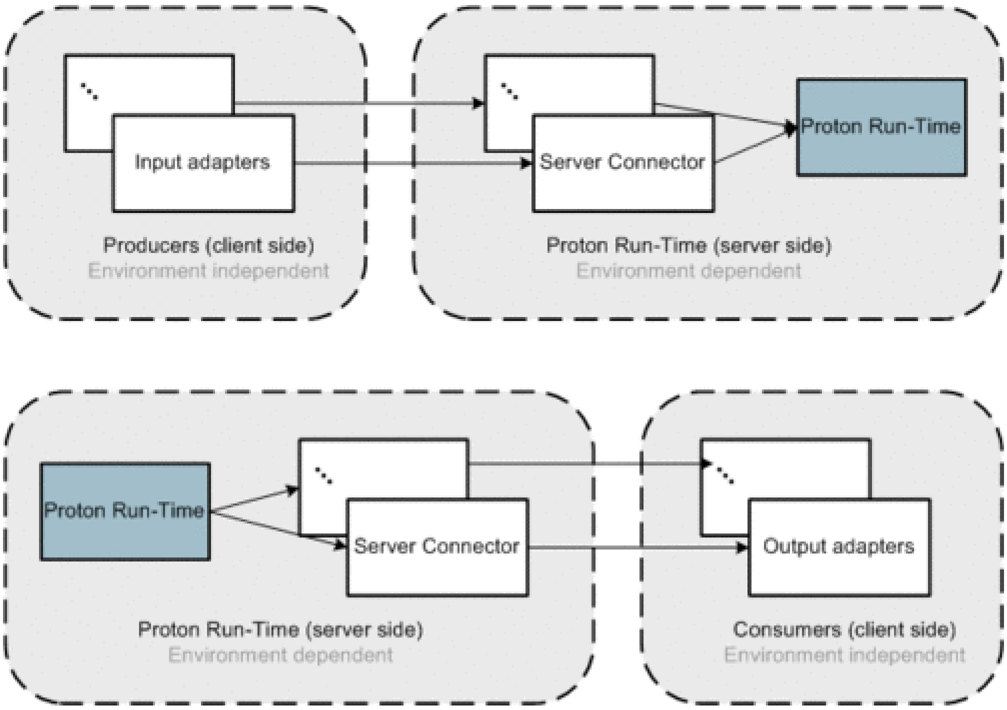
- Input Adapters for getting incoming events
- Output Adapters for sending derived events
- CEP Application definition

The application definitions, the EPN, is written by the application developer using CEP build-time web based user interface, by filling definition forms. The CEP build-time user interface generates a definition file which is sent to the CEP run-time. Alternatively this definition file, in JSON format, can be generated programmatically by any other application. At runtime, CEP receives incoming events through the input adapters. CEP processes those incoming events according to the application definitions and sends derived events through the output adapters.



CEP High Level Architecture

CEP semantic layer allows the user to define producers and consumers for event data (see the figure above). Producers produce event data, and consumers consume the event data. The definitions of producers and consumer, which is specified during the application buildtime are translated into input and output adapters in CEP execution time. The physical entities representing the logical entities of producers and consumers in CEP are adapter instances.



Adapters layer representation

As can be seen in the above figure, for each producer an input adapter is defined, which defines how to pull the data from the source resource, how to format the data into CEP's object format before delivering it to the engine. The adapter is environment-agnostic but uses the environment-specific connector object, injected into the adapter during its creation, to connect to CEP runtime.

The consumers and their respective output adapters are operated in a push mode – each time an event is published by the runtime it is pushed through environment-specific server connectors to the appropriate consumers, represented by their output adapters, which publish the event in the appropriate format to the designated resource.

The server connectors are environment-specific, they hide the implementation of the connectivity layer from the adapters which allows them to be environment-agnostic.

The J2SE implementation of CEP runtime includes an input and output socket servers, which allow the input and output server connectors to communicate with the runtime using sockets mechanism.

11.4.1 Adapters design principles

As part of the CEP application design the user specifies the events producers as sources of event data and the event consumers as sinks for event data. The specification of producer includes the resource from which the adapter pulls the information (whether this resource is a file in a file system, a JMS queue, REST service), and format settings which allow the adapter to transform the resource specific information to CEP event data object. The formatting depends on the kind of resource we are dealing with – for file it can be a tagged file formatter, for JMS an object transformer. Likewise, the specification of consumer includes the resource to which the event created by CEP runtime should be published and a formatter describing on how to transform a CEP event data object into resource-specific object.

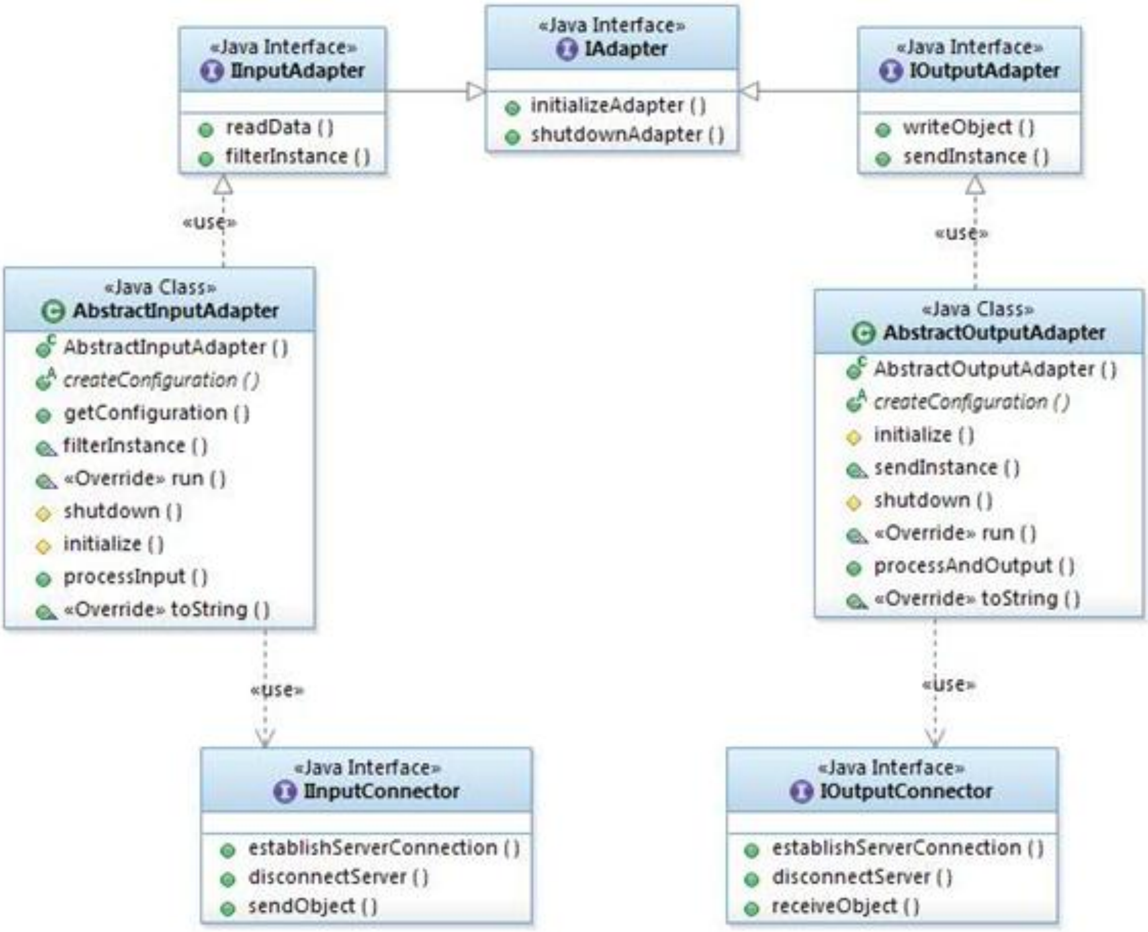
The design of adapter's layer satisfies the following principles:

- A producer is a logical entity which holds such specifications as the source of the event data, the format of the event data. The input adapter is the physical entity representing a producer, an entity which actually interacts with the resource and communicates event information to CEP runtime server.
- A consumer is a logical entity which hold such specifications as the sink for the vent data, the format of the sink event data. The output adapter is the physical representation of the consumer, it is an entity which is invoked by the CEP runtime when an event instance should be published to the resource.
- The input adapters all implement a standard interface, which is extendable for custom input adapter types and which allows to add new producers for custom-type resources.
- The output adapters all implement a standard interface, which is extendable for custom output adapter types and which allows to add new consumers for custom-type resources.
- A single event instance can have multiple consumers
- A producer can produce events of different types, a single event instance might serve as input to multiple logical agents within the event processing network, according to the network's specifications

- Producers operate in pull mode, each input adapter pulls the information from designated resource according to its specifications, each time processing the incremental additions in the resource. However, producers that operate in a push mode are planned to be supported as well.
- Consumers define a list of event types they are interested in, they can also specify a filter condition on each event type – only event instances satisfying this condition will be actually delivered to this consumer.
- Consumers operate in push mode, each time the CEP runtime publishes an event instance it is pushed to the relevant consumer.
- The producers and consumers are not directly connected, but the raw event's data supplied by a certain producer can be delivered to a consumer if the consumer specifies this event type in it's desirable events list.

11.4.2 Adapters design

A user defines a producer or consumer to act as event data supplier or consumer, he does so using CEP's authoring tool to create the appropriate JSON definition file that is the metadata of an application (the JSON definition file can also be written problematically by another application). The metadata defines the access information to the event data source or sink, and the information on how to format the data to/from CEP readable event object. In case the defined built-in adapter types (currently file, JMS, Rest client adapters) do not support the required communication channel of the application, the user can implement a custom adapter. A customer adapter is implemented above the given adapter framework which is built on the notion of extending a common abstract adapter (one for all input adapters and one for all output adapters). The adapter framework provides all the sequence of adapter's lifecycle management (initialization, establishing connection to the server, processing of data, and shutdown) ,all that each specific adapter implementation have to supply is resource-specific implementations of readData() or writeObject() methods in order to pull the data from this resource or push the data to the resource (see below the class diagram of the framework), as well as createConfiguration() adapter-specific method, which fetches the required adapter-specific properties from producer/consumer metadata and creates appropriate configuration object.



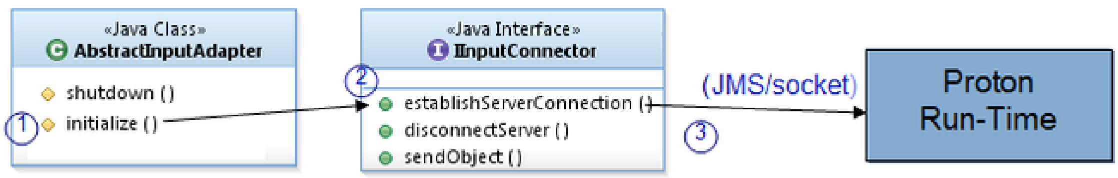
Adapters framework class diagram

11.5 Main Interactions

11.5.1 Definition of Input Adapters

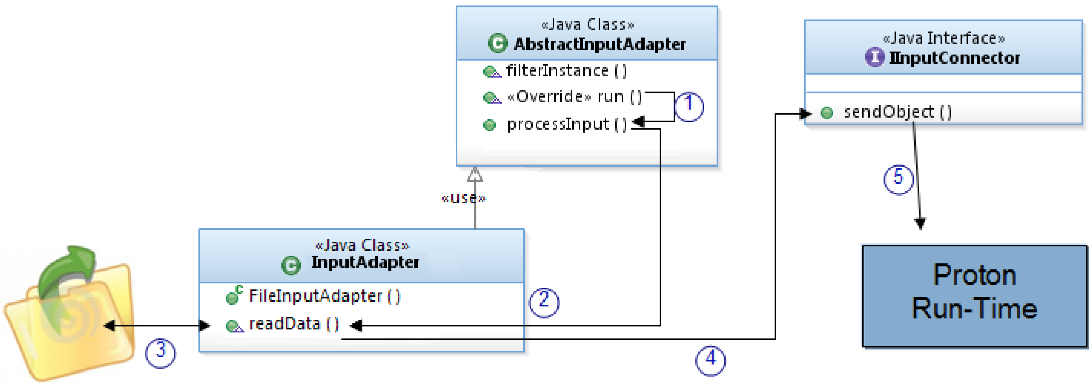
11.5.1.1 *Runtime*

The adapter representing the producer is configured, upon startup it is supplied with server connector which handles all communication of the adapter with CEP runtime (see an initialization sequence diagram below).



Adapter initialization sequence-establishing connection to CEP server

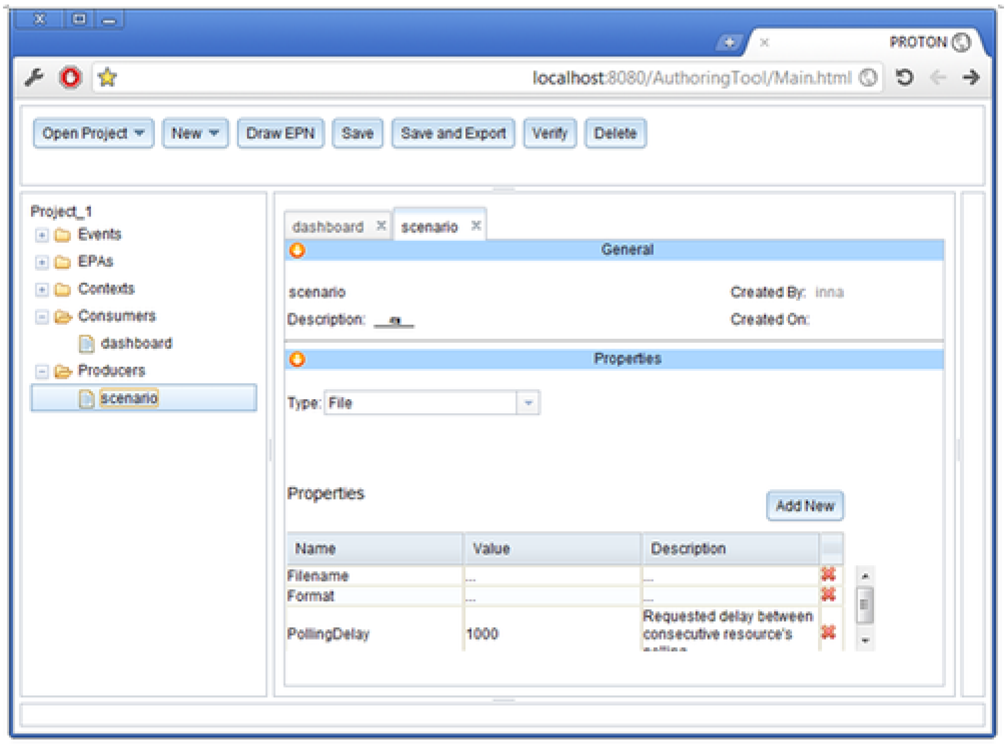
Once the adapter starts running (see below a diagram for processing sequence), it constantly polls the data source for changes (1,2), transforms an entry within the data source into CEP readable event object (3), and sends the information via server connector to the server (4), without being aware of the underlying communication infrastructure which the connector uses to establish the connection and send the data to the server.



Input adapter processing sequence

11.5.1.2 *Definition*

In order to define a producer we use the buildtime tool to choose the producer type and define suitable properties. The figure below depicts a producer definition screen in the CEP authoring tool.



Buildtime representation of a producer definition

In order to define a producer we need to supply the following information:

- The general metadata including the name of the producer, the description, the createdDate and createdBy information.
- The type of the producer – at the moment supporting FILE ,JMS and REST client adapters. DB adapter is currently not implemented, and there is an infrastructure to allow the user to add implementation of a custom adapter.
- Per specific chosen type – list of properties specifying the resource to access, the credentials to access the resource, and the formatter information
    - File adapter(see the figure below): the relevant properties include
        - The absolute path to the file representing the resource this file adapter is polling
        - Delay between sending event instances to the system
        - Polling interval - the interval for two consecutive polls of the resource for updates
        - Formatter type for the entries within the file : at the moment supporting tag-delimited formatter
        - Properties of the tag-delimited formatter, including the delimiter ,and the tag-data separator characters

```
{"name":"scenario","type":"file","properties":
[
    {"name":"filename","value":"C:\\scenario.txt"},
    {"name":"sendingDelay","value":"5000"},
    {"name":"pollingInterval","value":"1000"},
    {"name":"formatter","value":"tag"},
    {"name":"delimiter","value":";"},
    {"name":"tagDataSeparator","value":"="}
],
"desription":"","createdDate":"","createdBy": "inna"}
```

Producer of file type JSON definition

- JMS adapter (see the figure below): the relevant properties include
  - The hostname of the server where the input JMS destination resides
  - The port to connect to on the server where the input JMS destination resides
  - The JNDI name of the connection factory object
  - The JNDI name of the destination object
  - Sending delay between sending available event instances to the system
  - Polling interval – the interval for two consecutive polls of the resource for updates
  - Timeout – the polling timeout to wait for a new object on the JMS destination

```
{"name":"scenario2","type":"jms",
    "properties":
    [
        {"name":"hostname","value":"hostname.com"},
        {"name":"port","value":"2809"},
        {"name":"connectionFactory","value":"jms/inputConnectionFactory"},
        {"name":"destinationName","value":"jms/inputQueue"},
        {"name":"sendingDelay","value":"5000"},
        {"name":"pollingInterval","value":"1000"},
        {"name":"timeout","value":"3000"}

    ],
    "desription":"","createdDate":"","createdBy": "inna"}
```

Producer of JMS type JSON definition

If those are the only properties mentioned, the JMS producer assumes the JMS destination contains serializable objects which implement the IObjectMessage interface (see later in the description of interfaces) We can specify additional options for the formatter, in which case the JMS adapter implementation assumes the JMS message is a tag-delimited text message with the specified formatting information.

Additional properties for JMS producer which wishes to use formatted text messages are: (see the figure below)

- Formatter – the formatter type (right now only tag-delimited messages are supported so the only option is 'tag')
- Delimiter – the delimiter string between the tag-data groups
- TagDataSeparator – the separator within the tag-data pair


```
{"name":"scenario2","type":"jms",
    "properties":
    [
        {"name":"hostname","value":"hostname.com"},
        {"name":"port","value":"2809"},
        {"name":"connectionFactory","value":"jms/inputConnectionFactory"},
        {"name":"destinationName","value":"jms/protonQueue"},
        {"name":"sendingDelay","value":"5000"},
        {"name":"pollingInterval","value":"1000"},
        {"name":"timeout","value":"3000"},{"name":"formatter","value":"tag"},
        {"name":"delimiter","value":";"},
        {"name":"tagDataSeparator","value":"="}

    ],
    "desription":"","createdDate":"","createdBy": "inna"}
```


Producer of type JMS formatted text message JSON definition

- REST adapter (see the figure below) , is a REST client adapter which is capable to access the REST web service declared by the producer and pull events using the GET method. The relevant properties of this adapter include:
  - URL - the fully qualified URL of the web service for event pull operation. There is an assumption here that pull operation is done by calling GET method , which is viable assumption if the web service follows REST design patters.
  - contentType - can be "text/plain", "text/xml", "application/xml", "application/json" etc. This is basically defined by the web service and have to be entered here so the client knows how to access the web service.
  - sendingDelay - delay between pulling the events from the web service (in batch) and sending them to the CEP engine in runtime. This is convenient when we want to space the input events for some reason
  - pollingInterval - the time to wait between two consecutive approaches to the the web service to pull events.
  - pollingMode - whether web service returns a single instance or batch of event instances
  - Formatter properties - the same as in file. The user have to supply correct formatters to work with the defined content type, for example he will have to add formatters to deal with XML or JSON content. If the user defines formatter not suitable for contentType (for example he

defines tag formatter for a content which is not plain text) he will get an exception in the producer.

```
{"name":"rest","type":"rest","properties":
        [
        {"name":"URL","value":"http://localhost:9080/RESTWeb/rest/helloworld/producer"},
        {"name":"contentType","value":"text/plain"},
        {"name":"sendingDelay","value":"5000"},
        {"name":"pollingInterval","value":"1000"},
        {"name":"pollingMode","value":"BATCH/SINGLE"},
        {"name":"formatter","value":"tag"},
        {"name":"delimiter","value":";"},
        {"name":"tagDataSeparator","value":"="}
        ],
 "desription":"","createdDate":"","createdBy": "inna"},
```

Producer of REST type JSON definition

11.5.1.3 *Interfaces to implement for custom adapter*

Any newly added custom input adapter needs to implement a few interfaces:

1. It should extend the ***AbstractInputAdapter*** abstract class, which in turn implements the ***IInputAdapter*** interface , the developer should provide implementation for the following methods:

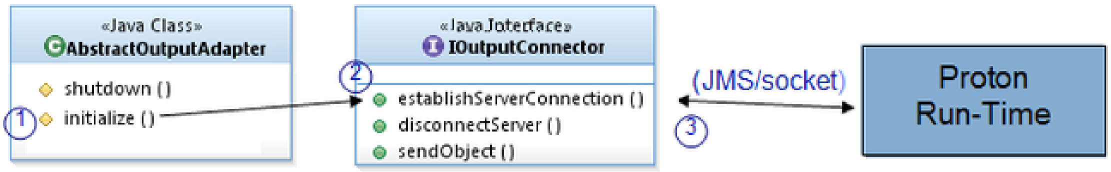
- ***IInputAdapterConfiguration createConfiguration(ProducerMetadata metadata)*** - creates the adapter-specific configuration object after extracting the relevant properties from the producer metadata object
- ***void initializeAdapter()*** - a method existing in the abstract interface but which should be overloaded with any resource-specific initialization after the call to the parent's method (such as acquiring a handle to a file, opening a connection to datasource etc.)
- ***IEventInstance readData()*** – which accesses the resource, pulls the event data and transforms each event data entry into CEP event instance object.
- ***void shutdownAdapter()*** – a method existing in the abstract interface but which should be overloaded with any resource-specific shutdown actions after the call to the parent's method (such as closing a handle to a file, closing a connection to datasource etc.)

2. It should provide an adapter specific implementation of ***IInputAdapterConfiguration*** interface for an adapter-specific configuration object. This is basically a bean with getters method carrying adapter specific information which helps the adapter to access the specific resource (such as filename for file adapter, or hostname and port name for JMS server for the JMS adapter)

11.5.2 Definition of Ouput Adapters

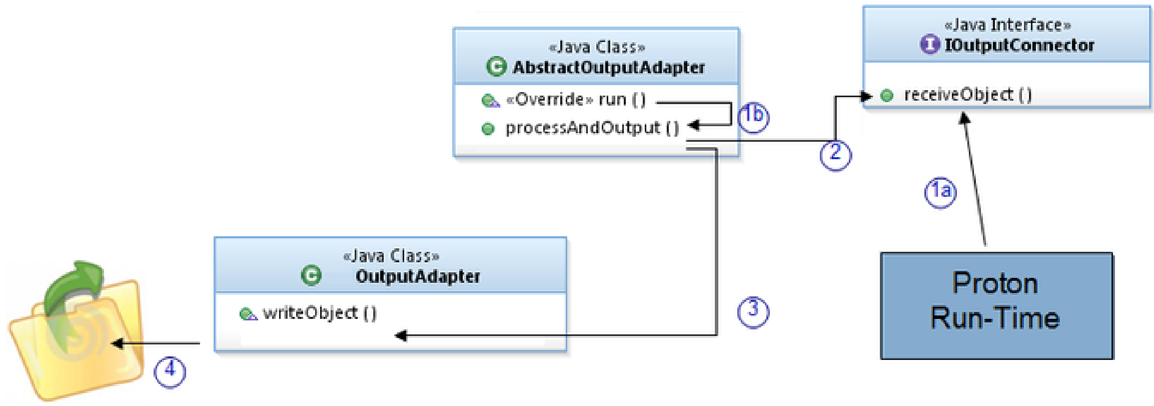
11.5.2.1 *Runtime*

The adapter representing the consumer is configured, upon startup it is supplied with server connector which handles all communication of CEP runtime with the adapter (see the figure below for initialization sequence diagram).



Adapter initialization sequence-establishing connection to CEP server

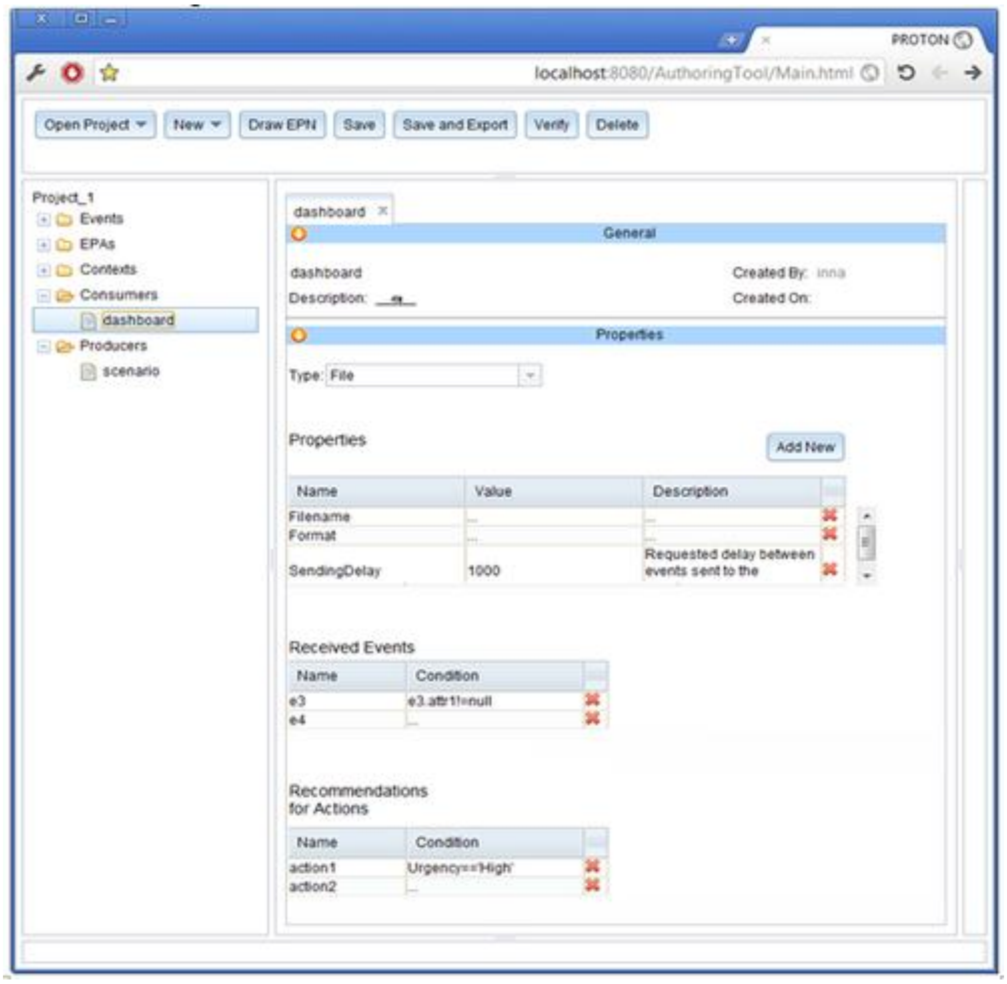
The CEP runtime pushes all published events for the specific consumer to the consumer's connector object, where it is stored in the queue. (see step 1a in the figure below). The output adapter accesses the queue and pulls the event objects from the queue. Once the adapter starts running (see the processing sequence in the figure below), it constantly polls the server connector for new published event objects (1b,2), transform the event data entry received from CEP runtime into resource-specific format(3), and writes the transformed object to the destination resource (4).



Output adapter processing sequence

11.5.2.2 *Definition*

In order to define a consumer we use the buildtime tool to choose the consumer type and define suitable properties. The figure below depicts a consumer definition screen in the CEP authoring tool.



Buildtime representation of a consumer definition

In order to define a consumer we need to supply the following information:

- The general metadata including the name of the consumer, the description, the createdDate and createdBy information.
- The type of the consumer – at the moment supporting FILE , JMS and REST client adapter. DB adapter is currently not implemented, and there is an infrastructure to allow the user to add implementation of a custom adapter.
- The list of all event types this consumer is interested to receive. For each event a filtering condition might also be specified – only instances satisfying this condition will be delivered to the consumer
- Per specific chosen type – list of properties specifying the resource to access, the credentials to access the resource, and the formatter information
    - File adapter(see the figure below) the relevant properties include
        - The absolute path to the file representing the resource this file adapter is writing to
        - Formatter type for the entries within the file : at the moment supporting tag-delimited formatter

- Properties of the tag-delimited formatter, including the delimiter ,and the tag-data separator characters


```
{"name":"consumer1","type":"file",
   "properties":[
      {"name":"filename","value":"C:\\output.txt"},
      {"name":"formatter","value":"tag"},
      {"name":"delimiter","value":";"},
      {"name":"tagDataSeparator","value":"="}
      ]
   ,"desription":"","createdDate":"","createdBy": "inna",
   "events":
      [
         {"name":"NewRoute"},{"name":"NewLocation", "condition":"NewLocation.deliveryId  ='1'"}
      ]
   }
```


Consumer of file type JSON definition

- JMS adapter (see the figure below) : the relevant properties include
  - The hostname of the server where the output JMS destination resides
  - The port to connect to on the server where the output JMS destination resides
  - The JNDI name of the connection factory object
  - The JNDI name of the destination object


```
{"name":"consumer3","type":"jms","properties":
    [
        {"name":"hostname","value":"localhost"},
        {"name":"port","value":"2809"},
        {"name":"connectionFactory","value":"jms/ProtonOutputQueueCF"},
        {"name":"destinationName","value":"eis/jms/protonOutputQueue"}
    ],
    "desription":"","createdDate":"","createdBy": "",
    "events":[{"name":"NewRoute"},{"name":"NewLocation"}]}
```


Consumer of type JMS object message JSON definition

If those are the only properties mentioned, the JMS consumer assumes the JMS destination will consume serializable objects which implement the IObjectMessage interface (see later in the description of interfaces) and creates an implementation instance of such interface which it places on the JMS destination. We can specify additional options for the formatter, in which case the JMS adapter implementation assumes the JMS message is a tag-delimited text message with the specified formatting information. Additional properties for JMS consumer which wishes to write formatted text messages to the JMS destination are: (see the figure below)

- Formatter – the formatter type (right now only tag-delimited messages are supported so the only option is 'tag')
- Delimiter – the delimiter string between the tag-data groups
- TagDataSeparator – the separator within the tag-data pair

```
{"name":"consumer3","type":"jms","properties":
    [
        {"name":"hostname","value":"localhost"},
        {"name":"port","value":"2809"},
        {"name":"connectionFactory","value":"jms/ProtonOutputQueueCF"},
        {"name":"destinationName","value":"eis/jms/protonOutputQueue"},
        {"name":"formatter","value":"tag"},
        {"name":"delimiter","value":";"},
        {"name":"tagDataSeparator","value":"="}
    ],
    "desription":"","createdDate":"","createdBy": "",
    "events":[{"name":"NewRoute"},{"name":"NewLocation"}]}
```


Consumer of type JMS formatted text message JSON definition

- REST adapter (see the figure below) , is a REST web-service client, which is capable to access the web-service declared by the consumer and push CEP events into it. The relevant definitions include:
    - URL - the fully qualified URL of the web service for event push operation.
    - contentType - can be "text/plain", "text/xml", "application/xml", "application/json" etc. This is basically defined by the web service and have to be entered here so the client knows how to access the web service.
    - Formatter properties - the same as in file.
    - Properties of the tag-delimited formatter, including the delimiter ,and the tag-data separator characters
    - A list of event types (either raw or derived) which should be delivered to this consumer


```
{"name":"consumer3","type":"rest","properties":
        [
        {"name":"URL","value":"http://localhost:9080/RESTWeb/rest/helloworld/consumer"},
        {"name":"contentType","value":"text/plain"},
        {"name":"formatter","value":"tag"},
        {"name":"delimiter","value":";"},
        {"name":"tagDataSeparator","value":"="}
        ],
"desription":"","createdDate":"","createdBy": "inna",
"events":[
        {"name":"NewRoute"},{"name":"PredictedArrivalTime"},{"name":"Reroute"},
        {"name":"BeginShipment"},{"name":"TrafficAlert"},{"name":"FlightCargoUpdate"}
        ]
}
```


Consumer of REST type JSON definition

11.5.2.3 *Interfaces to implement for custom adapter*

Any newly added custom output adapter needs to implement a few interfaces:

1. It should extend the ***AbstractOutputAdapter*** abstract class, which in turn implements the ***IOutputAdapter*** interface , the developer should provide implementation for the following methods:

- ***IOutputAdapterConfiguration createConfiguration(ConsumerMetadata metadata)*** - creates the adapter-specific configuration object after extracting the relevant properties from the consumer metadata object
- ***void initializeAdapter()*** - a method existing in the abstract interface but which should be overloaded with any resource-specific initialization after the call to the parent's method (such as acquiring a handle to a file, opening a connection to datasource etc.)
- ***void writeObject(IDataObject dataObject)*** – which takes the CEP data object, transforms it to resource-specific format and writes it to the resource represented by this consumer.
- ***void shutdownAdapter()*** – a method existing in the abstract interface but which should be overloaded with any resource-specific shutdown actions after the call to the parent's method (such as closing a handle to a file, closing a connection to datasource etc.)

2. It should provide an adapter specific implementation of ***IOutputAdapterConfiguration*** interface for an adapter-specific configuration object. This is basically a bean with getters method carrying adapter specific information which helps the adapter to access the specific resource (such as filename for file adapter, or hostname and port name for JMS server for the JMS adapter)

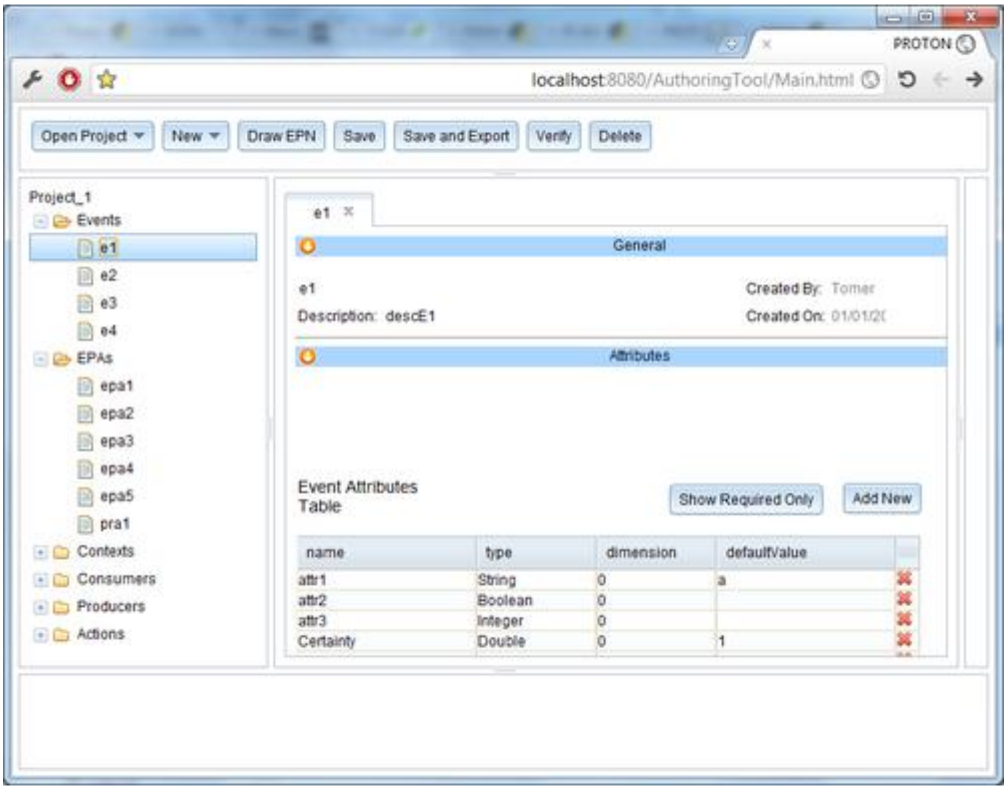
11.5.3 Definition of CEP Application

CEP definition file is created using the CEP build-time web based user interface. Using this UI, the application developer creates the building blocks of the application definitions. This is done by filling up forms without the need to write any code. Alternatively this definition file, in JSON format, can be generated programmatically by any other application and fed to the CEP engine.

The building blocks of a CEP application are:

- Event type – the events that are expected to be received as input or to be sent as output. An event type definition includes the event name and a list of its attributes.
- Producers – the event sources and the way CEP gets events from those sources.
- Consumers – the event consumers and the way they get derived events from CEP.
- Temporal Contexts – time windows contexts in which event processing agents are active.
- Segmentation Contexts – semantic contexts that are used to group several events to be used by the event processing agents.
- Composite Contexts – group together several different contexts.
- Event processing agents – patterns above incoming events in specific context that detect situations and generate derived events.

The UI (see a figure below) allows defining an CEP application, to examining the event processing network of this application, to validate it and to export the event processing network definition to a file. This file is exported in a JSON format and is fed to the CEP engine.



CEP Web based User interface for application definition

11.6 Basic Design Principles

- The EPN application definition is done using a user interface without the need to write any code, with intention for visual programming.
- The CEP application is composed from a network of Event Processing Agents. This allows the agents to run in parallel and to be distributed on several machines.
- Logical Event Processing Network (EPN) definition which is decoupled from the actual running configuration. The same EPN can run on a single machine or be distributed on several machines.
- The event producers and event consumers can be distributed among different machines.
- Event producers and consumers are totally decoupled.
- Adapter framework that is extensible to allow adding any type of custom adapter for sending or receiving events.
- The expression language is extensible and functions can be added if needed.

11.7 References

[EPIA]	O. Etzion and P. Niblett, Event Processing in Action, Manning Publications, 2010.

12 FIWARE ArchitectureDescription Data Location

You can find the content of this chapter as well in the wiki of fi-ware.

12.1 Copyright

12.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

12.3 Overview

12.3.1 Introduction to the Data Location GE

The Location Platform provides location-based services to two types of users:

- Third-party location clients

  Typically yellow pages is one of many third-party location clients that can interact with the location platform using the ***Mobile Location Protocol*** (MLP, [1]) interface or ***RESTful Network API for Terminal Location*** ([2]) both standardized by Open Mobile Alliance (OMA, [3]).

  These interfaces facilitate many services to retrieve the position of a compatible target mobile terminal for various types of applications, ranging from single shot location retrieval to area event retrieval (geo-fencing).

  The target mobile terminal position is retrieved using Assisted Global Positioning System (AGPS), WiFi and Cell-Id positioning technologies intelligently triggered depending on end-user environment and location request content (age of location, accuracy, etc.).

- Mobile end-users

  When an end-user searches for its position using a compatible terminal via any kind of application requiring location information, the terminal connects to the location platform to exchange assistance data in order to compute or retrieve its position, as negotiated between the terminal and the platform.

  Moreover, some applications on the compatible terminal may include the sharing of location information with external third-parties, including other end-users.

  Such service relies on another OMA standard, called ***Secure User Plane*** (SUPL, [4]).

In both scenarios, the target handset to localize must comply with the following requirements:

- 3G capable
- WiFi optional
- Be equipped with an assisted GPS chipset
- Support Secure User Plane (SUPLv2) stack

12.3.2 Target usage

The Location GE in FI-WARE targets any third-party application, GEs in FI-WARE, or any complementary platform enabler, that aims to retrieve mobile device positions and area events. The Location GE is based on various positioning techniques such as A-GPS, WiFi and Cell-Id intelligently triggered whilst taking into account the end-user privacy. Note that the location retrieval from the end-user itself is out of scope for FI-WARE.

This GE addresses issues related to Location of mobile devices in difficult environments such as urban canyons and light indoor environments where the GPS receiver in the mobile device is not able to acquire weak GPS signals without assistance. In more difficult conditions like deep indoor, the Location GE selects other positioning techniques like WiFi to locate the end-user. It therefore improves localization yield which enhances the end-user experience and the performance of applications requesting the position of mobile devices.

12.4 Basic Concepts

12.4.1 Third-party location services

Services provided for third-party location clients are standardized under the name "network-initiated" procedures, since the location request is established somewhere from an application on the mobile operator or external network. Such an external network can be the Internet, since both ***MLP*** and ***NetAPI Terminal Location*** protocols are HTTP based. Please note that those services require ***SUPL*** interface towards the compatible terminal, which is based on TCP/IP.

The following ***MLP*** services are supported by the location platform:

- Synchronous and asynchronous Standard Location Immediate Service, which provides immediate location retrieval of a target terminal for standard LBS applications (non-emergency),
- Synchronous and asynchronous Emergency Location Immediate Service, which provides immediate location retrieval of a target terminal for emergency LBS applications,
- Triggered Location Reporting Service, which facilitates the retrieval of periodic location or event reports from a target terminal in order to track an end-user using reported positions or reported events, such as specific zone entry.

Similar services are available on the ***NetAPI Terminal Location*** interface with limited functionality:

- Location Query: provides immediate location retrieval of a target terminal,
- Periodic Notification Subscription: facilitates the retrieval of periodic location reports from a target terminal,
- Area (Circle) Notification Subscription: facilitates the retrieval of area event reports from a target terminal (geofencing).

12.4.2 Access control and privacy management

Fortunately, not all applications can access to these location services. Strong access control and privacy management rules are applied to authenticate a third-party and authorize it to localize a particular end-user terminal for a specific location service. The third-party is authenticated thanks to the credentials provided in the location request and compared with the content of the location platform database. Once the third-party is authenticated, the service requested is checked against the internal database to retrieve the service settings, further detailed in this document. Based on those settings and end-user customization, the service is allowed to localize the end-user.

12.4.3 Mobile end-user services

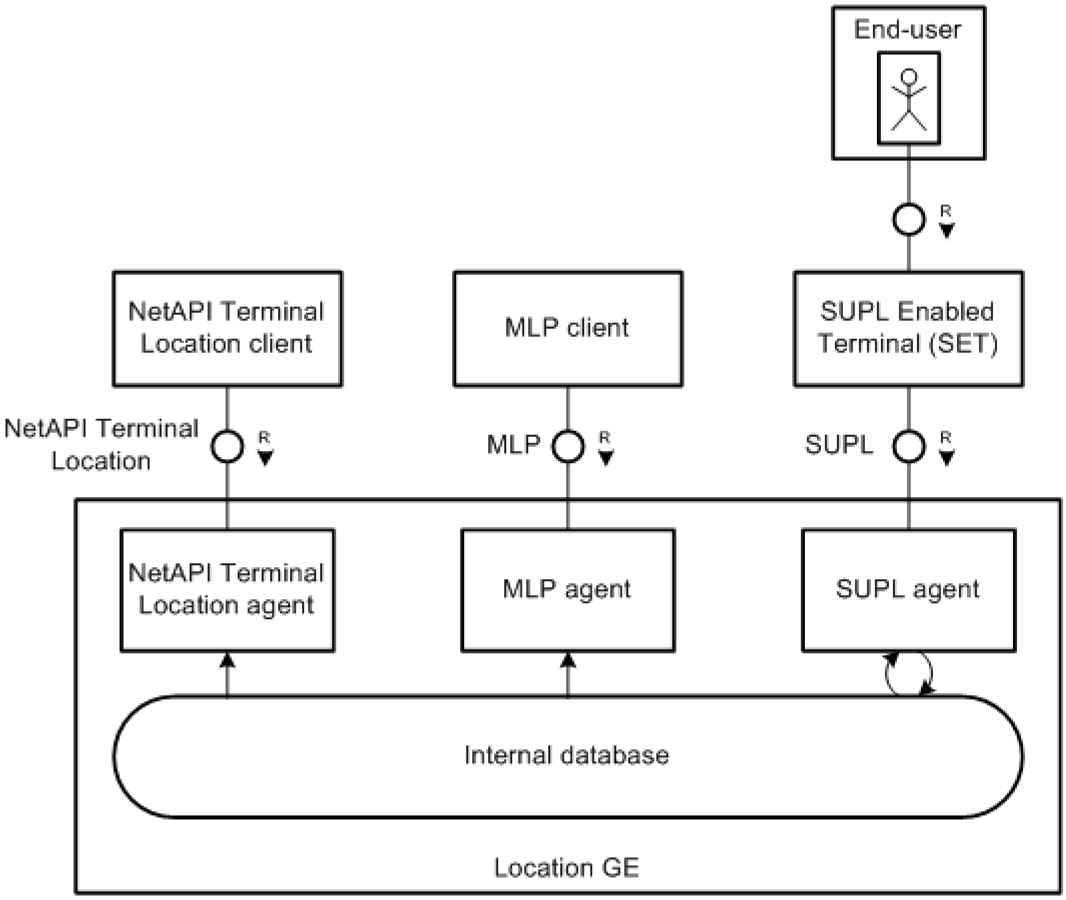
Services provided for mobile end-users are standardized under the name "set-initiated" procedures, since the location request is established by the SUPL Enabled Terminal (SET) on behalf of the end-user launching the application requiring location information.

The following set-initiated services are supported by the location platform, however not exposed to FI-WARE developers since rely on more complex protocols (TCP/ASN.1) than network-initiated services that expose a simple RestFul API.

- Standard location request: the SET requests its actual position, for example to be displayed on a map.
- Location request with transfer to third party: the SET requests its actual position and requests it to be sent to a third-party, based on the third-party information (credentials) provided. This feature is mainly used for social networks.
- Periodic trigger: the SET requests on a periodic basis its actual position, for example for navigation purposes.

12.4.4 Interfaces and data model

The following diagram illustrates the interfaces previously presented:



The following agents are the grounds of the Location GE:

- ***MLP*** agent: made of an HTTP stack, it processes MLP compliant requests and after authorization of such request, it triggers the SUPL agent to establish communication with the target handset to retrieve location or events depending on the content of the request. Such request is encoded in XML format fully specified in MLP standard.
- ***NetAPI Terminal Location*** agent: similar to MLP agent, it decodes HTTP requests using RESTFul procedures and once authenticated triggers the establishment of a SUPL connection with the target handset for similar services to MLP.
- ***SUPL*** agent: made of a TCP stack, this server is used both to establish communication with a target handset and receive connection from the handset. The SUPL server implements SUPL standardized procedures based on ASN1. Such procedures include single shot location retrieval and triggers used for periodic and area event tracking. Such interface is also used to exchange GPS assistance data via the 3GPP RRLP protocol encapsulated in the SUPL payload.

The mySQL internal database, shared between agents, contains the following data:

- Network cell information: cell identifiers associated with cell mast position and coverage radius to be currently provided by Telco. A dynamic provisioning is planned for future FI-WARE releases in order to build this database with GPS location and cell information retrieved from the SET.
- Third-party information: third-party account credentials and settings.

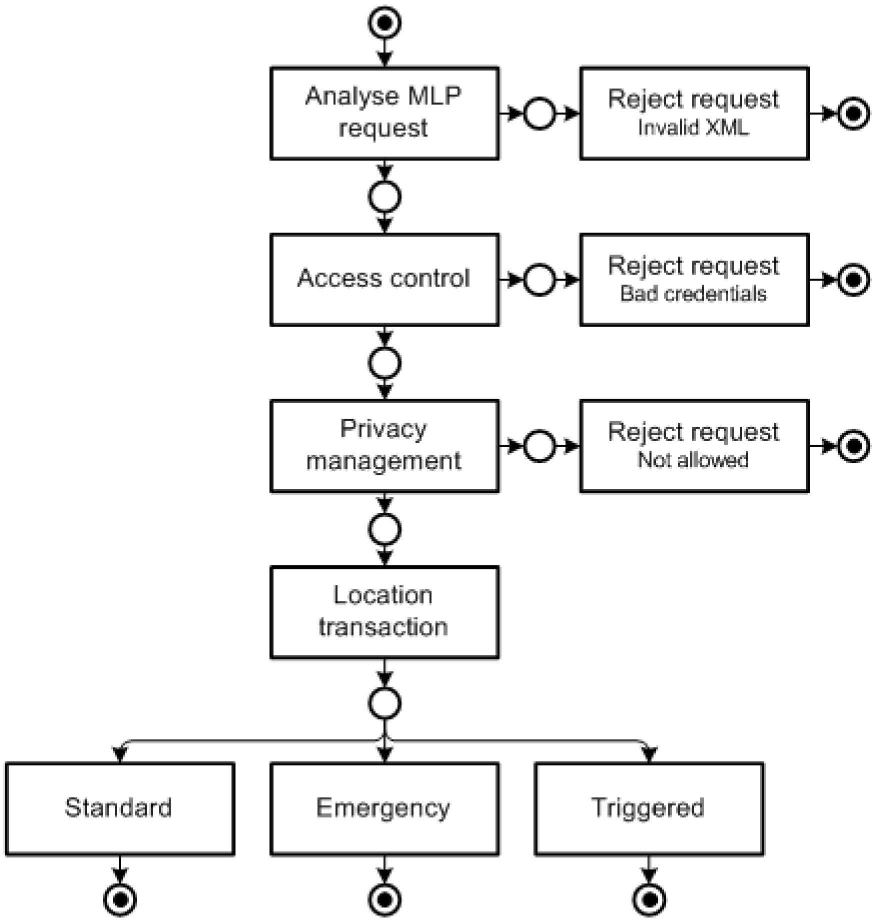
- Third-party location services information: contains many parameters, including level of authority (lawful/standard), authorized level of accuracy (low/high), type of location authorized (standard/emergency/tracking), flow control parameters.
- User information: contains many parameters, including friends list, global settings for authorizing localization and position caching of all location services.
- User privacy policy: overlays service settings for a specific end-user. Many parameters are also available, including service authorization (permanent/one-shot/time-based), position caching authorization.
- User position cache: if activated in user privacy policy, each actual position retrieved is stored locally in the location platform database. This is mainly used by third-party location services that do not need necessarily an actual location.

The provisioning interface of such database is currently not exposed to FI-WARE developers, access to the database is reserved to Location GE administrators.

12.5 Main Interactions

12.5.1 MLP services

Each MLP service is triggered by different types of location request, further detailed in this section. Each location request is first analyzed for access and privacy management before processing the actual location transaction, as illustrated on the below diagram:



Access control and privacy management

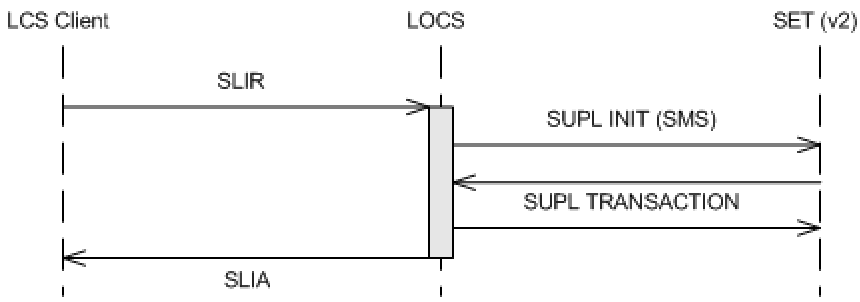
Each of the incoming MLP requests are checked for authentication and authorization before localizing the end-user. The following example shows the MLP request header:

```
<?xml version="1.0" ?>
<svc_init ver="3.2.0">
  <hdr ver="3.2.0">
    <client>
      <id>login</id>
      <pwd>password</pwd>
      <serviceid>servicename</serviceid>
    </client>
    <requestor type="MSISDN">
      <id>33612345680</id>
    </requestor>
  </hdr>
  <!-- Location request -->
</svc_init>
```

The <client/> section contains the elements required for authentication and facilitates the retrieval of the third-party location service requested. The <requestor/> element is used for checking the friends list of the target end-user, identified by its MSISDN. The <serviceid/> and target end-user MSISDN are utilized to check the end-user privacy policy previously presented.

Standard Location Immediate Service

This service facilitates the location retrieval of the handset on a one-shot basis. The sequence of messages is illustrated below:



It is triggered by an MLP SLIR request, as follows:

```
<slir ver="3.2.0" res_type="SYNC">
    <msids>
      <msid type="MSISDN">33612345678</msid>
      <msid type="MSISDN">33612345679</msid>
    </msids>
    <eqop>
      <hor_acc>1000</hor_acc>
    </eqop>
```

```
    <loc_type type="CURRENT_OR_LAST" />
  </slir>
```

This request triggers a standard network-initiated SUPL transaction towards the handset. Once the handset location is retrieved, the Location Platform responds with a SLIA response, containing the position of the target end-user:


```
<slia ver="3.2.0" >
  <pos pos_method="CELL">
    <msid type="MSISDN">33612345678</msid>
    <pd>
      <time>20020623134453</time>
      <shape>
        <EllipticalArea>
             <coord>
              <X>50.445668</X>
              <Y>2.803677</Y>
             </coord>
             <angle>0.0</angle>
             <semiMajor>707</semiMajor>
             <semiMinor>707</semiMinor>
             <angularUnit>Radians</angularUnit>
        </EllipticalArea>
      </shape>
      <alt>0</alt>
      <alt_unc>707</alt_unc>
    </pd>
  </pos>
  <pos>
    <msid>33612345679</msid>
    <pd>
      <time>20020623134454</time>
      <shape>
        <EllipticalArea>
             <coord>
              <X>50.445668</X>
              <Y>2.803677</Y>
             </coord>
             <angle>0.0</angle>
             <semiMajor>707</semiMajor>
             <semiMinor>707</semiMinor>
             <angularUnit>Radians</angularUnit>
        </EllipticalArea>
      </shape>
      <alt>0</alt>
      <alt_unc>707</alt_unc>
    </pd>
  </pos>
</slia>
```

Emergency Location Immediate Service

This service facilitates the location retrieval of the handset on a on-shot basis for emergency purposes. It is triggered by an MLP EME_LIR request instead of a SLIR, as follows:

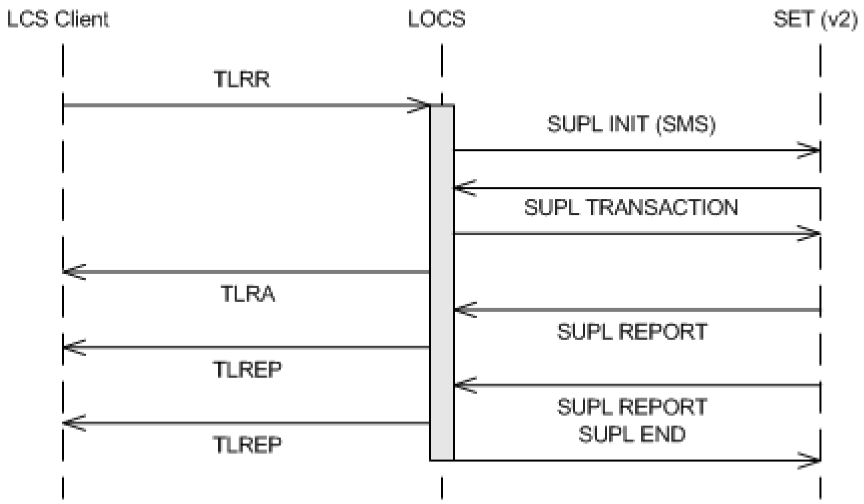
```
<eme_lir ver="3.2.0">
  <msids>
    <msid type="MSISDN">33612345678</msid>
  </msids>
  <loc_type type="CURRENT_OR_LAST" />
</eme_lir>
```

This request triggers an emergency network-initiated SUPL transaction towards the handset. Once the handset location is retrieved, the Location Platform responds with an EME_LIA response instead of a SLIA, containing the position of the target end-user:

```
<eme_lia ver="3.2.0">
  <eme_pos>
    <msid type="MSISDN">33612345678</msid>
    <pd>
      <time>20020623134454</time>
      <shape>
        <EllipticalArea>
             <coord>
              <X>50.445668</X>
              <Y>2.803677</Y>
             </coord>
             <angle>0.0</angle>
             <semiMajor>707</semiMajor>
             <semiMinor>707</semiMinor>
             <angularUnit>Radians</angularUnit>
        </EllipticalArea>
      </shape>
      <alt>0</alt>
      <alt_unc>707</alt_unc>
    </pd>
  </eme_pos>
</eme_lia>
```

12.5.1.1 *Triggered Location Reporting Service*

This service facilitates the periodic location or event-based reports retrieval from the handset. The message sequence is illustrated below:



It is triggered by an MLP TLRR request, as follows:

```
<tlrr ver="3.2.0">
  <msids>
    <msid type="MSISDN">33612345678</msid>
  </msids>
  <interval>00003000</interval>
  <start_time>20021003112700</start_time>
  <stop_time>20021003152700</stop_time>
  <qop>
    <hor_acc>100</hor_acc>
  </qop>
  <pushaddr>
    <url>http://location.application.com</url>
  </pushaddr>
  <loc_type type="CURRENT"/>
</tlrr>
```

The Location Platform acknowledges the request with a TLRA when the SUPL transaction confirmed that the target SET received all trigger parameters and has exchanged eventually assistance data if needed. The TLRA only contains a unique transaction identifier that can be used to map trigger reports with the original location request:

```
<tlra ver="3.2.0">

  <req_id>25293</req_id>

</tlra>
```

Each location/event report returned by the handset via SUPL is returned in a TLREP, as follows:

```
<tlrep ver="3.2.0">
  <req_id>25293</req_id>
  <trl_pos trl_trigger="PERIODIC">
```

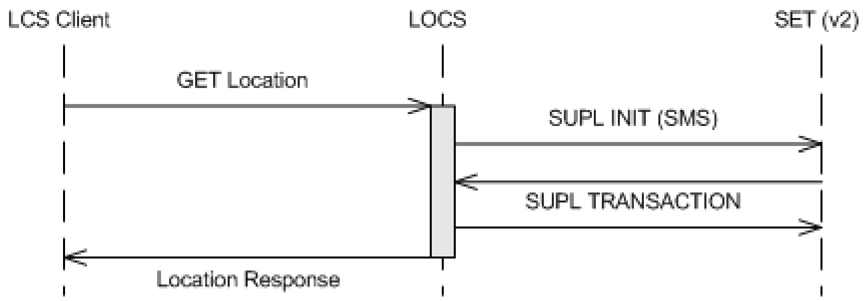
```
    <msid type="MSISDN">33612345679</msid>
    <pd>
      <time>20020623134453</time>
      <shape>
        <EllipticalArea>
             <coord>
              <X>50.445668</X>
              <Y>2.803677</Y>
             </coord>
             <angle>0.0</angle>
             <semiMajor>707</semiMajor>
             <semiMinor>707</semiMinor>
             <angularUnit>Radians</angularUnit>
        </EllipticalArea>
      </shape>
      <alt>0</alt>
      <alt_unc>707</alt_unc>
    </pd>
  </trl_pos>
</tlrep>
```

12.5.2 NetAPI Terminal Location services

As stated before, the NetAPI Terminal Location interface provides similar services to MLP with some limitations. The main interactions between third-party application and Location GE are presented in this chapter. XML location request content type is supported in current FI-WARE release. Support of json and url-form-encoded content types will be soon added as specified in Location GE API. The following section present XML content type.

12.5.2.1 *Location Query*

The Location Query facilitates the retrieval of the current location of a target terminal. The message sequence is illustrated on the following diagram:



The Location GE receives an HTTP GET request including many parameters that are used for the authentication of the third-party application and quality of position parameters that define the type of location requested. The full list of supported parameters are provided in API (see references). An example of a request is provided below:

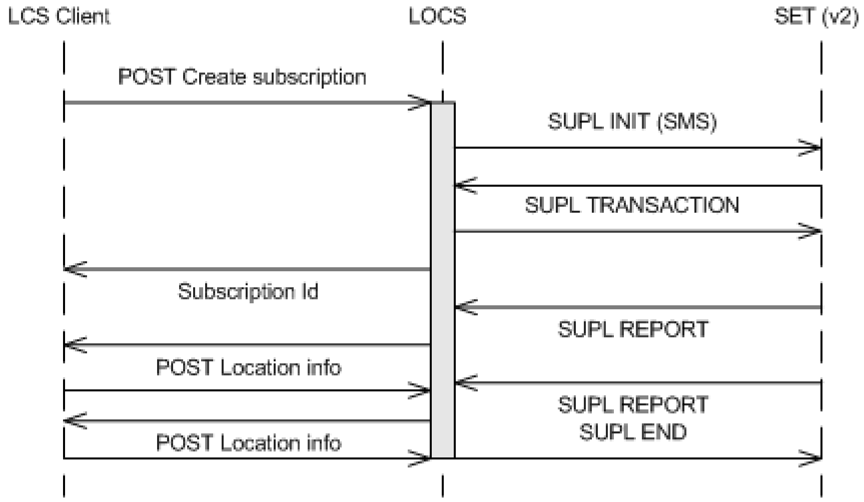
```
GET
/location/v1/queries/location?requester=test:test&address=3361122334
4&requestedAccuracy=50&acceptableAccuracy=60
 &maximumAge=100&tolerance=DelayTolerant HTTP/1.1
 Accept: application/xml
 Host: example.com
```

Once authenticated, the location request triggers a SUPL transaction towards the target handset to retrieve its location. When retrieved the following content is returned:


```
 HTTP/1.1 200 OK
 Content-Type: application/xml
 Content-Length: nnnn
 Date: Thu, 02 Jun 2011 02:51:59 GMT

 <?xml version="1.0" encoding="UTF-8"?>
 <tl:terminalLocationList
xmlns:common="urn:oma:xml:rest:netapi:common:1"
xmlns:tl="urn:oma:xml:rest:netapi:terminallocation:1">
  <tl:terminalLocation>
   <tl:address>33611223344</tl:address>
   <tl:locationRetrievalStatus>Retrieved
   </tl:locationRetrievalStatus>
   <tl:currentLocation>
    <tl:latitude>49.999737</tl:latitude>
    <tl:longitude>-60.00014</tl:longitude>
    <tl:altitude>30.0</tl:altitude>
    <tl:accuracy>55</tl:accuracy>
    <tl:timestamp>2012-04-17T09:21:32.893+02:00</tl:timestamp>
   </tl:currentLocation>
   <tl:errorInformation>
    <common:messageId>QOP_NOT_ATTAINABLE</common:messageId>
    <common:text>The requested QoP cannot be provided.</common:text>
   </tl:errorInformation>
  </tl:terminalLocation>
 </tl:terminalLocationList>
```


12.5.2.2 *Location Subscriptions*

This type of query is used to retrieve either periodic location reports or area entry/leaving/inside/outside type events from a target terminal. The message flow is illustrated below:



The Location GE receives in this case an HTTP POST method including many parameters that are used for the authentication of the third-party application and quality of position parameters that define the type of location/events requested.The full list of supported parameters are provided in API (see references). An example of a request is provided below:

```
 POST /location/v1/subscriptions/periodic HTTP/1.1
 Accept: application/xml
 Host: example.com
 Content-Length: nnnn

 <?xml version="1.0" encoding="UTF-8"?>
 <tl:periodicNotificationSubscription
xmlns:common="urn:oma:xml:rest:netapi:common:1"
xmlns:tl="urn:oma:xml:rest:netapi:terminallocation:1">
 <tl:clientCorrelator>0003</tl:clientCorrelator>
 <tl:callbackReference>

<tl:notifyURL>http://application.example.com/notifications/LocationN
otification</tl:notifyURL>
   <tl:callbackData>4444</tl:callbackData>
 </tl:callbackReference>
 <tl:address>tel:+19585550100</tl:address>
 <tl:requestedAccuracy>10</tl:requestedAccuracy>
 <tl:frequency>10</tl:frequency>
 <tl:duration>100</tl:duration>
 </tl:periodicNotificationSubscription>
```

Once authenticated, the location request triggers a SUPL transaction towards the target handset to program it with requested information. When acknowledged by the handset, the following response is returned:

```
 HTTP/1.1 201 Created
```

```
 Content-Type: application/xml
 Location:
http://example.com/location/v1/subscriptions/periodic/sub003
 Content-Length: nnnn
 Date: Thu, 02 Jun 2011 02:51:59 GMT

 <?xml version="1.0" encoding="UTF-8"?>
 <tl:periodicNotificationSubscription
xmlns:common="urn:oma:xml:rest:netapi:common:1"
xmlns:tl="urn:oma:xml:rest:netapi:terminallocation:1">
 <tl:clientCorrelator>0003</tl:clientCorrelator>

<tl:resourceURL>http://example.com/location/v1/subscriptions/area/ci
rcle/sub003</tl:resourceURL>
 <tl:callbackReference>

<tl:notifyURL>http://application.example.com/notifications/LocationN
otification</tl:notifyURL>
 <tl:callbackData>4444</tl:callbackData>
 </tl:callbackReference>
 <tl:address>tel:+19585550100</tl:address>
 <tl:requestedAccuracy>10</tl:requestedAccuracy>
 <tl:frequency>10</tl:frequency>
 <tl:duration>100</tl:duration>
 </tl:periodicNotificationSubscription>
```

Each location / event report sent by the SET to the Location GE is then forwarded to the client application using a POST method containing the following data:

```
 POST /notifications/LocationNotification HTTP/1.1
 Content-Type: application/xml
 Accept: application/xml
 Host: application.example.com
 Content-Length: nnnn

 <?xml version="1.0" encoding="UTF-8"?>
 <tl:subscriptionNotification
xmlns:common="urn:oma:xml:rest:netapi:common:1"
xmlns:tl="urn:oma:xml:rest:netapi:terminallocation:1">
 <tl:callbackData>4444</tl:callbackData>
 <tl:terminalLocation>
 <tl:address>tel:+19585550100</tl:address>
 <tl:locationRetrievalStatus>Retrieved</tl:locationRetrievalStatus>
 <tl:currentLocation>
 <tl:latitude>-80.86302</tl:latitude>
 <tl:longitude>41.277306</tl:longitude>
 <tl:altitude>1001.0</tl:altitude>
 <tl:accuracy>100</tl:accuracy>
 <tl:timestamp>2011-06-02T00:27:23.000Z</tl:timestamp>
 </tl:currentLocation>
 </tl:terminalLocation>
 <tl:link rel="CircleNotificationSubscription"
href="http:/location/v1/subscriptions/periodic/sub0003"/>
 </tl:subscriptionNotification>
```

12.5.3 Positioning

12.5.3.1 *A-GNSS location technology*

In all SUPL transactions presented before, the Location GE and the SET may exchange GNSS (Global Navigation Satellite System) assistance data in order to improve mainly time to first fix and handset sensitivity. SUPL is used as transport layer to carry the following assistance data encoded in RRLP (Radio Resource Location Protocol) format:

- Almanac
- UTC model
- Ionospheric model
- DGPS corrections
- Reference location
- Reference time
- Acquisition assistance
- Real-time integrity
- Navigation model

Based on this assistance data, the handset only needs to acquire satellites and use the provided information to either compute its position (ms-based mode) or provide its pseudo-range measurements to the Location GE to get its position (ms-assisted mode).

12.5.3.2 *C-ID location technology*

The first SUPL message sent by the handset contains location identifier(s). Those identifiers can be of type 'GSM', 'WCDMA' (3G) or 'WLAN' (WiFi). Based on the internal database, the Location Platform is able to convert those identifiers into a position and, in case of multiple location identifiers, perform triangulation of those access points.

12.5.3.3 *Location Technology selection*

Today, C-ID (including WiFi) is always used based on the location identifiers returned by the SET, as part of the SUPL exchange. A-GPS is only used if the third-party application is authorized to perform precise positioning. An evolution of the location technology selection is planned in future FI-WARE releases, as described below.

Depending on the content of the location request and the end-user environment recognized by its cell, the Location GE will decide what Location Technology to use. The following parameters will contribute to this decision:

- End-user environment: indoor, outdoor
- QoP parameters: delay, accuracy
- Client type: standard or emergency

The intelligence and innovation of the Location GE lies in this section. The Location GE will be able to select dynamically what location technology is the most relevant based on the third-party application needs and end-user environment.

A future evolution includes also the dynamic provisioning of the internal cell-id database where the Location GE will trigger standalone GPS technique to automatically record the retrieved GPS position against the cell identifiers.

All those evolutions will be fully described in future FI-WARE releases.

12.6 Basic Design Principles

The Location GE is based on existing OMA standards (refer to [5]):

- MLP: DTDs are available from the OMA website.
- NetAPI Terminal location: refer to Location GE RESTful API
- SUPL: ASN.1 data format is provided as part of the SUPL specification.

The 3GPP RRLP standard is also followed for GNSS assistance data exchange.

12.7 References

MLP	Mobile Location Protocol (MLP), Open Mobile Alliance, specification OMA-TS-MLP-V3_2-20110719-A
SUPL	Secure User Plane Location Protocol (SUPL), Open Mobile Alliance, specification OMA-TS-ULP-V2_0-20111222-D
RRLP	Radio Resource LCS Protocol (RRLP), 3GPP, specification 3GPP TS 44.031 V9.2.0 (2010-03)

13 FIWARE ArchitectureDescription Data MetadataPreprocessing

You can find the content of this chapter as well in the wiki of fi-ware.

13.1 Copyright

- Copyright © 2012 by SIEMENS

13.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

13.3 Overview

13.3.1 Target usage

Target users are all stakeholders that need to convert metadata formats or need to generate objects (as instantiation of classes) that carry metadata information. The requirements to transform metadata typically stem from the fact that in real life various components implementing different metadata formats need to inter-work. However, typically products from different vendors are plugged together. In this case, the Metadata Preprocessing GE acts as a mediator between the various products.

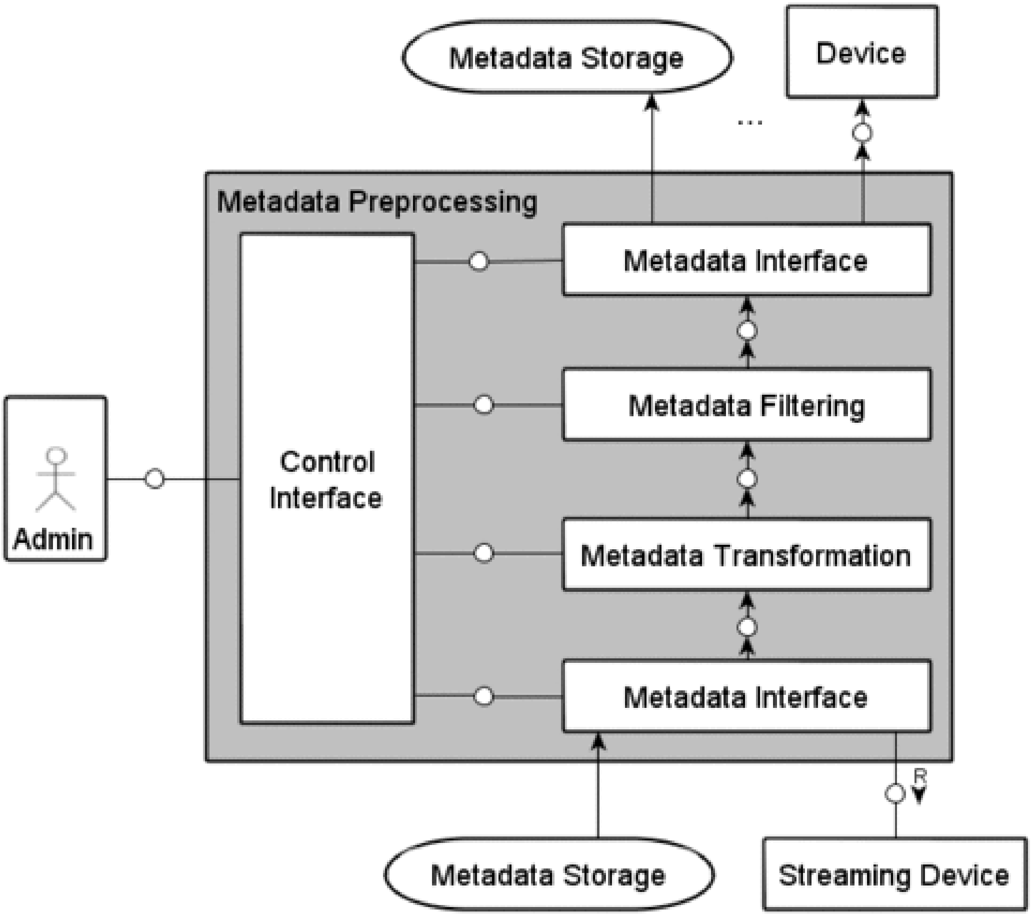
13.3.2 Example scenarios and main services exported

The Metadata Preprocessing GE is typically used for preparing metadata coming from a data-gathering device for subsequent use in another component. The data-gathering device can be a sensor, e.g., the analytics component of a surveillance camera. Depending on the manufacturer of the camera, different metadata schemes are used for structuring the metadata. The Metadata Preprocessing GE generally transforms the metadata into a format that is expected by a subsequent component, e.g., a storage device. In addition to transformation of the format, also some elements of the metadata can be removed from the stream by a filtering component. This is especially useful in case these elements cannot be interpreted by the receiving component.

13.4 Basic Concepts

13.4.1 Functional components of the Metadata Preprocessing GE

The following figure depicts the components of the Metadata Preprocessing Generic Enabler. These functional blocks are the Metadata Interface for inbound streams, Metadata Transformation, Metadata Filtering, and Metadata Interface for outbound (processed) streams. The mentioned methods are described in more detail in Section “Main Interactions”.



Functional components of the Metadata Preprocessing GE

The functionality of the components is described in the following.

- **Control Interface**: The control interface is the entity for configuring and controlling the metadata processing engine. The algorithms used for transformation and filtering as well as the metadata source are configured(connected using the ***configureInstance*** method. Sinks receiving the outbound streams are connected and disconnected via the ***addSink*** and ***removeSink*** methods, respectively. More details on the APIs are described below.
- **Metadata Interface** (for inbound streams): Different interchange formats (such as ones for streaming or for file access) can be realized. An example formats is the Real-time Transport Protocol (RTP) as standardized in RFC 3550 [RFC3550]. Different packetization formats for the contained payload data (i.e., the metadata) depending on the application might be used.
- **Metadata Transformation**: The Metadata Transformation component is the core component of this Generic Enabler. Based on an XML Stylesheet Language for Transformations (XSLT) [XSLT] and a related stylesheet, the processing of the metadata is performed. In principle, also other kind of transformations (other than XSLT) can be applied. The output of this step is a new encapsulation/formatting of the metadata received. This could also be an instantiation of a class (e.g., JAVA, C++, C#, etc.)

- **Metadata Filtering**: Metadata Filtering is an optional step in the processing chain. The filtering can be used, e.g., for thinning and aggregation of the metadata, or simple fact generation (i.e., simple reasoning on the transformed metadata). Depending on the configuration of the GE, filtering can happen before, after, or even during transformation.
- **Metadata Interface** (for outbound streams): Through this interface, the transformed (and possibly filtered) metadata or metadata stream is accessed.

13.4.2 Realization by MetadataProcessor asset

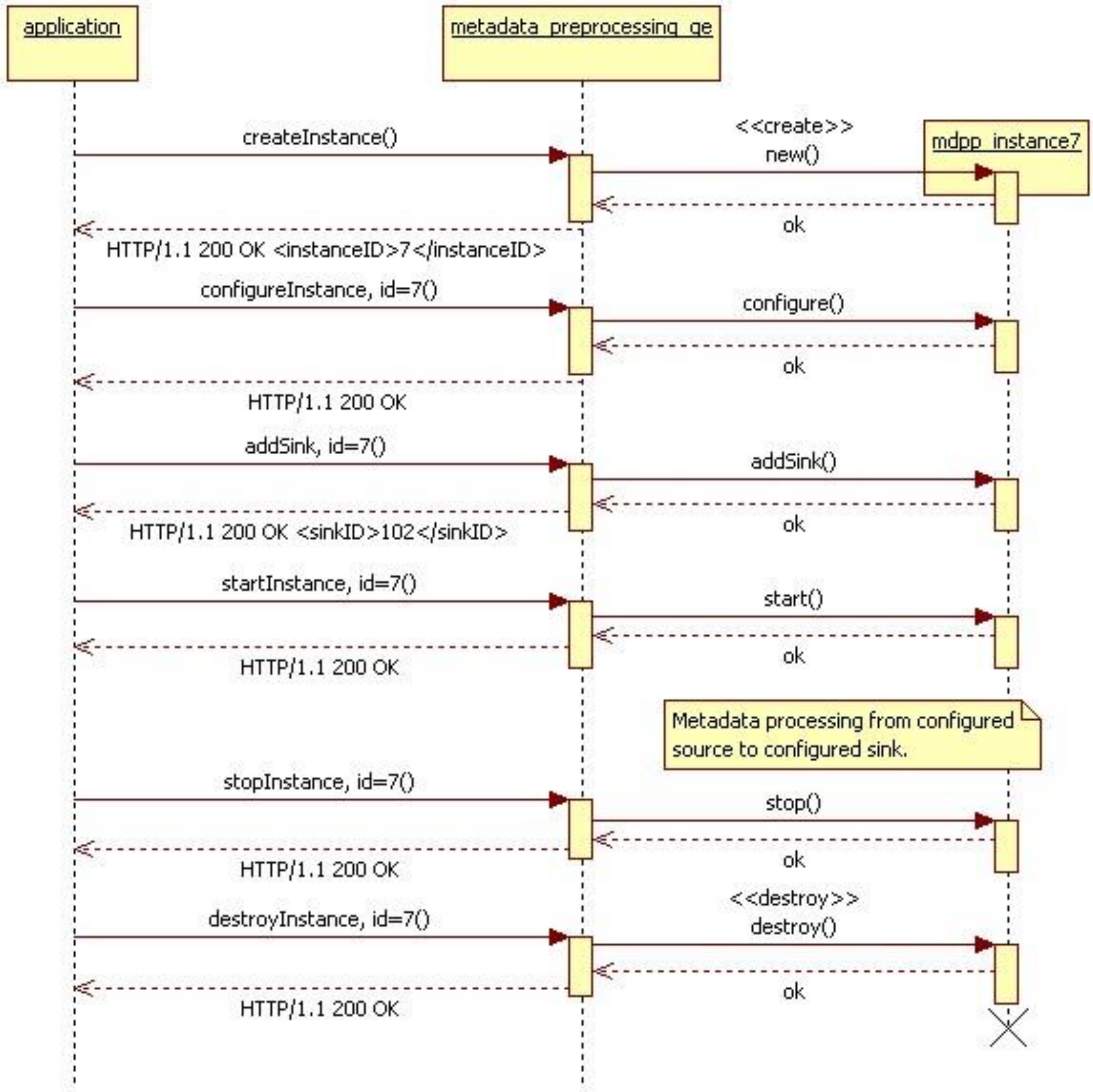
The MetadataProcessor asset realizes a specific implementation of the Metadata Preprocessing GE. Timed metadata is received over an RTSP/RTP interface, which implements the metadata interface for inbound data/streams. Different RTP sessions can be handled; therefore metadata streams can be received from several devices (e.g., cameras or other type of sensors). The target in such a realization could be the provision of metadata as facts to a metadata broker, which would be the receiver of the outbound stream.

13.5 Main Interactions

The external API is a RESTful API that permits easy integration with web services or other components requiring metadata access and transformation services (e.g., other GEs). The following interface will be supported:

- ***getVersion***: The version of the Metadata Preprocessing GE is returned.
- ***listInstances***: All instances (i.e., processing units) of the Metadata Preprocessing GE are listed.
- ***createInstance***: An instance for processing metadata streams/events is created.
- ***getInstanceInfo***: The information about a specific instance (i.e., processing unit) is returned.
- ***destroyInstance***: An existing metadata processing instanced is destroyed.
- ***startInstance***: The metadata processing (e.g., transformation and/or filtering) is started.
- ***stopInstance***: The metadata processing is stopped/halted.
- ***getConfig***: The configuration of a specific processing unit is returned.
- ***configureInstance***: A metadata source (e.g., another GE) is connected to the enabler and/or the metadata processing (e.g., the XSLT stylesheet for the conversion of metadata formats and filtering of metadata streams/events) is configured for a specific instance (i.e., processing unit).
- ***listSinks***: All sinks of a specific processing unit are listed.
- ***addSink***: A metadata sink (e.g., another GE) is connected to the enabler. Note that multiple sinks can be connected to a single instance of of the Metadata Preprocessing GE.
- ***getSinkInfo***: The information about a specific sink is returned.
- ***removeSink***: A specific metadata sink (e.g., another GE) is disconnected.

The following figure explains the main interactions in an example scenario. In the first step, a new instance for metadata processing is created. The ID of the instance is returned to the calling application/component. In a second step the processing of the Metadata Preprocessing GE is configured (e.g., by providing an XSLT stylesheet). In a third and fourth step the source and the sink of the metadata processing are configured. Note that the order of the configuration steps (i.e., ***configureInstance***, ***addSink***) is arbitrary. Note further that more than one sink can be added as receiving component, but only one source can be configured. In a fifth step, the processing is started.



Example scenario

After the processing is done, the specific instance of the GE is stopped. Note that the instance could be started again afterwards. Also the processing of the source could be reconfigured and sinks can be added or removed. As a final step in this example scenario, the specific instance of the Metadata Preprocessing instance is destroyed. Note that it is not necessary to stop the instance before destroying it, since this will be done automatically.

13.6 Basic Design Principles

The following basic design principles apply:

- The Metadata Preprocessing GE realizes a generic metadata transformation approach, which is not restrictive to specific metadata schemes.
- Encapsulation of transport and metadata transformation can be implemented as-a-service, usable from other web applications or components.
- Transformation can be based on standardized and commonly used XML Stylesheet Language for Transformations (XSLT).
- In addition to encapsulation in (XML- or JSON-based) metadata formats, also incorporation of the metadata into objects (e.g., serialized Java/C++/C# classes) can be realized (by simply exchanging the stylesheet for the XSLT).

13.7 References

[RFC3550]	H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications", RFC 3550, Jul. 2003.
[XSLT]	W3C / M. Kay (editor), "XSL Transformations (XSLT) Version 2.0", http://www.w3.org/TR/xslt20/, Jan. 2007.

14 FIWARE ArchitectureDescription Data CompressedDomainVideoAnalysis

You can find the content of this chapter as well in the wiki of fi-ware.

14.1 Copyright

14.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

14.3 Overview

14.3.1 Introduction to the Compressed Domain Video Analysis GE

The target users of the Compressed Domain Video Analysis GE are all applications that want to extract meaningful information from video content and that need to automatically find characteristics in video data bases on given tasks. The GE can work for previously stored video data as well as for video data streams (e.g., received from a camera in real time).

In the media era of the web, much content is user-generated (UGC) and span over any possible kind, from amateur to professional, nature, parties, etc. In such context, video content analysis can provide several advantages for classifying content and later search, or to provide additional information about the content itself.

Example applications in different industries addressed by this Generic Enabler are:

- Telecom industry: Identify characteristics in video content recorded by single mobile users; identify communalities in the recordings across several mobile users (e.g., within the same cell).
- Mobile users: (Semi-)automated annotation of recorded video content, point of interest recognition and tourist information in augmented reality scenarios, social services (e.g., facial recognition).
- IT companies: Automated processing of video content in databases.
- Surveillance industry: Automated detection of relevant events (e.g., alarms, etc.).
- Marketing industry: Object/brand recognition and sales information offered (shops near user, similar products, etc.).

14.3.2 Target Usage

The target users of the Compressed Domain Video Analysis GE are all applications that want to extract meaningful information from video content and that need to automatically find characteristics in video data bases on given tasks. The GE can work for previously stored video data as well as for video data streams (e.g., received from a camera in real time).

In the media era of the web, much content is user-generated (UGC) and span over any possible kind, from amateur to professional, nature, parties, etc. In such context, video content analysis can provide several advantages for classifying content and later search, or to provide additional information about the content itself.

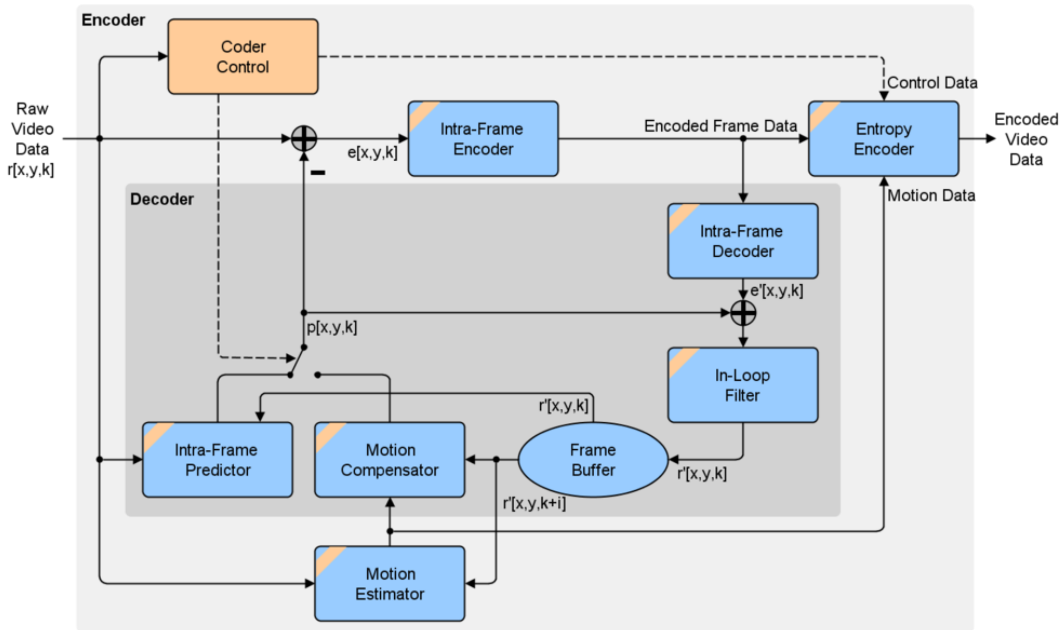
Example applications in different industries addressed by this Generic Enabler are:

- Telecom industry: Identify characteristics in video content recorded by single mobile users; identify communalities in the recordings across several mobile users (e.g., within the same cell).
- Mobile users: (Semi-)automated annotation of recorded video content, point of interest recognition and tourist information in augmented reality scenarios, social services (e.g., facial recognition).
- IT companies: Automated processing of video content in databases.
- Surveillance industry: Automated detection of relevant events (e.g., alarms, etc.).
- Marketing industry: Object/brand recognition and sales information offered (shops near user, similar products, etc.).

14.4 Basic Concepts

14.4.1 Block-Based Hybrid Video Coding

Video coding is always required if a sequence of pictures has to be stored or transferred efficiently. The most common method to compress video content is the so-called block-based hybrid video coding technique. A single frame of the raw video content is divided into several smaller blocks and each block is processed individually. Hybrid means that the encoder as well as the decoder consists of a combination of motion compensation and prediction error coding techniques. A block diagram of a hybrid video coder is depicted in the figure below.



Block diagram of a block-based hybrid video coder

A hybrid video coder can be divided in several generic components:

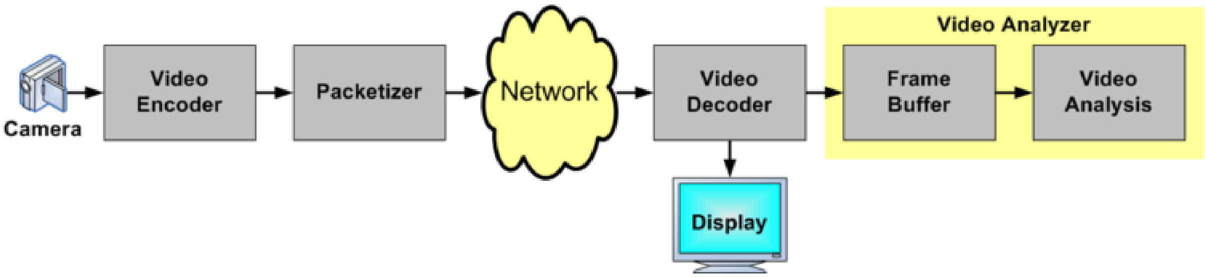
- **Coder Control**: Controls all other components to fulfill pre-defined stream properties, like a certain bit rate or quality. (Indicated by colored block corners)
- **Intra-Frame Encoder**: This component usually performs a transform to the frequency domain, followed by quantization and scaling of the transform coefficients.
- **Intra-Frame Decoder**: To avoid a drift between encoder and decoder, the encoder includes a decoder. Therefore, this component reverses the previous encoding step.
- **In-Loop Filter**: This filter component could be a set of consecutive filters. The most common filter operation here is deblocking.
- **Motion Estimator**: Comparing blocks of the current frame with regions in previous and/or subsequent frames permits modelling the motion between these frames.
- **Motion Compensator**: According to the results of the **Motion Estimator**, this component compensates the estimated motion by creating a predictor for the current block.
- **Intra-Frame Predictor**: If the control decides to use intra-frame coding techniques, this component creates a predictor for the current block by just using neighboring blocks of the current frame.
- **Entropy Encoder**: The during the encoding process gathered information is entropy encoded in this component. Usually, a cost-efficient variable length coding technique (e.g., CAVLC in H.264/AVC) or even an arithmetic coder (e.g., CABAC in H.264/AVC) is used.

During the encoding process, the predicted video data ***p[x,y,k]*** (where ***x*** and ***y*** are the Cartesian coordinates of the ***k***-th sample, i.e., frame) gets subtracted from the raw video data ***r[x,y,k]***. The resulting prediction error signal ***e[x,y,k]*** then gets intra-frame and entropy encoded.

The decoder within the encoder sums up the en- and decoded error signal ***e'[x,y,k]*** and the predicted video data ***p[x,y,k]*** to get the reconstructed video data ***r'[x,y,k]***. These reconstructed frames are stored in the **Frame Buffer**. During the motion compensation process, previous and/or subsequent frames of the current frame (***r'[x,y,k+i]***, $i \in \mathbb{Z} \setminus \{0\}$) are extracted from the buffer.

14.4.2 Compressed Domain Video Analysis

In literature, there are several techniques for different post-processing steps for videos. Most of them operate in the so-called pixel domain. Pixel domain means that any processing is directly performed on the actual pixel values of a video image. Thereto all compressed video data has to be decoded before analysis algorithms can be performed. A simple processing chain of pixel domain approaches is depicted in the figure below.

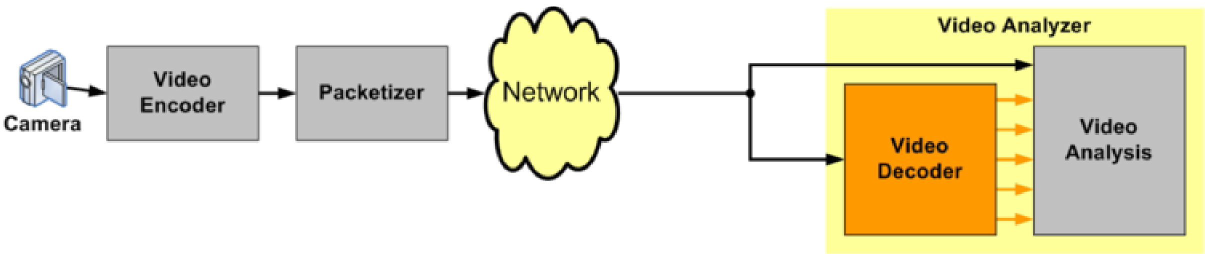


A simple pixel domain processing chain

The simplest way of analyzing video content is to watch it on an appropriate display. For example, a surveillance camera could transmit images of an area that is relevant for security to be evaluated by a watchman. Although this mode obviously finds its application in practice, it is not applicable for all systems, because of two major problems. The first problem is that at any time someone needs to keep track of the monitors. As a result this mode is indeed on the one hand real-time capable, but on the other hand quite expensive. A second major problem is that it is not scalable. If a surveillance system has a huge amount of cameras installed, it is nearly impossible to keep track of all of the monitors at the same time. So the efficiency of this mode will decrease with an increasing number of sources.

Beside a manual analysis of video content, an automated analysis became more and more important in the last years. At first, the received video content from the network has to be decoded. Thereby the decoded video frames are stored in a frame buffer to have access to them during the analysis procedure. Based on these video frames an analysis algorithm, e.g., object detection and tracking can be performed. A main advantage over a manual analysis is that this mode is usually highly scalable and less expensive. But due to the decoding process, the frame buffer operations, and the usually high computing time of pixel domain detection algorithms, this mode is not always real-time capable and has furthermore a high complexity.

Due to the limitations of pixel domain approaches, more and more attempts were made to transfer the video analysis procedures from pixel domain to compressed domain. Working within compressed domain means to work directly on compressed data. The following figure gives an example for a compressed domain processing chain.

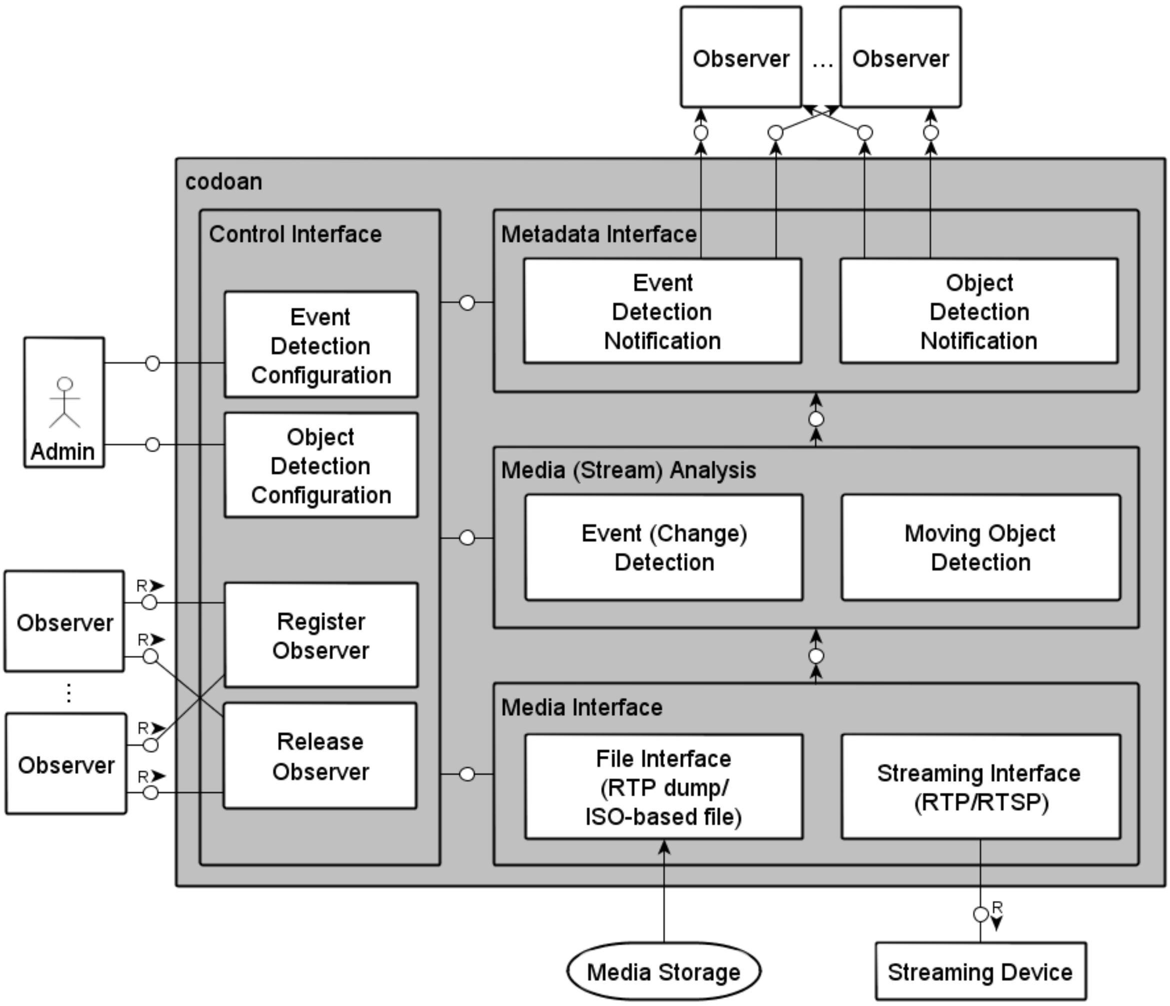


A simple compressed domain processing chain

Due to the omission of the preceding decoder it is now possible to work directly with the received data. At the same time, the now integrated decoder permits to extract single required elements from the data stream and to use them for analyzing. As a result, the analysis becomes less computationally intensive due to the reason that the costly decoding process must not be passed through completely at any time. Furthermore, this solution consumes less resources since it is not required anymore to store the video frames in a buffer. This leads to a technique that is compared to pixel domain techniques usually more efficient and appears more scalable.

14.5 Architecture

The Compressed Domain Video Analysis GE consists of set of tools for analyzing video streams in the compressed domain. Its purpose is to avoid costly video content decoding prior to the actual analysis. Thereby, the tool set processes video streams by analyzing compressed or just partially decoded syntax elements. The main benefit is its very fast analysis due to a hierarchical architecture. The following figure illustrates the functional blocks of the GE. Note that *codoan* is the name of the tool that represents the reference implementation of this GE. Therefore, in some figures one will find the term *codoan* instead of *CDVA GE*.



CDVA GE – Functional description

The four components of the Compressed Domain Video Analysis GE are **Media Interface**, **Media (Stream) Analysis**, **Metadata Interface**, and the **Control Interface**. They are described in detail in the following sub-sections. A realization of a Compressed Domain Video Analysis GE consists of a composition of different types of realizations for the four building blocks (i.e., components). The core functionality of the realization is determined by the selection of the **Media (Stream) Analysis** component (and the related sub-components).

Input and output format are determined by the selection of the inbound and outbound interface component, i.e., **Media Interface** and **Metadata Interface** components. The interfaces are stream-oriented.

14.5.1 Media Interface

The **Media Interface** receives the media data through different formats. Several streams/files can be accessed in parallel (e.g., different RTP sessions can be handled). Two different usage scenarios are regarded:

- **Media Storage**: A multimedia file has already been generated and is stored on a server in a file system or in a database. For analysis, the media file can be accessed independently of the original timing. This means that analysis can happen slower or faster than real-time and random access on the timed media data can be performed. The **File Interface** is able to process the following file types:
    - RTP dump file format used by the RTP Tools, as described in [rtpdump]
    - An ISO-based file format (e.g., MP4), according to ISO/IEC 14496-12 [ISO08], is envisioned
- **Streaming Device**: A video stream is generated by a device (e.g., a video camera) and streamed over a network using dedicated transport protocols (e.g., RTP, DASH). For analysis, the media stream can be accessed only in its original timing, since the stream is generated in real time. The **Streaming Interface** is able to process the following stream types:
    - Real-time Transport Protocol (RTP) packet streams as standardized in RFC 3550 [RFC3550]. Payload formats to describe the contained compression format can be further specified (e.g., RFC 6184 [RFC6184] for the H.264/AVC payload).
    - Media sessions established using RTSP (RFC 2326 [RFC2326])
    - HTTP-based video streams (e.g., REST-like APIs). URLs/URIs could be used to identify the relevant media resources. (envisioned)

Note that according to the scenario (file or stream) the following component either operates in the **Media Analysis** or **Media Stream Analysis** mode. Some sub-components of the **Media (Sream) Analysis** component are codec-independent. Sub-components on a lower abstraction level are able to process H.264/AVC video streams. MPEG-4 is envisioned in addition.

14.5.2 Media (Stream) Analysis

The main component is the **Media (Stream) Analysis** component. The GE operates in the compressed domain, i.e., the video data is analyzed without prior decoding. This allows for low-complexity and therefore resource-efficient processing and analysis of the media stream. The analytics can happen on different semantic layers of the compressed media (e.g., packet layer, symbol layer, etc.). The higher (i.e., more abstract) the layer, the lower the necessary computing power. Some schemes work codec-agnostic (i.e., across a variety of compression/media formats) while other schemes require a specific compression format.

Currently two sub-components are integrated:

- **Event (Change) Detection**
    - Receiving RTP packets and evaluating their size and number per frame leads to a robust detection of global changes
    - Codec-independent
    - No decoding required
    - For more details see [CDA]
- **Moving Object Detection**
    - Analyzing H.264/AVC video streams
    - Evaluating syntax elements leads to a robust detection of moving objects.

In principle, the analytics operations can be done in real time. In practical implementations, this depends on computational resources, the complexity of the algorithm and the quality of the implementation. In general, low complexity implementations are targeted for the realization of this GE. In some more sophisticated realizations of this GE (e.g., crawling through a multimedia database), a larger time span of the stream is needed for analytics. In this case, real-time processing is in principle not possible and also not intended.

14.5.3 Metadata Interface

The **Metadata Interface** should use a metadata format used for subsequent processing. The format could, for instance, be HTTP-based (e.g., REST-like APIs) or XML-based.

The **Media (Stream) Analysis** sub-component either detects events or moving objects. Therefore, the **Metadata Interface** consists of an **Event Detection Notification** and an **Object Detection Notification** component.

14.5.4 Control Interface

The **Control Interface** component is used to access and configure the Compressed Domain Video Analysis GE from outside. It currently consists of methods for the configuration of the analysis modules (**Event Detection Configuration** and **Object Detection Configuration**) and the management of the so-called **Observers**, e.g., other GEs or users. The **Observer** management implements to interface methods:

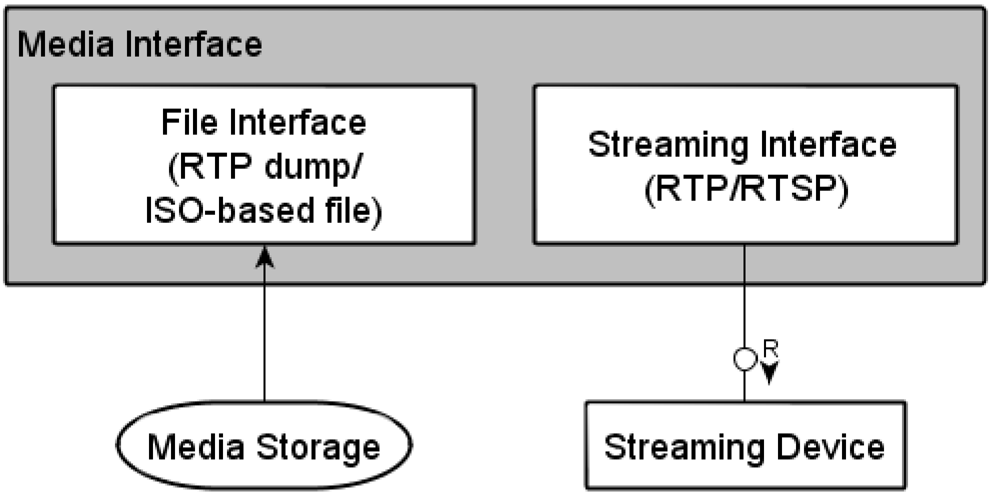
- **Register Observer** : Registered Observers will be notified in case of detections.
- **Release Observer** : Released Observers will not be notified anymore.

14.6 Main Interactions

14.6.1 Interfaces

14.6.1.1 *Media Interface*

The following figure shows the ***Media Interface*** of the GE.



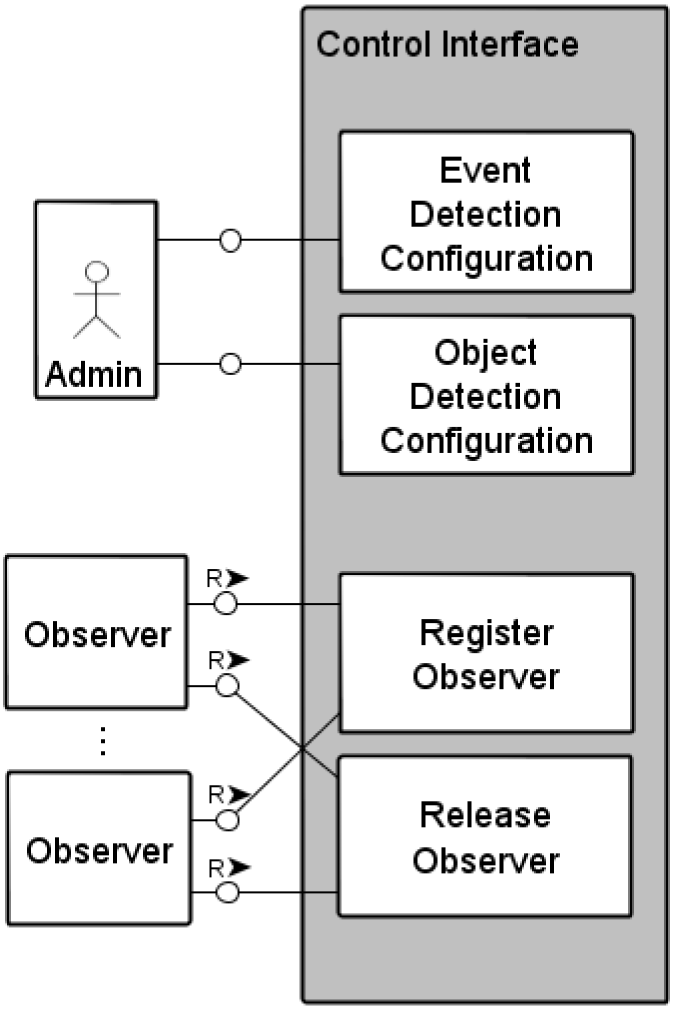
CDVA GE – Media Interface

This interface has no explicitly callable operations. The communications are invoked by the registration of the first observer of a specific source at the ***Control Interface***. Thereby, this component either automatically decides, according to the requested video data, if the data will be fetched from a file or a network stream or the observer indicates from where the data should be requested. If the last observer of a specific source has been released, the file or the stream reception will be closed.

In case of the ***File Interface*** is used, the GE must already have access to the media storage and invokes a simple read function. If the ***Stream Interface*** should fetch the video data, an RTSP session is established by the GE to receive the data by using RTP. Thereto, the GE implements an RTSP client as well as an RTP stack.

14.6.1.2 *Control Interface*

The following figure shows the ***Control Interface*** of the GE.

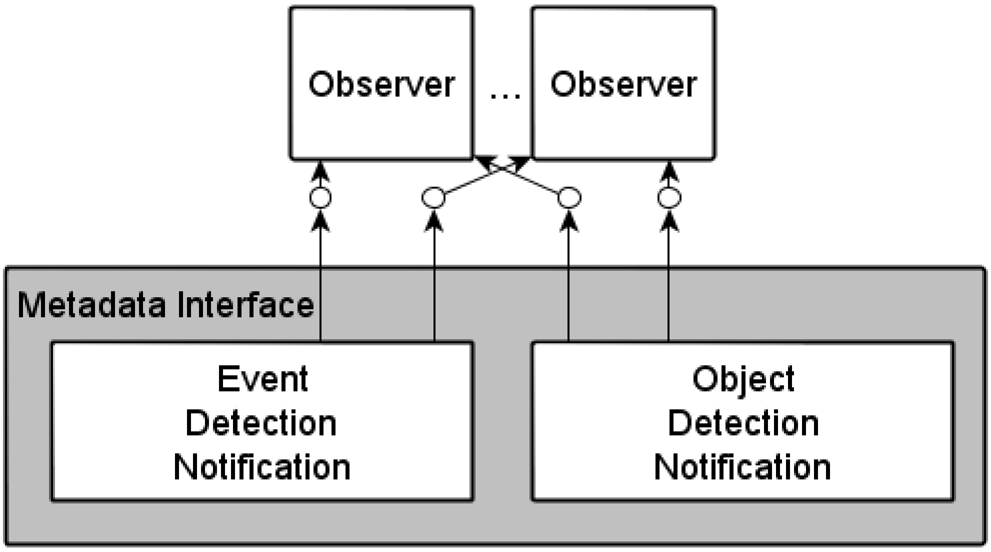


CDVA GE – Control Interface

This interface can either be used by administrators or observers, where administrators are able to configure the GE by calling the ***Configuration*** modules and observers are able to request a registration or a release (***RegisterObserver***, ***ReleaseObserver***).

14.6.1.3 ***Metadata Interface***

The following figure shows the ***Metadata Interface*** of the GE.



CDVA GE – Metadata Interface

This interface notifies observers in case of detections. Thereto, the observers have to register successfully at the ***Control Interface*** first. The used format for the notifications is described in the following section (***EventDetectionNotification***, ***ObjectDetectionNotification***).

14.6.2 Operations

14.6.2.1 ***Configuration***

At the ***Control Interface*** the object and the event detection algorithms of the GE can be configured by the following two operations:

- ***EventDetectionConfiguration***
- ***ObjectDetectionConfiguration***

14.6.2.2 ***Notification***

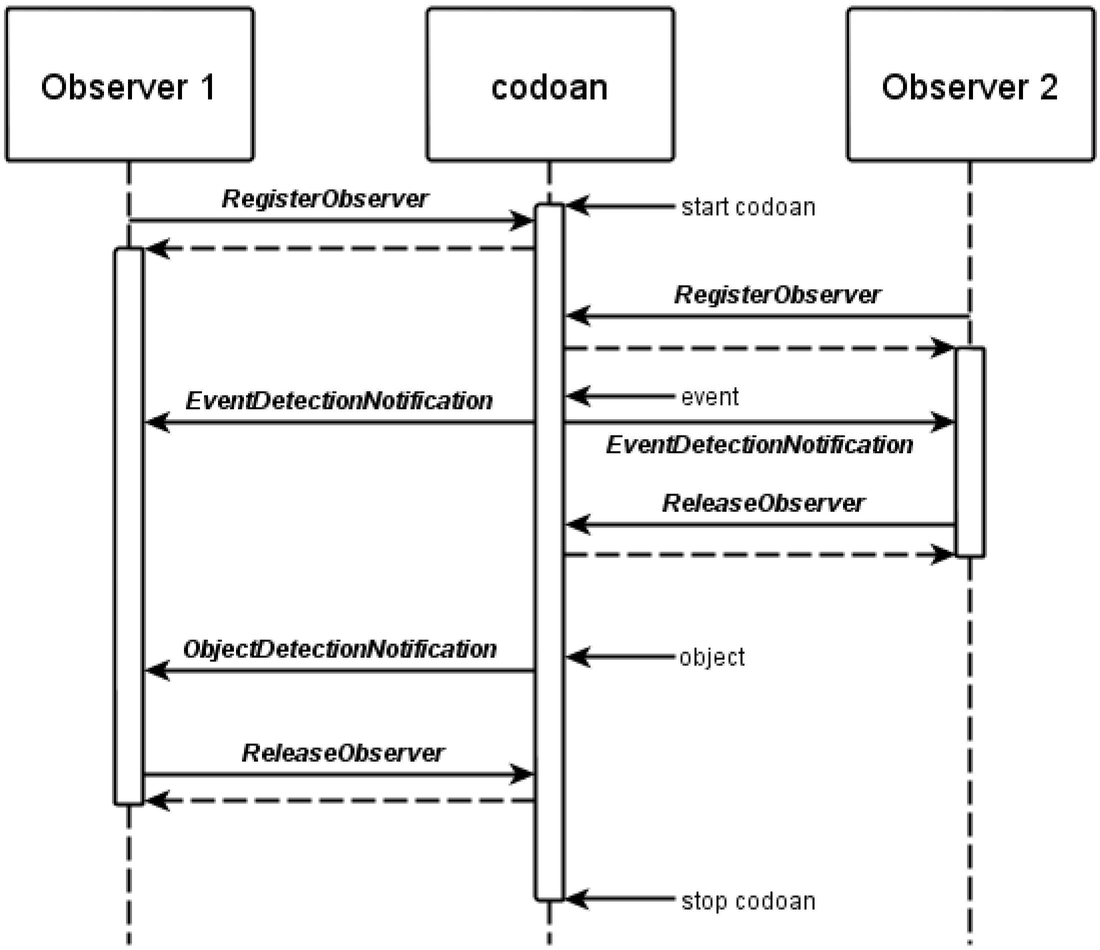
Observers can send requests for registration and release by calling the following methods, respectively:

- ***RegisterObserver***
- ***ReleaseObserver***

Each registered observer will be notified in case of detections by performing the following two operations:

- ***EventDetectionNotification***
- ***ObjectDetectionNotification***

An example scenario of the registration/notification/release process is depicted in the figure below.



CDVA GE – Example scenario

Event and object metadata (the output of the ***Notification*** modules) are encapsulated in an XML-based Scene Description format, according to the ONVIF specifications [ONVIF]. Thereby, the XML root element is called `MetadataStream`. The following code block depicts a brief example to illustrate the XML structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<MetadataStream xmlns="http://www.onvif.org/ver10/schema"
                xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"
                xmlns:tns="http://www.w3.org/2005/08/addressing"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
```

```
xsi:schemaLocation="http://www.onvif.org/ver10/schema
http://www.onvif.org/onvif/ver10/schema/onvif.xsd
                                    http://docs.oasis-
open.org/wsn/b-2 http://docs.oasis-open.org/wsn/b-2.xsd

http://www.w3.org/2005/08/addressing
http://www.w3.org/2005/08/addressing/ws-addr.xsd">
 <Event>
   <wsnt:NotificationMessage>
     <wsnt:SubscriptionReference>
       <tns:Address>anyURI</tns:Address>
     </wsnt:SubscriptionReference>
     <wsnt:Topic Dialect="anyURI" />
     <wsnt:ProducerReference>
       <tns:Address>anyURI</tns:Address>
     </wsnt:ProducerReference>
     <wsnt:Message>
       <self-definedElement>content</self-definedElement>
     </wsnt:Message>
   </wsnt:NotificationMessage>
 </Event>

 <VideoAnalytics>
   <Frame UtcTime="2011-11-22T00:00:00">
     <Object ObjectId="0">
       <Appearance>
         <Shape>
           <BoundingBox    bottom="0.0"    top="0.0"    right="0.0"
left="0.0" />
           <CenterOfGravity x="0.0" y="0.0" />
           <Polygon>
             <Point x="0.0" y="0.0" />
             <Point x="0.0" y="0.0" />
             <Point x="0.0" y="0.0" />
           </Polygon>
         </Shape>
       </Appearance>
     </Object>
   </Frame>
 </VideoAnalytics>
</MetadataStream>
```

Note that not all elements are mandatory to compose a valid XML document according to the corresponding ONVIF XML Schema.

14.7 Basic Design Principles

- Critical product attributes for the Compressed Domain Video Analysis GE are especially high detection/recognition ratios containing only few false positives and low-complexity operation.
- Partitioning to independent functional blocks enables the GE to support a variety of analysis methods on several media types and to get easily extended by new features. Even several operations can be combined.
- Low-complexity algorithms and implementations enable the GE to perform very fast analyses and to be highly scalable.
- GE implementations support performing parallel analyses using different sub-components.

14.8 References

[ISO08]	ISO/IEC 14496-12:2008, Information technology – Coding of audio-visual objects – Part 12: ISO base media file format, Oct. 2008.
[RFC2326]	H. Schulzrinne, A. Rao, and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, Apr. 1998.
[RFC3550]	H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications", RFC 3550, Jul. 2003.
[RFC6184]	Y.-K. Wang, R. Even, T. Kristensen, R. Jesup, "RTP Payload Format for H.264 Video", RFC 6184, May 2011.
[CDA]	M. Laumer, P. Amon, A. Hutter, and A. Kaup, "A Compressed Domain Change Detection Algorithm for RTP Streams in Video Surveillance Applications", MMSP 2011, Oct. 2011.
[ONVIF]	ONVIF Specifications
[rtpdump]	rtpdump format specified by RTP Tools

15 FIWARE ArchitectureDescription Data QueryBroker

You can find the content of this chapter as well in the wiki of fi-ware.

15.1.1 Copyright

15.1.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

15.2 Overview

15.2.1 Introduction to the Media-enhanced Query Broker GE

Multimedia data is produced at an immense rate. By investigating solutions and approaches for storing and archiving the produced data, one rapidly ends up in a highly heterogeneous environment of data stores. Usually, the involved domains feature individual sets of metadata formats for describing content, technical or structural information of multimedia data [Stegmaier 09a]. Furthermore, depending on the management and retrieval requirements, these data sets are accessible in different systems supporting a multiple set of retrieval models and query languages. By summing up all these obstacles, easy and efficient access and retrieval across those system borders is a very cumbersome task [Smith 08]. Standards are one way to introduce interoperability among different peers. Recent developments and achievements in the domain of multimedia retrieval concentrated on the establishment of a multimedia query language (MPEG) [Döller 08a], standardized image retrieval (JPEG) and the heterogeneity problem between metadata formats (JPEG) [Döller 10]. Another approach for interoperable media retrieval is the introduction of a mediator or middleware system abstracting the communication: a Media-enhanced Query Broker. Acting as middleware and mediator between multimedia clients and retrieval systems, collaboration can be remarkably improved. A Media-enhanced Query Broker accepts complex multi-part and multimodal queries from one or more clients and maps/distributes those to multiple connected Multimedia Retrieval Systems (MMRS). Consequently, implementation complexity is reduced at the client side as only one communication partner needs to be addressed. Result aggregation and query distribution is also accommodated, further easing client development. However, the actual retrieval process of the multimedia data is performed inside the connected data stores.

15.2.2 Target usage

The Media-enhanced Query Broker GE provides an intelligent, abstracting interface for retrieval of data from the FI-WARE data management layer. This is provided in addition to the publish/subscribe interface as another modality for accessing data.

Principal users of the Media-enhanced Query Broker GE include applications that require a selective, on-demand view on the content/context data in the FI-WARE data management platform via a single, unified API, without taking care about the specifics of the internal data storage and DB implementations and interfaces.

Therefore, this GE provides support for integration of query-functions into the users' applications by abstracting the access to databases and search engines available in the FI-WARE data management platform while also offering the option to simultaneously access outside data sources. At the same time its API offers an abstraction from the distributed and heterogeneous nature of the underlying storage, retrieval and DB / metadata schema implementations.

The Media-enhanced Query Broker GE provides support for highly regular ("structured") data such as the one used in relational databases and queried by SQL like languages. On the other hand it also supports less regular "semi-structured" data, which are quite common in the XML tree-structured world and can be accessed by the XQuery language. Another data structure supported by the Media-enhanced Query Broker is RDF as a well structured graph-based data model that is queried using the SPARQL language. In addition, the Media-enhanced Query Broker GE provides support for specific search and query functions required in (metadata based) multimedia content search (e.g., image similarity search using feature descriptors).

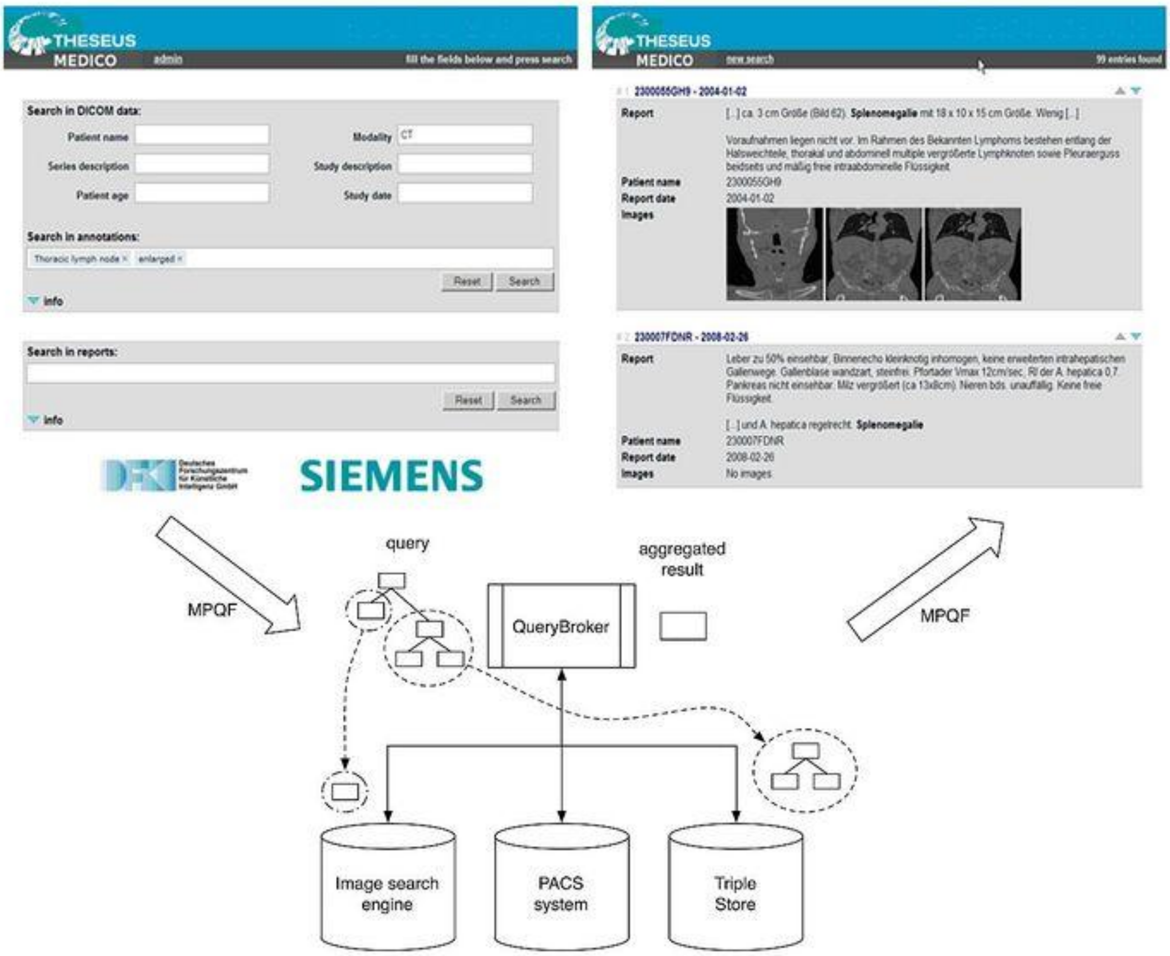
The question about how non-relational or "NoSQL" databases, which are becoming an increasingly important part of the database landscape, can be integrated, is one of the open points to be addressed during the FI-WARE project.

The underlying approach for the extension of the Media-enhanced Query Broker GE is to try identifying families of (abstract) query languages (based on minimum common denominators of existing query languages) together with preferred representatives allowing to categorize the capabilities of the data resources in respect to what and how they can be queried.

15.2.3 Example Scenario

The already identified issues of heterogeneity can be also found in the current diagnostic process at hospitals. The workflow of a medical diagnosis is mainly based on reviewing and comparing images coming from multiple time points and modalities in order to monitor disease progression over a certain period of time. For ambiguous cases the radiologist deeply relies on reference literature or second opinion. Beside textual data stored in appraisals, a vast amount of images (e.g., CT scans) is stored in Picturing Archive and Communications Systems (PACS), which could be reused for decision support. Unfortunately efficient access to this information is not available due to weak search capabilities.

The mission of the MEDICO application scenario is to establish an intelligent and scalable search engine for the medical domain by combining medical image processing and semantically rich image annotation vocabularies.



Search infrastructure: end-to-end workflow in MEDICO

The figure above sketches an end-to-end workflow inside the MEDICO system. It provides the user with an easy-to-use web-based form to describe the desired search query. Currently, this user interface utilizes a semantically rich data set composed of DICOM tags, image annotations, text annotations and gray-value based (3D) CT images. This leads to a heterogeneous multimedia retrieval environment with multiple query languages: DICOM tags as well as the raw image data are stored in a PACS, annotations describing images, doctor´s letter as well as laboratory exminations are saved in a triple store. Finally, a similarity search can be conducted by the use of an image search engine, which operates on top of extracted image features. Apparently, all these retrieval services are using their own query languages for retrieval (e.g., SPARQL) as well as the actual data representation for annotation storage (e.g., RDF/OWL). To fulfill a sophisticated semantic search, the present interoperability issues have to be solved. Furthermore, it is essential to enable federated search functionalities in this environment. These requirements have been taken into account in the design and implementation of the QueryBroker. An overview of the architecture can be found in [Stegmaier 10] and [Stegmaier 09b].

15.3 Basic Concepts

The QueryBroker is implemented as middleware to establish unified retrieval in distributed and heterogeneous environments with extension functionalities to integrate multimedia specific retrieval paradigms in the overall query execution plan, e.g., multimedia fusion techniques.

15.3.1 Design Principles

To ensure interoperability between the query applications and the registered database services, the Media-enhanced Query Broker is based on the following internal design principles:

- Query language abstraction:

  The Media-enhanced Query Broker is capable of federating an arbitrary amount of retrieval services utilizing various query languages/APIs (e.g., XQuery, SQL or SPARQL). This is achieved by converting all incoming queries into an internal abstract format that is finally translated into the respective specific query languages/APIs of a data store. As an internal abstraction layer, the Media-enhanced Query Broker makes use of the MPEG Query Format (MPQF) [Smith 08], which supports most of the functions in traditional query languages as well as several types of multimedia specific queries (e.g., temporal, spatial, or query-by-example).

- Multiple retrieval paradigms:

  Retrieval systems do not always follow the same data retrieval paradigms. Here, a broad variety exists, e.g. relational, No-SQL or XML-based storage or triple stores. The Media-enhanced Query Broker attempts to shield the applications/users from this variety. Further, it is most likely in such systems, that more than one data store has to be accessed for query evaluation. In this case, the query has to be segmented and distributed to applicable retrieval services. Following this, the Media-enhanced Query Broker acts as a federated database management system.

- Metadata format interoperability:

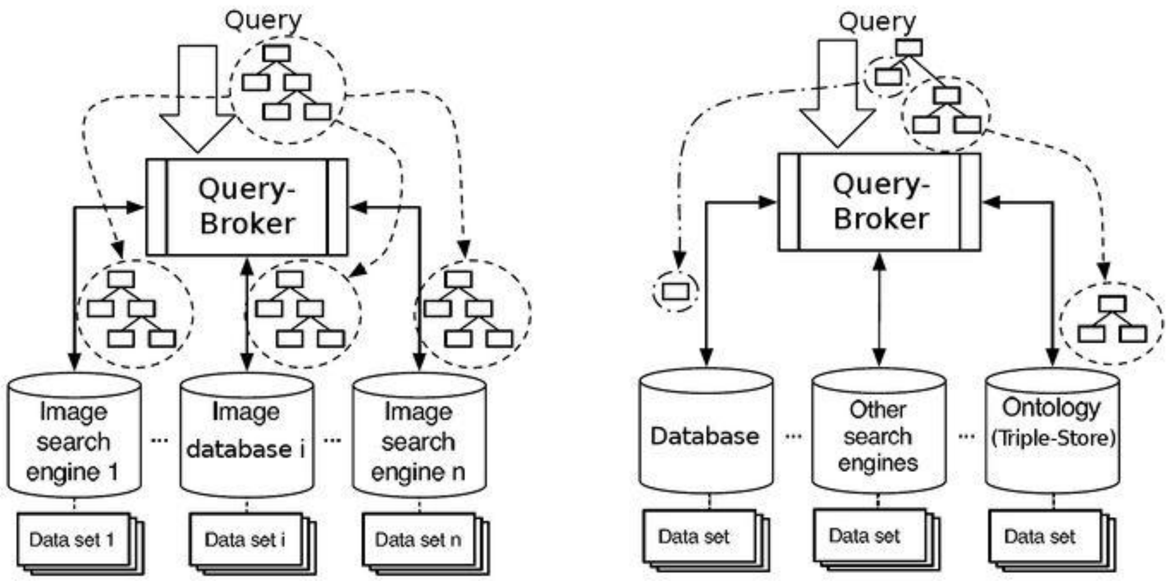
  For an efficient retrieval process, metadata formats are applied to describe syntactic or semantic attributes of (media) resources. There currently exist a huge number of standardized or proprietary metadata formats covering nearly every use case and domain. Thus more than one metadata format are in use in a heterogeneous retrieval scenario. The Media-enhanced Query Broker therefore provides functionalities to perform the transformation between diverse metadata formats where a defined mapping exists and is made available.

- Modular architectural design:

  A modular architectural design should always be striven for in software development. The central aspects in these topics are convertibility, extensibility and reusability. These ensure that the components are loosely coupled in the overall system supporting an easy extension of the provided functionality of components, or even the replacement of these by new implementations.

15.3.2 Query Processing Strategies

The Media-enhanced Query Broker is a middleware component that can be operated in different facets within a distributed and heterogeneous search and retrieval framework including multimedia retrieval systems. In general, the tasks of each internal component of the Media-enhanced Query Broker depend on the registered databases and on the use cases. In this context, two main query-processing strategies are supported, as illustrated in the following figure.



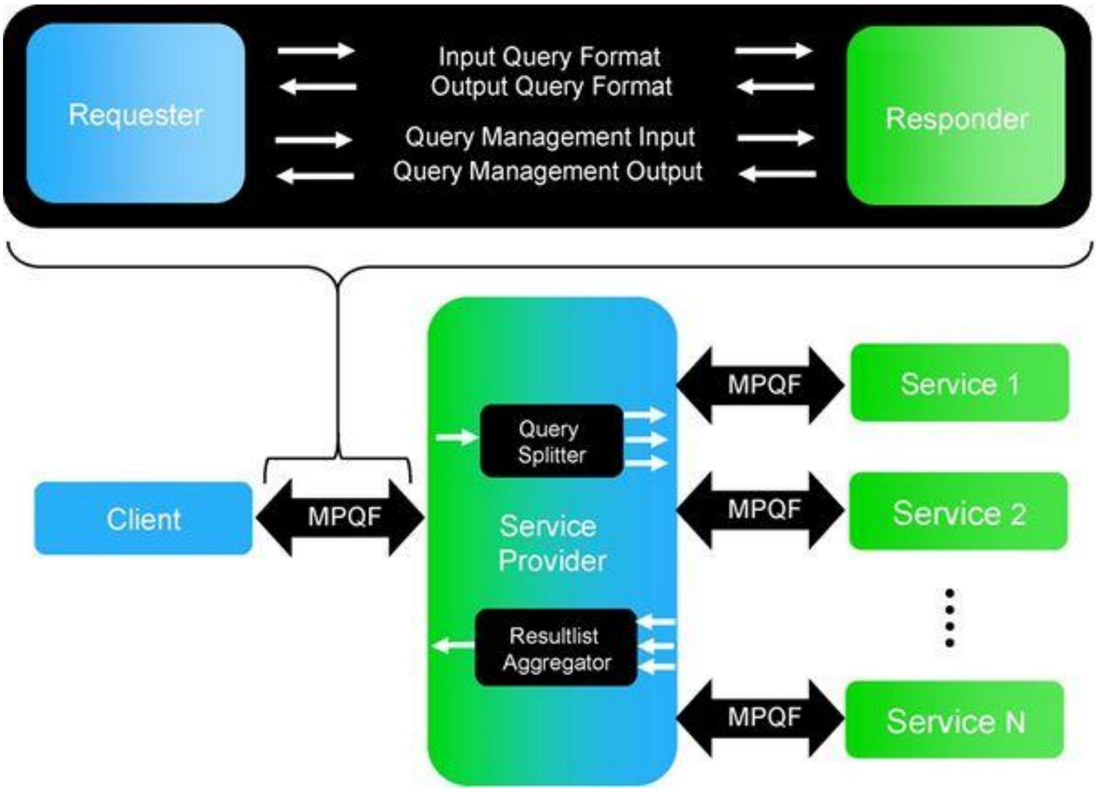
(a) Local/autonomous processing (b) Distributed processing
Query processing strategies

The first paradigm deals with registered and participating retrieval systems that are able to process the whole query locally, see the left side (a) of the figure above. In this sense, those heterogeneous systems may provide their local metadata format and a local / autonomous data set. A query transmitted to such systems can be completely evaluated by the data store and the items of the result set are the outcome of an execution of the query. In case of differing metadata formats in the data stores a transformation of the metadata format is needed before the (sub-) query is transmitted. In addition, depending on the degree of overlap among the data sets, the individual result sets may contain duplicates. However, the most central task for the Media-enhanced Query Broker is the result aggregation process that performs an overall ranking of the partial results. Here, duplicate elimination algorithms may be applied as well.

The second paradigm deals with registered and participating retrieval systems that allow distributed processing on the basis of a global data set as illustrated in the right side (b) of the figure above. The involved heterogeneous systems may depend on different data representation (e.g., ontology based semantic annotations and XML-based feature values) and query interfaces (e.g., SPARQL and XQuery) but describe a common (linked) global data set. In this context, a query transmitted to the Media-enhanced Query Broker needs to be evaluated and optimized resulting in a specific query execution plan. Segments of the query are forwarded to the respective engines to be executed in parallel. Subsequently, the result aggregation has to deal with the correct consolidation and (if required) format conversion of the partial result sets. In this context, the Media-enhanced Query Broker behaves like a federated Database Management System.

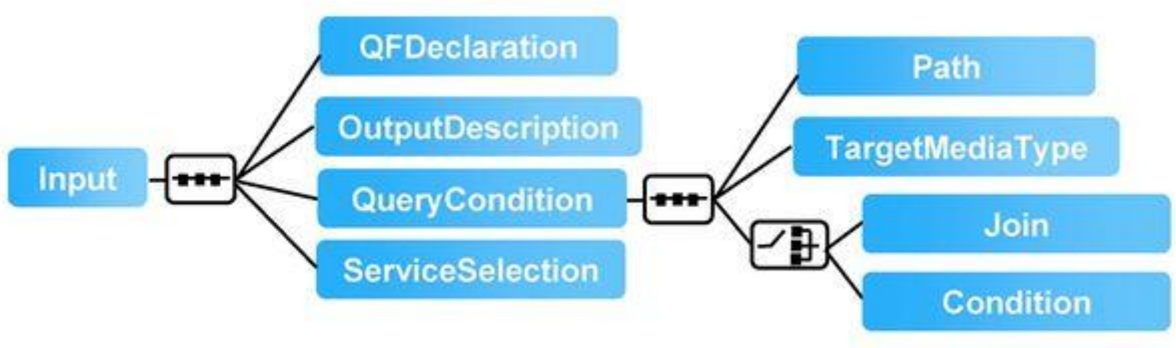
15.3.3 MPEG Query Format (MPQF)

Before discussing the design and the implementation of the Media-enhanced Query Broker in more detail, the main features of MPQF will be introduced. MPQF became an international standard in early 2009 as part 12 of the MPEG-7 standard [MPEG-7]. The main intention of MPQF is to formulate queries in order to address and retrieve multimedia data, like audio, images, video, text or a combination of these. At its core, MPQF is a XML based query language and intended to be used in a distributed multimedia retrieval services (MMRS). Beside the standardization of the query language, MPQF specifies the service discovery and the service capability description. Here, a service is a particular system offering search and retrieval abilities (e.g. image retrieval).



Possible scenario for the use of MPQF

The figure above shows a possible retrieval scenario in a MMRS. The Input Query Format (IQF) provides means for describing query requests from a client to a MMRS. The Output Query Format (OQF) specifies a message container for MMRS responses and finally the Query Management Tools (QMT) offer functionalities such as service discovery, service aggregation and service capability description.



Structure of the Input Query Format

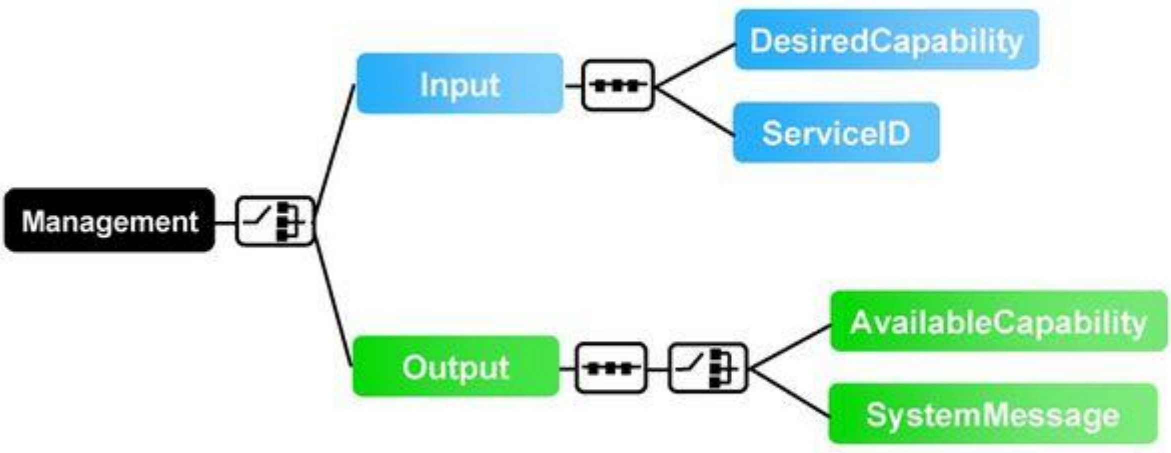
In detail, the IQF (see the figure above) can be composed of three different parts. The first is a declaration part pointing to resources (e.g., image file or its metadata description, etc.) that are used within the query condition or output description part. The output description part allows, by using the respective MMRS metadata description, the definition of the structure as well as the content of the expected result set. Finally, the query condition part denotes the search criteria by providing a set of different query types (see the table below) and expressions (e.g., GreaterThan), which can be combined by Boolean operators (e.g., AND). In order to respond to MPQF query requests, the OQF provides the ResultItem element and attributes signalling paging and expiration dates.

Query type	***Description/Functionality***
QueryByMedia	Similarity or exact search using query by example (using multimedia data)
QueryByDescription	Similarity or exact search using XML based metadata (like MPEG-7)
QueryByFeatureRange	Range retrieval for e.g., low level features like color
QueryByFreeText	Free text retrieval

SpatialQuery	Retrieval of spatial elements within media objects
TemporalQuery	Retrieval of temporal elements within media objects (e.g., a scene in a video)
QueryByXQuery	Container for limited XQuery expressions
QueryByRelevanceFeedback	Retrieval that takes result items of a previous search into accountFree text retrieval
QueryByROI	Retrieval based on a certain region of interest.

Available MPQF query types

Semantic expressions and the QueryBySPARQL query type ensure the retrieval on semantic annotations stored in ontologies possibly defined by RDF/OWL.



Structure of the Query Management Tools

The QMT of MPQF cope with the task of searching for and choosing desired multimedia services for retrieval. This includes service discovery, querying for service capabilities and service capability descriptions. The figure above depicts the element hierarchy of the management tools in MPQF. The management part of the query format consists of either the Input or Output element depending on the direction of the communication (request or response). The MPEG Query Format has been explicitly designed for its use in a distributed heterogeneous retrieval scenario. Therefore, the standard is open for any XML based metadata description format (e.g., MPEG-7 [Matinez 02] or Dublin Core [DublinCore]) and supports, as already mentioned, service discovery functionalities. First approaches in this direction have been realized by [Gruhne 08] and [Döller 08b] which address the retrieval in a multimodal scenario and introduce a MPQF aware Web-Service based middleware. Besides, MPQF adds support for asynchronous search requests as well. In contrast to a synchronous request (the result is allocated as fast as possible) in an asynchronous scenario the user is able to define a time period after when the result will be caught. Such a retrieval paradigm might be of interest for e.g. users of mobile devices with limited hardware/software capabilities. The results of requests (triggered by the mobile device) like “Show me some videos containing information about the castle visible on the example image that has been taken with the digital camera” can then be gathered and viewed at a later point in time from a different location (e.g., the home office) and a different device (e.g., a PC).

15.3.4 Federated Query Evaluation Workflow

As already mentioned, the Media-enhanced Query Broker is not only a routing service for queries to specific data stores, but it is capable of managing federated query execution, too. Thereby Media-enhanced Query Broker transforms incoming user queries (of different formats) to a common internal representation for further processing and distribution to registered data resources and aggregates the returned results before delivering it to the client. In particular it runs through the following central phases:

- Query analysis

  The first step after receiving a query is to register it in the Media-enhanced Query Broker. During registration, the query will be analysed and an according query-tree will be generated. Each sub-query comprising a single query type will become a leaf node. Using the information from the data store registration (cf. "KnowledgeManager" in chapter QueryBroker Architecture) a set of data stores is identified that are able to evaluate certain parts of the incoming query.

- Query segmentation

  The next step is to conduct the actual segmentation of the query based on the already created query-tree. Here, the query will be divided in semantically correct sub-queries, which are again valid MPQF queries but with different semantics. The segmentation has a direct coherence to the set of identified data stores.

- Generation of a query execution plan

  In order to ensure an efficient retrieval, the incoming query (or the generated segments) is transferred into a graph tree structure (directed acyclic graph). After this initial transfer, various techniques for optimization will be applied. The current implementation is able to perform the following optimizations: Early selection push down, move/combination and decamping selections as well as projections, insertion of projections into query execution, join ordering on the basis of selectivity and finally pipelining. Further statistics of the query cache component are used to create an efficient query execution plan on the basis of physical information. Further, it enables the injection of equal (or similar) partial results directly in the query execution planning process.

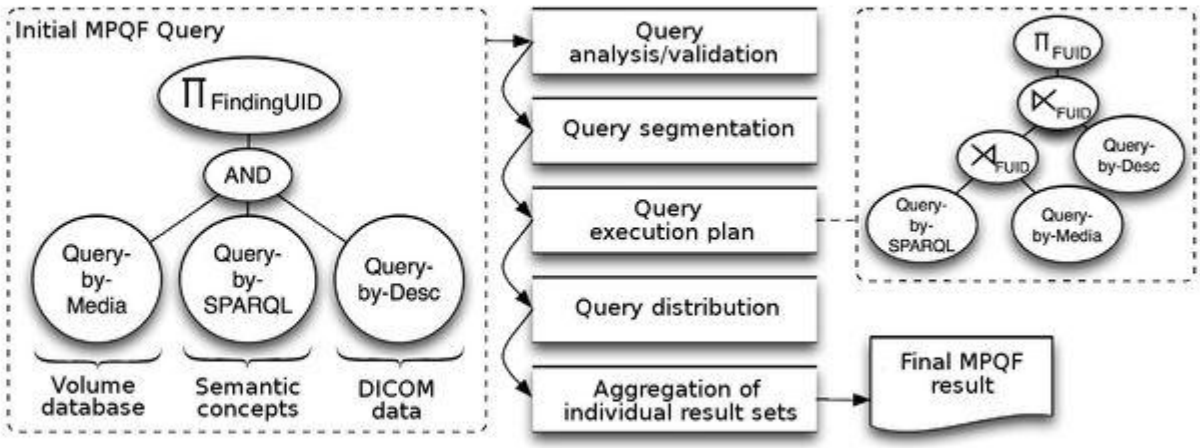
- Distribution of query

  The query or its segments will be distributed in parallel to the appropriate data stores. After retrieval, the partial result sets will be collected.

- Consolidation of partial results

  The partial result sets will be aggregated with respect to the overall query semantics. For this the query-tree is processed backwards from the leaves to the root in a "breadth-first" manner. In the case where the corresponding parent node defines an AND the partial results are joined with the help of a corresponding established semantic link ("join attribute" - see also Creating a Semantic Link) whereas a union operation is carried out if the parent node presents an OR. Unary operators (cf. Querying) are processed directly on the intermediate result.

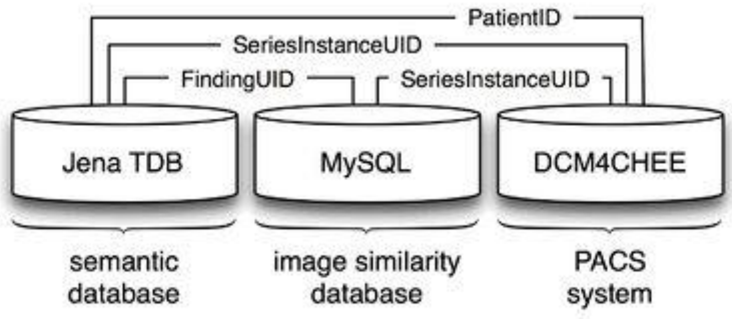
The described workflow of the federated query processing can best be illustrated using the example scenario, as depicted in the figure below.



Central steps of the query execution plan

The federation process always needs a global data set or at least knowledge about the interlinking of the data stores in order to perform an aggregation of the partial results. This interlinking is a way to enable a non-invasive integration of the data stores at the mediator.

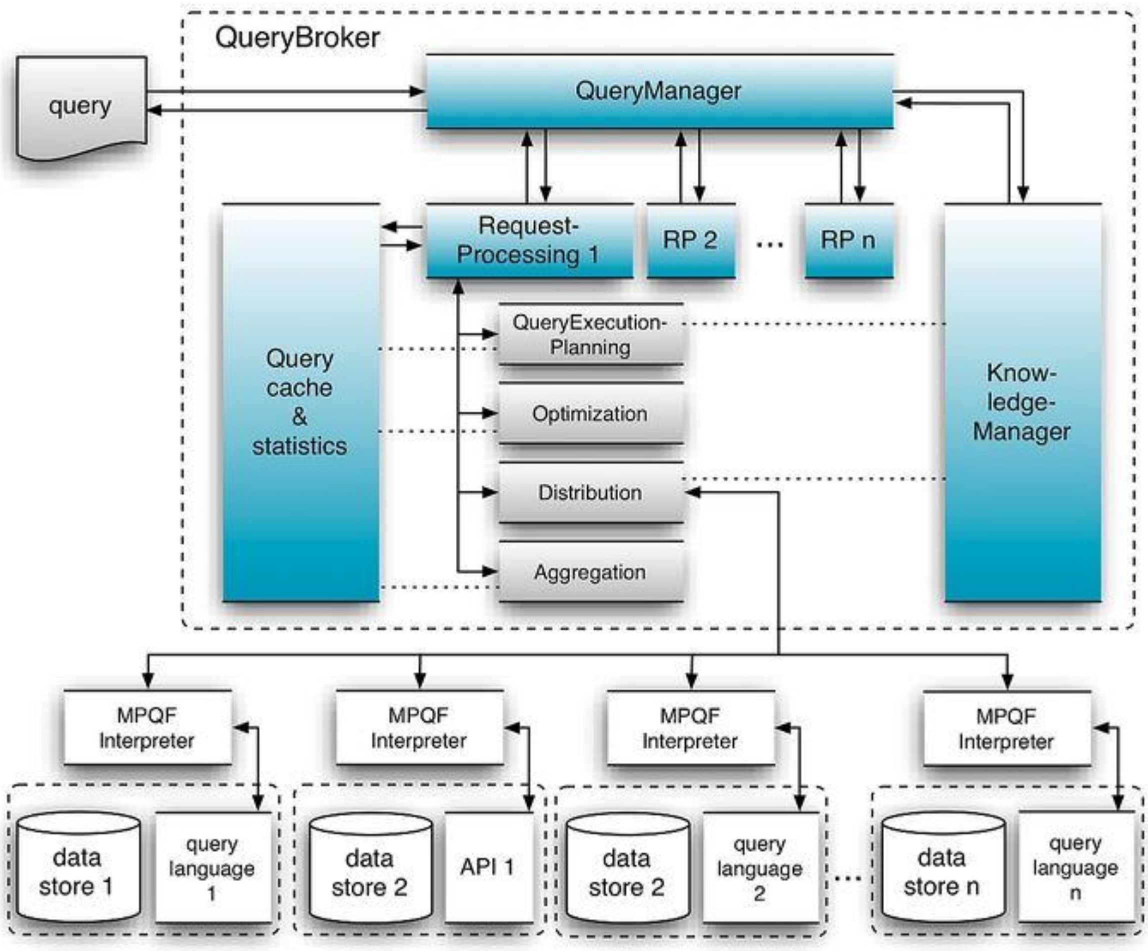
This principle is called semantic links, for a definition and examples see Creating a Semantic Link. The following figure depicts for the example scenario the diverse data sources forming a common (semantically linked) global data set.



Semantic Link between knowledge bases

15.4 QueryBroker Architecture

Knowing the principle processing steps an end-to-end workflow scenario in a distributed retrieval scenario can be sketched, also revealing the architecture. The following figure illustrates the global workflow starting from incoming user queries to returning the aggregated results to the client. It is possible to handle synchronous as well as asynchronous queries. In the following, the subcomponents of a reference implementation of the QueryBroker, based on internal usage of the MPEG Query Format (MPQF), are briefly described. This discussion will be continued in below with a focus on the actual implementation.



Architecture of the QueryBroker

- **QueryManager:**

  The QueryManager is the entry point of every user request. Its main purpose is the receiving of an incoming query as well as API assisted MPQF query generation and validation of MPQF queries. In case an application is not aware in formulating MPQF queries, these can be build by consecutive API calls. Following this, two main parts of the MPQF structure will be created: First, the QueryCondition element holds the filter criteria in an arbitrary complex condition tree. Second, the OutputDescription element defines the structure of the result set. In this object, the needed information about required result items, grouping or sorting is stored. After finalizing the query creation step, the generated MPQF query will be registered at the QueryBroker using the query cache & statistics component In case an instance of a query is created at the client side in MPQF format then this query will be directly registered at the QueryBroker. After a query has been validated, the QueryManager acts as a routing service. It forwards the query to its destination, namely the KnowledgeManager or the RequestProcessing component.

- **KnowledgeManager:**

  The main functionalities of the KnowledgeManager are the (de-) registration of data stores with their capability descriptions and the service discovery as an input for the distribution of (sub-) queries. These capability descriptions are standardized in MPQF, allowing the specification of the retrieval characteristics of registered data stores. These characteristics consider for instance the supported query types or the metadata formats. Subsequently, depending on those capabilities, this component is able to filter registered data stores during the search process (service discovery). For a registered retrieval system, it is very likely that not all functions specified in the incoming queries are supported. In such an environment, one of the important tasks for a client is to identify the data stores, which provide the desired query functions or support the desired result representation formats identified by e.g. an MIME type using the service discovery.

- **RequestProcessing:**

  For each query a single RequestProcessing component will be initialized. This ensures parallelism as well as guaranteeing that a single object manages the complete life cycle of a query. The main tasks of this component are query execution planning, optimization of the chosen query execution plan, distribution of a query and result aggregation, as already discussed above. Besides managing the different states of a query, this component sends a copy of the optimized query to the query cache and statistics component, which collects information in order to improve optimization. Regarding the lifetime of a query, the following states have been defined: pending (query registered, process not started), retrieval (search started, some results missing), processing (all results available, aggregation in progress), finished (result can be fetched) and closed (result fetched or query lifetime expired). These states are also valid for the individual query segments, since they are also valid MPQF queries.

- **Query cache and statistics:**

  The query cache and statistics organizes the registration of queries in the query cache. It collects information about data stores, such as execution times, network statistics, etc. Besides, the data store statistics, the complete query will be stored as well as the partial result sets. The information provided by this component will be used for two different optimization tasks, namely: internal query and query stream optimization. Internal query optimization is a technique following well-known optimization rules of the relational algebra (e.g., operator reordering on the basis of heuristics / statistics). In contrast to that, query stream optimization is intended to detect similar / equal query segments that have already been evaluated. If such a segment has been detected, the results can be directly injected into the query execution plan. Obviously, the query cache will also implement the paging functionality.

- **MPQF interpreter:**

  MPQF interpreters act as a mediator between the QueryBroker and a particular retrieval service. An interpreter receives an MPQF formatted query and transforms it into native calls of the underlying query language of the backend database or search engine system. In this context, several interpreters (mappers) for heterogeneous data stores have been implemented (e.g., Flickr, XQuery, etc.). Furthermore, an interpreter for object- or relational data stores is envisaged. After a successful retrieval, the Interpreter converts the result set in a valid MPQF formatted response and forwards it to the QueryBroker.

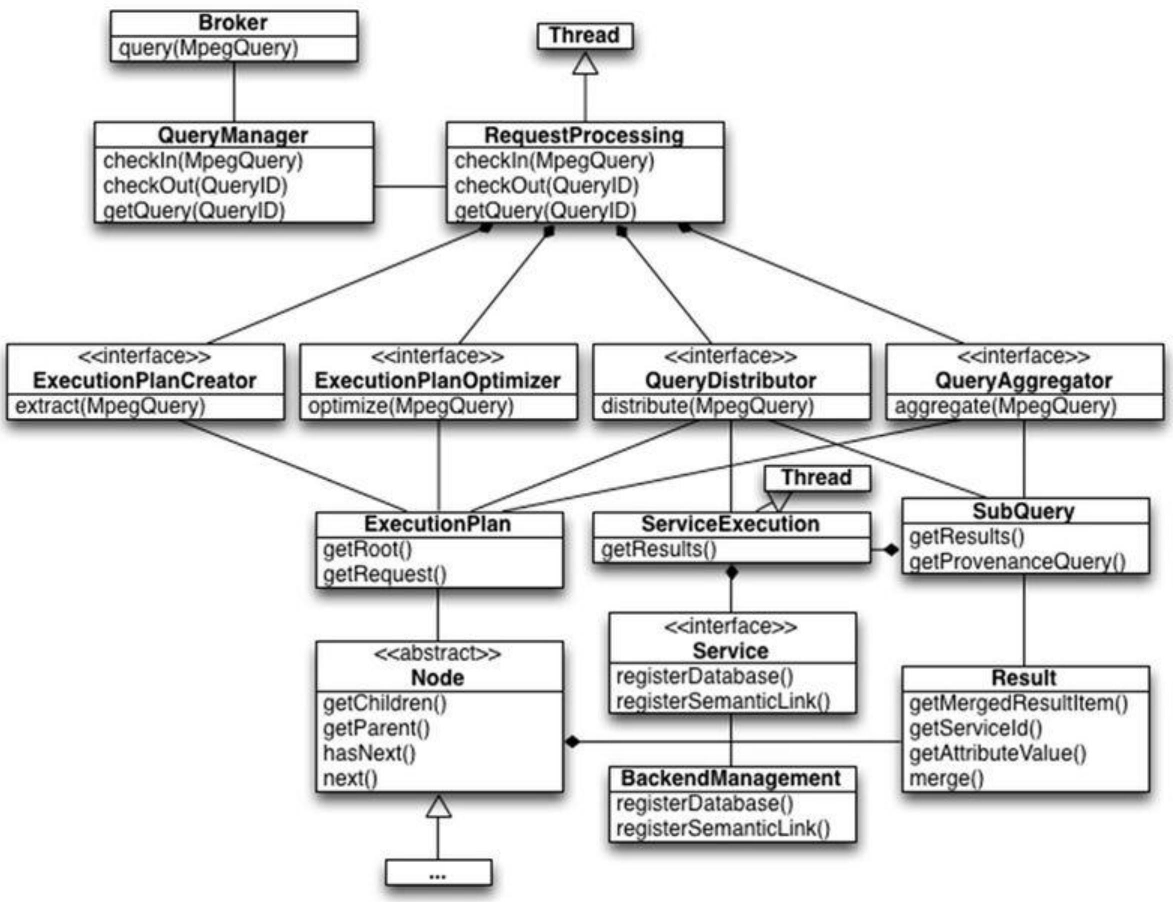
15.5 Main Interactions

15.5.1 Modules and Interfaces

This section covers the description of the software modules and interfaces of the QueryBroker. First, the overall architecture will be highlighted, followed by the actual backend and frontend functionalities. The section below shows an example implementation of all required steps to initialize the QueryBroker. The implementation at its core is based on the Spring Framework (e.g., enabling extensibility and inversion of control) and MAVEN (e.g., quality assurance and build management).

15.5.2 Architecture

The following figure shows an overview over the QueryBroker software architecture. Only the key elements are listed below, to give a short briefing how the elements are related.



QueryBroker software architecture

- **BackendManagement** provides the functionality to register and remove service endpoints. (See Chapter Backend Functionality for more information)

- **Service** interface has to be implemented by any service endpoint. A service endpoint connects a database or another dataset to the multimedia query framework.
- **Broker** represents the central access point to the federated query framework. It provides the functionality to query distributed and heterogeneous multimedia databases using MPQF as query format. The main task is to receive MPQF-queries and control the following request processing (synchronous / asynchronous mode of operation or result fetching). See the section on Frontend Functionality for more information.
- **QueryManager** handles all received and active queries. New queries can be checked-in and corresponding result sets can be checked-out by the Broker.
- **RequestProcessing** controls single query processing in a parallelized way. First an execution plan for the received query is created, followed by an optimization of the plan. Afterwards the query distribution and aggregation of the resulting sub-queries are performed. The implementations of the 4 parts are injected via the Spring framework and can be modified easily by XML configuration.
- **ExecutionPlanCreator** transforms the received MPQF query tree into an internal execution plan tree structure.
- **ExecutionPlanOptimizer** optimizes the default execution plan by replacing or switching the original tree nodes. The tree can be also transformed into a directed acyclic graph (DAG), to avoid isomorphic sub-trees in the execution plan.
- **QueryDistributor** has to analyse which sub-trees of the execution plan have to be distributed. Sub-queries can consist of one or many distributed queries to service endpoints. Each distributed query gets encapsulated in a Service Execution.
- **ServiceExecution** is a wrapper for a parallel execution of a service endpoint to utilize multicore processors.
- **QueryAggregator** gets the sub-queries including the results from the service endpoints and the query execution plan. The aggregator can combine theses two elements and process the queried results.

15.5.3 Backend Functionality

Before queries can be sent to the QueryBroker, the backend management has to be set up. All backend functionalities are reachable through the BackendManagement singleton (de.uop.dimis.air.backend.BackendManagement). Here, services endpoints can be (de-) registered and semantic links between them created. A service endpoint provides the functionality to connect a database or dataset to the multimedia query framework. A semantic link is meant to define the atomic connection between two heterogeneous and distributed knowledge bases on the basis of semantically equal properties. The semantic links can be set by XPath expressions.

15.5.3.1 *(De-)Register a Service*

Executable by BackendManagement.getInstance().registerDatabase(mqt); Service endpoints are able to execute sub trees of the query execution plan. At the moment only single leaves are supported as sub trees. These can be Query-By-Media or Query-by-Description. To register a service endpoint, which has to implement the Service Interface (de.uop.dimis.air.backend.Service), a valid MPQF message needs to be formulated like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<mpqf:MpegQuery mpqfID="" xmlns:mpqf="urn:mpeg:mpqf:schema:2008"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mpeg:mpqf:schema:2008
mpqf_semantic_enhancement.xsd">
	<mpqf:Management>
		<mpqf:Input>
			<mpqf:DesiredCapability>
				<!-- Query By Media: 100.3.6.1(Standard
Annex B.2) -->
				<mpqf:SupportedQueryTypes
href="urn:mpeg:mpqf:2008:CS:full:100.3.6.1" />
			</mpqf:DesiredCapability>
			<mpqf:ServiceID>
				de.uop.dimis.air.ExampleService
			</mpqf:ServiceID>
		</mpqf:Input>
	</mpqf:Management>
</mpqf:MpegQuery>
```

This contains the ServiceID, which is equal to the qualified name of the implementation class. The DesiredCapabilities declare which query types the service can handle. In this example the ExampleService can handle Query-By-Media. See the MPQF-Standard Annex B.2 for more Query URNs. In order to deregister a service endpoint a MPQF-Register-Message must be sent with an empty list of desired capabilities.

Java example:

```
// get the register query xml file as stream
InputStream stream = ...
// unmarshal the xml file
Unmarshaller u = NamespaceHelper.getInstance().getJAXBContextMpqfJPSearch().createUnmarshaller();
MpegQueryType mpqfQuery = (MpegQueryType) u.unmarshal(stream);
// register the database
BackendManagement.getInstance().registerDatabase(mpqfQuery);
```

15.5.3.2 *Creating a Semantic Link*

Executable by BackendManagement.getInstance().registerSemanticLink(sl); To be able to merge results from different services it is necessary to know which fields can be used for identification (cp. Primary key in relational database systems). For every pair of services a semantic link can be defined. If such a link is undefined, a default semantic link will be created at runtime. The default semantic link uses the *identifier* field of the JPSearch Core Meta Schema for every Service. KeyMatchesType-Messages are used for the registration of a semantic link:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<key:KeyMatches xmlns:key="urn:keyMatches:schema:2011"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
schemaLocation="urn:keyMatches:schema:2011 keys.xsd">
    <key:DB
id="de.uop.dimis.air.mpqfManagement.interpreter.DummyInterpreterQbM"
>
        <key:Key>
            <key:Field>identifier</key:Field>
            <key:ReferencedDB>

de.uop.dimis.air.mpqfManagement.interpreter.DummyInterpreterQbD
            </key:ReferencedDB>

<key:ReferencedDBField>identifier</key:ReferencedDBField>
        </key:Key>
     </key:DB>
</key:KeyMatches>
```

The *KeyMatchesType* contain the Ids of source and target/referenced database (service endpoint) and the fields that should be used to identify results from both services as equal. A *KeyMatchesType* can contain multiple referenced databases. When you register a new semantic link between two Services, three semantic links will be generated. In addition to the registered link, the reflexive links will also be created by using the identifier for this database. If this particular reflexive semantic link already exists, it will be updated with the current field. Note that semantic links are symmetric (undirected edges between services). One has to be aware that semantic links are not transitive.

Java example:

```java
// get the semantic link xml file as stream
InputStream stream = ...
// unmarshal the xml file
Unmarshaller u =
NamespaceHelper.getInstance().getJAXBContextsSemanticLinks().createU
nmarshaller();
KeyMatchesType link = (KeyMatchesType) ((JAXBElement<?>)
u.unmarshal(stream)).getValue();
// register the semantic link
BackendManagement.getInstance().registerSemanticLink(link);
```

15.5.4 Frontend Functionalities

After at least one service endpoint is registered and the backend configuration is done, the QueryBroker is available for multimedia queries. The frontend functionalities are reachable through the Broker singleton (de.uop.dimis.air.Broker). Here you can start synchronous/asynchronous queries or fetch the query results for a specified asynchronous query.

Querying

The QueryBroker uses the MPEQ Query Format (MPQF) to describe queries. The XML-based query format is implemented by use of the Java Architecture for XML Binding (JAXB) . The transformed binding java code is located in the package de.uop.dimis.air.internalObjects.mpqf. It is possible to describe a query with an xml file or specify the conditions directly in Java. Since the MPQF-Standard has much complex functionality, not all query operators are currently implemented in the QueryBroker. The section Code Example shows how the operators are used properly. Implemented operators:

- Projection
- Limit
- Distinct
- GroupBy (with aggregation) over multiple attributes
- Or (half blocking, merging, using hashmaps for improved runtime)
- And (half blocking, merging, using hashmaps for improved runtime)
- SortBy over a single attribute

Synchronous Query

A synchronous query can be sent by setting the *isImmediateResponse*-field of the MPQF-Query to true. The QueryBroker blocks the query until the query process is finished and the client gets the results immediately. A possible minimal synchronous query can look like the following XML-file. Here, a single Query-By-Media (similar search for an image with the url http://any.uri.com) is sent to the QueryBroker:

```
<?xml version="1.0" encoding="UTF-8"?>
<mpqf:MpegQuery mpqfID="" …>
     <mpqf:Query>
          <mpqf:Input immediateResponse="true">
               <mpqf:QueryCondition>
                    <mpqf:Condition xsi:type="mpqf:QueryByMedia"
matchType="similar">
                         <mpqf:MediaResource resourceID="res01">
                              <mpqf:MediaResource>

<mpqf:MediaUri>http://any.uri.com</mpqf:MediaUri>
                              </mpqf:MediaResource>
                         </mpqf:MediaResource>
                    </mpqf:Condition>
               </mpqf:QueryCondition>
          </mpqf:Input>
     </mpqf:Query>
</mpqf:MpegQuery>
```

The following Java-Code example shows how the specified MPQF-Query can be forwarded to the QueryBroker and how the results can be retrieved from the response object:

Java example:

```
// get the query xml file as stream
InputStream stream = ...
// unmarshal the xml file
Unmarshaller u =
NamespaceHelper.getInstance().getJAXBContextMpqfJPSearch().createUnm
arshaller();
MpegQueryType mpqfQuery = (MpegQueryType) u.unmarshal(stream);
// query the databases
MpeqQueryType response = Broker.getInstance().query(mpqfQuery);
// get the results from the response object
List<ResultItemType> results =
response.getQuery().getOutput().getResultItem();
```

Alternatively the same query can be described and forwarded to the QueryBroker via pure Java Code:

Java example:

```
// create mpqf objects with the mpqf object factory
ObjectFactory factory = new ObjectFactory();
MpegQueryType mpqfQuery = factory.createMpegQueryType();
Query query = factory.createMpegQueryTypeQuery();
mpqfQuery.setQuery(query);
InputQueryType input = factory.createInputQueryType();
// activate the synchronous query
input.setImmediateResponse(true);
query.setInput(input);
QueryConditionType conditions = factory.createQueryConditionType();
input.setQueryCondition(conditions);
QueryByMedia qbm = factory.createQueryByMedia();
conditions.setCondition(qbm);
MediaResourceType resource = factory.createMediaResourceType();
qbm.setMediaResource(resource);
MediaLocatorType locator = factory.createMediaLocatorType();
resource.setMediaResource(locator);
locator.setMediaUri("http://any.uri.com");
```

```
// query the databases
MpeqQueryType response = Broker.getInstance().query(mpqfQuery);
// get the results from the response object
List<ResultItemType> results =
response.getQuery().getOutput().getResultItem();
```

Asynchronous Query

To start an asynchronous query the *isImmediateResponse*-field of the MPQF-Query has to be set to false. The QueryBroker sends a response with a unique MPQF query id. So, the results for the query can be fetched afterwards by referring to the retrieved id. The following code example demonstrates an asynchronous result retrieval in detail.

Java Example:

```
// create a MPQF-Query with XML or pure Java with
isImmediateResponse = false
MpegQueryType mpqfQuery = …
// query the databases
MpegQueryType response = Broker.getInstance().query(mpqfQuery);
// get the mpqf id for result fetching
String mpqfID = response.getMpqfID();
// ... wait ...
// create the fetch query
ObjectFactory factory = new ObjectFactory();
MpegQueryType fetchMpqf = factory.createMpegQueryType();
Query query = factory.createMpegQueryTypeQuery();
fetchMpqf.setQuery(query);
FetchResult fetch = factory.createMpegQueryTypeQueryFetchResult();
// refer to the retrieved MPQF-ID
fetch.setQueryID(mpqfID);
query.setFetchResult(fetch);
// fetch the results from the query broker
MpegQueryType response2 = Broker.getInstance().query(fetchMpqf);
// get the results from the response object
List<ResultItemType> results =
response2.getQuery().getOutput().getResultItem();
```

If the broker hasn't finished the retrieval already or an error occurs during the processing the system message in the MPQF-Response has a corresponding status message (e.g. "101 – Server resource busy"). These messages can be retrieved via java as follows:

Java Example:

```
response.getQuery().getOutput().getSystemMessage().getStatus();
```

See the MPQF-Standard for more information about system messages and error codes.

Complex Query Example

The following XML code shows a more complex query example. The result count is limited to 10 items (maxItemCount), the results are sorted ascending by the “identifier”-field and a projection on the field “description” (ReqField) takes place. The query condition consists of a join of a QueryByMedia and a QueryByDescription, which contains metadata constraints described by the MPEG-7 metadata schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<mpqf:MpegQuery mpqfID="101" xmlns:mpqf="urn:mpeg:mpqf:schema:2008"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="urn:mpeg:mpqf:schema:2008
mpqf_semantic_enhancement.xsd">
    <mpqf:Query>
        <mpqf:Input>
            <mpqf:OutputDescription maxItemCount="10"
distinct="true">
                <mpqf:ReqField
typeName="description"></mpqf:ReqField>
                <mpqf:SortBy xsi:type="mpqf:SortByFieldType"
order="ascending">
                    <mpqf:Field>identifier</mpqf:Field>
                </mpqf:SortBy>
            </mpqf:OutputDescription>
            <mpqf:QueryCondition>
                <mpqf:Condition xsi:type="mpqf:AND">
                    <mpqf:Condition xsi:type="mpqf:QueryByMedia">
                        <mpqf:MediaResource resourceID="ID_5001">

<mpqf:MediaUri>http://tolle.uri/1</mpqf:MediaUri>
                        </mpqf:MediaResource>
                    </mpqf:Condition>
                    <mpqf:Condition
xsi:type="mpqf:QueryByDescription">
                        <mpqf:DescriptionResource
resourceID="desc001">
                            <mpqf:AnyDescription

xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004"

xsi:schemaLocation="urn:mpeg:mpeg7:schema:2004 M7v2schema.xsd">
                                <mpeg7:Mpeg7>
                                    <mpeg7:DescriptionUnit

xsi:type="mpeg7:CreationInformationType">
                                        <mpeg7:Creation>
                                            <mpeg7:Title>Example
Title</mpeg7:Title>
                                        </mpeg7:Creation>
                                    </mpeg7:DescriptionUnit>
                                </mpeg7:Mpeg7>
                            </mpqf:AnyDescription>
```

```
                        </mpqf:DescriptionResource>
                    </mpqf:Condition>
                </mpqf:Condition>
            </mpqf:QueryCondition>
        </mpqf:Input>
    </mpqf:Query>
</mpqf:MpegQuery>
```

Query Execution Tree Evaluation

The query aggregator evaluates the query execution plan (QEP). The result of this evaluation is a number of results that will later be returned to the querying client. Every leaf of the QEP has a reference to a sub-query, which includes one or many service executions, so the results from the sub-query are accessible. For the evaluation an iterator driven model is used. Every node has a next and hasNext method to get the next result item from its children. Since the QEP can be a DAG, hasNext and next need a parameter to decide on which path in the DAG they are called, to be able to choose the correct iterator. HasNext checks if there is a next result item in the local result list. If there is one, next can call this item. If not, hasNext tries to query its child nodes for a next valid result item. If this is possible the new item will be appended in the local result list. Only if no successor can be computed hasNext will return false. So make always sure to call hasNext before next. There are blocking, half blocking and none blocking operators. A blocking operator needs all results from its children to decide which result will returned next. The SortBy operator is a blocking operator. An operator is half blocking, if it doesn't need all results from every child. The AND operator is implemented in such a way. None blocking operators like LIMIT can forward results without knowing every other possible result. Some operators have to merge results. If two results are equal (according to the specific semantic link) they must be merged. Merging operators are for example AND and OR. Merging two results means that one result is augmented with additional information from the second result. No information is overwritten.

15.5.4.1 *Code Example*

The following code examples describe a full initial setup of the QueryBroker. An implementation of a QueryByDescription-Service is presented (ExampleService.java), followed by the registration of this service and the registration of the semantic link to a fictional second service endpoint. The semantic link registration can be omitted if the default semantic link ("identifier") is demanded.

Java example:

```
// ExampleService.java
public class ExampleService implements Service {
    @Override
    public MpegType execute(MpegQueryType distributedQuery) {
        // get the query conditions
        BooleanExpressionType conditions =

distributedQuery.getQuery().getInput().getQueryCondition();

        // … do your program logic with the query conditions …

        // create a result container for the computed results
        ObjectFactory mpqfObjFac = new ObjectFactory();
```

```
        MpegQueryType result = mpqfObjFac.createMpegQueryType();
        Query qry = mpqfObjFac.createMpegQueryTypeQuery();
        result.setQuery(qry);
        OutputQueryType oqt = mpqfObjFac.createOutputQueryType();
        qry.setOutput(oqt);
        List<ResultItemType> resultItems = oqt.getResultItem();

        // for each result of the service endpoint create a result
item and add it
        // to the results list
        de.uop.dimis.air.internalObjects.jpsearch.ObjectFactory
jpsearchObjFac =
                        new
de.uop.dimis.air.internalObjects.jpsearch.ObjectFactory();
        for(…) {
            ResultItemType resultItem =
mpqfObjFac.createResultItemType();
            resultItmes.add(resultItem);
            Description description =
mpqfObjFac.createResultItemTypeDescription();
            resultItem.getDescription().add(description);
            JPSearchCoreType coreType =
jpsearchObjFac.createJPSearchCoreType();
            description.getContent().add(coreType);

            // set the result properties in the jpsearch coreType
object. (e.g.
            // identifier)
            coreType.setIdentifier("…");
            // set the origin to identify from which service
endpoint the
            // result item was generated
            resultItem.setOriginID("MedicoExecuteDICOM");
        }

    return result;
    }
}
// RegisterExampleService.xml
<?xml version="1.0" encoding="UTF-8"?>
<mpqf:MpegQuery mpqfID="" xmlns:mpqf="urn:mpeg:mpqf:schema:2008"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mpeg:mpqf:schema:2008
mpqf_semantic_enhancement.xsd">
       <mpqf:Management>
              <mpqf:Input>
                     <mpqf:DesiredCapability>
                            <!-- Query By Description: 100.3.6.2 -->
                            <mpqf:SupportedQueryTypes
href="urn:mpeg:mpqf:2008:CS:full:100.3.6.2" />
                     </mpqf:DesiredCapability>
                     <mpqf:ServiceID>
                            de.uop.dimis.air.ExampleService
```

```
                    </mpqf:ServiceID>
             </mpqf:Input>
      </mpqf:Management>
</mpqf:MpegQuery>
```

We assume that a second service with the identifier “de.uop.dimis.air.SecondService” gets registered, too. The two databases have the semantic link between the fields “identifier” (ExampleService) and “title” (SecondService).

```
//SemanticLinks.xml
<?xml version="1.0" encoding="UTF-8"?>
<key:KeyMatches xmlns:key="urn:keyMatches:schema:2011"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
schemaLocation="urn:keyMatches:schema:2011 keys.xsd">
    <key:DB id="de.uop.dimis.air.ExampleService">
        <key:Key>
            <key:Field>identifier</key:Field>
            <key:ReferencedDB>
                de.uop.dimis.air.SecondService
            </key:ReferencedDB>
            <key:ReferencedDBField>title</key:ReferencedDBField>
        </key:Key>
     </key:DB>
</key:KeyMatches>
```

Now the xml files are loaded and transferred to the QueryBroker. This has to be done only once for initialization. After theses steps the QueryBroker is available for query requests.

Java example:

```
// Initialize the QueryBroker and register the service (e.g. in a
main-method)
// get the register query xml file as stream
InputStream stream =

ExampleService.class.getResourceAsStream("RegisterExampleService.xml
");
// unmarshal the xml file
Unmarshaller u =

NamespaceHelper.getInstance().getJAXBContextMpqfJPSearch().createUnm
arshaller();
MpegQueryType mpqfQuery = (MpegQueryType) u.unmarshal(stream);
// register the database
BackendManagement.getInstance().registerDatabase(mpqfQuery);
// get the semantic link xml file as stream
InputStream stream =

ExampleService.class.getResourceAsStream("SemanticLinks.xml");
// unmarshal the xml file
```

```
Unmarshaller u =
NamespaceHelper.getInstance().getJAXBContextsSemanticLinks().createU
nmarshaller();
KeyMatchesType link = (KeyMatchesType) ((JAXBElement<?>)
u.unmarshal(stream)).getValue();
// register the semantic link
BackendManagement.getInstance().registerSemanticLink(link);
// the queryBroker is now ready for queries
```

15.6 References

[DICOM]	Digital Imaging and Communications in Medicine; The DICOM Standard
[Döller 08a]	M. Döller, R. Tous, M. Gruhne, K. Yoon, M. Sano, and I. S. Burnett, “The MPEG Query Format: On the Way to Unify the Access to Multimedia Retrieval Systems,” IEEE Multimedia, vol. 15, no. 4, pp. 82–95, 2008.
[Döller 08b]	Mario Döller, Kerstin Bauer, Harald Kosch and Matthias Gruhne. Standardized Multimedia Retrieval based on Web Service technologies and the MPEG Query Format. Journal of Digital Information, 6(4):315-331,2008.
[Döller 10]	M. Döller, F. Stegmaier, H. Kosch, R. Tous, and J. Delgado, “Standardized Interoperable Image Retrieval,” in Proceedings of the ACM Symposium on Applied Computing, Track on Advances in Spatial and Image-based Information Systems, (Sierre, Switzerland), pp. 881–887, 2010.
[DublinCore]	Dublin Core Metadata Initiative. Dublin Core metadata element set – version 1.1: Reference description. http://dublincore.org/documents/dces/, 2008.
[Gruhne 08]	Matthias Gruhne, Peter Dunker, Ruben Tous and Mario Döller. Distributed Cross-Modal Search with the MPEG Query Format. In 9th International Workshop on Image Analysis for Multimedia Interactive Services, pages 211-224, Klagenfurt, Austria, May 2008. IEEE Computer Society.
[JAXB]	Java Architecture for XML Binding (JAXB), Metro Projekt, http://jaxb.dev.java.net/
[Matinez 02]	J. M. Matinez, R. Koenen and F. Pereira. MPEG-7. IEEE Multimedia, 9(2):78-87, April-June 2002.
[MAVEN]	Apache Maven, Apache Software Foundation, http://maven.apache.org/
[MPEG-7]	ISO/IEC 15938-1:2002 - Information technology -- Multimedia content description interface -- http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=46427
[Smith 08]	J. R. Smith, “The Search for Interoperability,” IEEE Multimedia, vol. 15, no. 3, pp. 84–87, 2008.
[Spring]	The Spring Framework, SpringSource, 2012, http://www.springsource.org/

[Stegmaier 09a]	F. Stegmaier, W. Bailer, T. Bürger, M. Döller, M. Höffernig, W. Lee, V. Malaisé, C. Poppe, R. Troncy, H. Kosch, and R. V. de Walle, "How to Align Media Metadata Schemas? Design and Implementation of the Media Ontology," in Proceedings of the 10th International Workshop of the Multimedia Metadata Community on Se- mantic Multimedia Database Technologies in conjunction with SAMT, vol. 539, (Graz, Austria), pp. 56–69, December 2009.
[Stegmaier 09b]	Florian Stegmaier, Udo Gröbner, Mario Döller. "Specification of the Query Format for medium complexity problems (V1.1)", Deliverable CTC 2.5.15 of Work-Package 2 ("Video, Audio, Metadata, Platforms") of THESEUS Basic Technologies, 2009.
[Stegmaier 10]	Florian Stegmaier, Mario Döller, Harald Kosch, Andreas Hutter and Thomas Riegel. "AIR: Architecture for Interoperable Retrieval on distributed and heterogeneous Multimedia Repositories". 11th International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS2010), Desenzano del Garda, Italy, p 1-4.
[XPath]	XML Path Language (XPath), 2.0 (Second Edition, W3C Recommendation, 14. December 2010, http://www.w3.org/TR/xpath20/

16 FIWARE ArchitectureDescription Data SemanticAnnotation

You can find the content of this chapter as well in the wiki of fi-ware.

16.1 Copyright

16.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

16.3 Overview

16.3.1 Introduction to the Semantic Annotation GE

The principle standing behind Semantic Web is to evolve the "link" concept from an unspecified element describing the relationship between two element into a "named relationship" in order to clarify which is(are) the relationship(s) between those elements.

That is the main reason why RDF, the language of Linked Open Data was invented. RDF is based on Triples, in the form of<SUBJECT><PREDICATE><OBJECT>.

The predicate (and sometimes the object) can describe objects and their relationships. Semantic Annotator is basically a tool which tries to identify important entities(places,persons,organization) a text and describe them with Linked Open Data.

This GE provides a general purpose text analyzer to identify and disambiguate LOD (Linked Open Data) resources related to the entities in the text. It is build following a modular approach to optimize and distribute text processing & LOD sources (plug-in). Also it allows RDF triple generation that easily links to LOD resources.

The main conceptual idea of the SA GE is shown in the Figure 1 below.

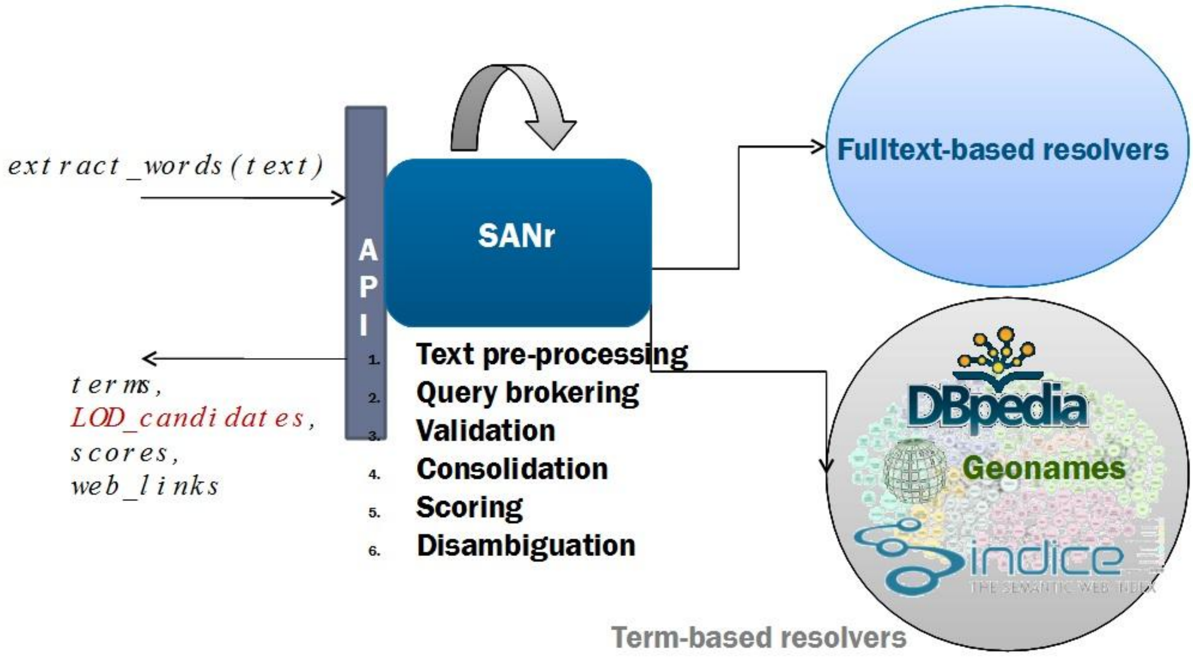


Figure 1: Conceptual Model of Semantic Annotation GE

16.3.2 Target usage

This GE may be used in the augmenting of content (news, books, etc) with additional information and links to LOD. It provides filtering and search based on LOD resources used as categories/tags.

Target users are all stakeholders that want to enrich textual data (tags or text) with meaningful and external content.

In the media era of the web, much content is text-based or partially contains text, either as media itself or as metadata (e.g. title, description, tags, etc). Such text is typically used for searching and classifying content, either through folksonomies (tag-based search), predefined categories, or through full-text based queries. To limit information overload with meaningless results there is a clear need to assist this searching process with semantic knowledge, thus helping in clarifying the intention of the user. This knowledge can be further exploited not only to provide the requested content, but also to enrich results with additional , yet meaningful content, which can further satisfy the user needs.

Semantics, and in particular Linked Open Data (LOD), is helpful in both annotating & categorizing content, but also in providing additional rich information that can improve the user experience.

As end-user content can be of any type, and in any language, such enabler requires a general purpose & multilingual approach in addressing the annotation task.

Typical users or applications can be thus found in the area of eTourism or eReading, where content can benefit from such functionality when visiting a place or reading a book, for example being provided with additional information regarding the location or cited characters.

The pure semantic annotation capabilities can be regarded as helpful for editors to categorize content in a meaningful manner thus limiting ambiguous search results (e.g. an article wouldn't be simply tagged with apple, but with its exact concept, i.e. a fruit, New York City or the brand)

16.3.3 Basic Design Principles

The Enabler has been designed following a modular approach, as it is shown in Figure 1. This way each component in the enabler can be developed or changed, given that it gives the same input/output format.

This leaves open the road to change data sources in order to have other data sources than dbpedia or geonames or to change the process standing behind the candidate's choice for each entity.

16.4 Basic Concepts

The GE has a web API, supports multilingual texts (it, en, es, pt) and includes "candidate" LOD resources and performs disambiguation. As result the GE created external links and HTML snippets (LOD information mashup).

The following modules listed in the Figure 2 are supposed to be included into the SA GE for its features and functionalities.

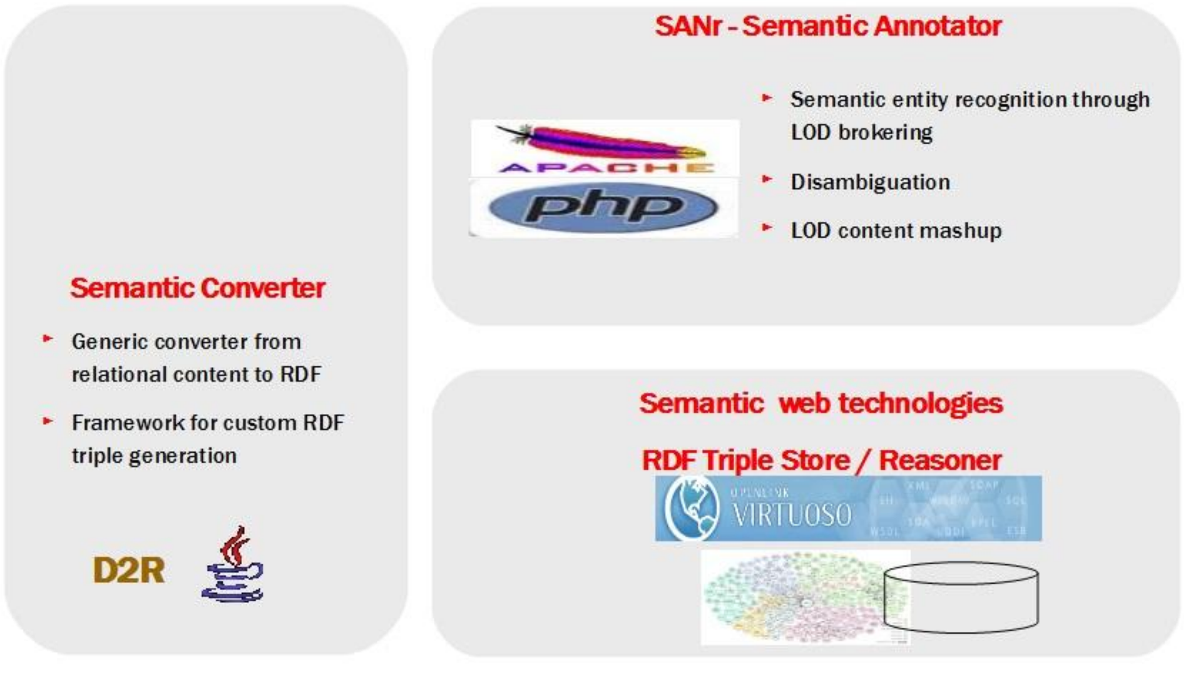


Figure 2: Semantic Annotation GE Modules

16.5 Main Interactions

The enabler basically consists of an API, which can be called by a simple HTTP GET request to this URL.

http://semantican.lab.fi-ware.eu/ajax/extract_words.php?text=

with a text to analyze as input which has to be passed as "text" parameter as shown above.

This API will:

1. Identigy Text Language
2. Identify Entities (People, Places, Organizations) in the Text
3. For each found entity It searches over Semantic Data Sources (DBPedia and Geonames) for related Linked Open Data Objects.
4. The found LOD objects for each entity are returned in JSON Format as "candidates". Eache candidate has a score. The candidate with the highest score is flagged as "preferred".
5. The query is logged into a DB with an ID.

Here's an example of the JSON return.


```
{
    "queryId": "12143",
    "lang": "it",
    "keywords": "Mario+Monti",
    "extags": "Mario Monti",
    "freeling": "Mario_Monti",
    "proc_time": "13",
    "terms": [
        {
            "id": "tc-Mario+Monti",
            "term": "Mario Monti",
            "candidates": [
                {
                    "id": "tag--Mario_Monti--
http://dbpedia.org/resource/Mario_Monti",
                    "label": "Mario Monti",
                    "uri":
"http://dbpedia.org/resource/Mario_Monti",
                    "type": "user",
                    "ext": "Mario Monti",
                    "extra": [],
                    "wrapper": "dbpedia",
                    "lev": "2",
                    "sim": "0.909090909091",
                    "sis": "1",
                    "jw": "0.963636363636",
                    "sc": "1",
                    "class": "empty",
                    "preferred": "true"
                }
            ],
            "html": "<fieldset><div class=panel><div class=header>A
proposito di <b>Mario Monti</b></div><div
class=panel_body></div></div><div class=panel><div
class=panel_body><img
src='http://upload.wikimedia.org/wikipedia/commons/thumb/3/33/Il_Pre
sidente_del_Consiglio_incaricato_Mario_Monti_(cropped).jpg/200px-
```

```
Il_Presidente_del_Consiglio_incaricato_Mario_Monti_(cropped).jpg'
height=160 /><br><div class=info>Ãˆ senatore a vita dal 9 novembre
2011 e dal successivo 16 novembre assume, per la prima volta,
l'incarico di Presidente del Consiglio dei Ministri della Repubblica
Italiana e allo stesso tempo di Ministro dell'Economia e delle
Finanze dello stesso governo. Presidente dell'UniversitÃ  Bocconi
dal 1994, Monti Ã¨ stato c...<ul><li><a
href='http://www.guardian.co.uk/world/mario-monti'
target='_blank'>Link
utile</a></li></ul></div></div></div></fieldset><fieldset><legend>Co
ncetti associati a <strong>Mario Monti</strong></legend><ul><li><img
src='img/user.png' alt='user' title='user'> <a
href='http://dbpedia.org/resource/Mario_Monti' target='_blank'
title='[2-0.909090909091-0.963636363636/1]' >Mario Monti</a>
(dbpedia)</li></ul></fieldset>",
            "class": "empty"
        }
    ]
}
```


Moreover, by setting the 'html_snippet=on' parameter in the request URL, an HTML snippet for the preferred DBPedia entry is returned if possible. The HTML Snippet contains a Picture and Short Abstract for the resource.

17 FIWARE ArchitectureDescription Data SemanticSupport

You can find the content of this chapter as well in the wiki of fi-ware.

17.1.1 Copyright

17.1.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

17.2 Overview

17.2.1 Introduction to the Semantic Application Support GE

The Semantic Web Application Support enabler aims at providing an effective environment for developers to implement and deploy high quality Semantic Web-based applications. The Semantic Web was first envisioned more than a decade ago by Tim Berners-Lee, as a way of turning the Web into a set of resources understandable not only for humans, but also by machines (software agents or programs), increasing its exploitation capabilities. The Semantic Web has focused the efforts of many researchers, institutions and IT practitioners, and received a fair amount of investment from European and other governmental bodies. As a result of these efforts, a large amount of mark-up languages, techniques and applications, ranging from semantic search engines to query answering system, have been developed. Nevertheless the adoption of Semantic Web from the IT industry is still following a slow and painful process.

In recent years, several discussions had taken place to find out the reasons preventing Semantic Web paradigm adoption. There is a general agreement that those reasons range from technical (lack of infrastructure to meet industry requirements in terms of scalability, performance. distribution, security, etc.) to engineering (not general uptake of methodologies, lack of best practices and supporting tools), and finally commercial aspects (difficulties to penetrate in the market, lack of understanding of the main strengths and weaknesses of the semantic technologies by company managers, no good sales strategies, etc.).

The Semantic Application Support enabler addresses part of the abovementioned problems (engineering and technical) from a data management point of view, by providing:

- An infrastructure for metadata publishing, retrieving and subscription that meets industry requirements like scalability, distribution and security. From now and so on, we will refer to this infrastructure as SWAS Infrastructure.
- A set of tools for infrastructure and data management, supporting most adopted methodologies and best practices. From now and so on, we will refer to this tools as SWAS Engineering Environment.

17.2.2 Target usage

Target users are mainly ontology engineers and developers of semantically-enabled applications that need RDF storage and retrieval capabilities. Other GE from the FI-WARE, such as for example the GE for semantic service composition or the query broker, or from the usage areas of the PPP that need semantic infrastructure for storage and querying are also target users of this GE.

17.2.3 Example Scenario

There is a need for semantically-enabled applications in many fields and domains, ranging from research projects to enterprise intranets or public web sites. Semantic applications often rely on ontologies and knowledge bases to develop business functionality such as discovery, composition, annotation, etc., with the aim of enhancing exploitation capabilities of resource (services, text documents, media documents, etc.). The need of an infrastructure that ease the development, storage and use of ontologies and allow practitioners to efficiently manage their knowledge bases, providing the means to manage metadata effectively is therefore of paramount interest.

The TaToo (http://www.tatoo-fp7.eu/tatooweb/) project can be taken as an example in order to show how this generic enabler can help to future internet application developers. TaToo is a research project on the environmental domain with the goal of developing tools to facilitate the discovery of environmental resources. In order to enhance the discovery process, the application stores annotations (metadata) of existing environmental resources by tagging them with ontology terms. Therefore, a ontology framework [Pariente 2011] has been developed including three domain ontologies that describe three different environmental domains plus a bridge ontology that allow cross domain interoperability. Moreover, TaToo ontologies are not built from scratch but by reusing, complete or partially, existing ontologies publicly available over the internet. Nowadays the TaToo ontology framework is the result of the integration of more than 15 ontologies. The development of such a framework is a complex task, involving several domain experts and ontology developers. To achieve a common understanding, is imperative the usage of appropriate methodology and tools. In TaToo the NeOn Methodology [Suarez-Figueroa 2008] and the NeOn Toolkit [NeOn-Toolkit], one of the baseline assets of this generic enabler, have been the basis of the ontology engineering process. The Neon Toolkit helped TaToo's ontology developers to apply the NeOn methodology to develop ontologies providing several functionality such us ontology edition, ontology modularization, ontology search, etc. Besides, these ontologies are expected to evolve in time, and therefore they would need a system that helps the ontology expert to tackle the ontology evolution problems. This is not completely covered by the NeOn Toolkit, as there are aspects such as ontology versioning, knowledge base maintenance, workspace environments, etc. that are not fully covered by the tool. This functionality will be developed in the scope of FI-WARE project. Figure SWAS-1 shows a screenshot of NeOn Toolkit being used in the scope of TaToo.

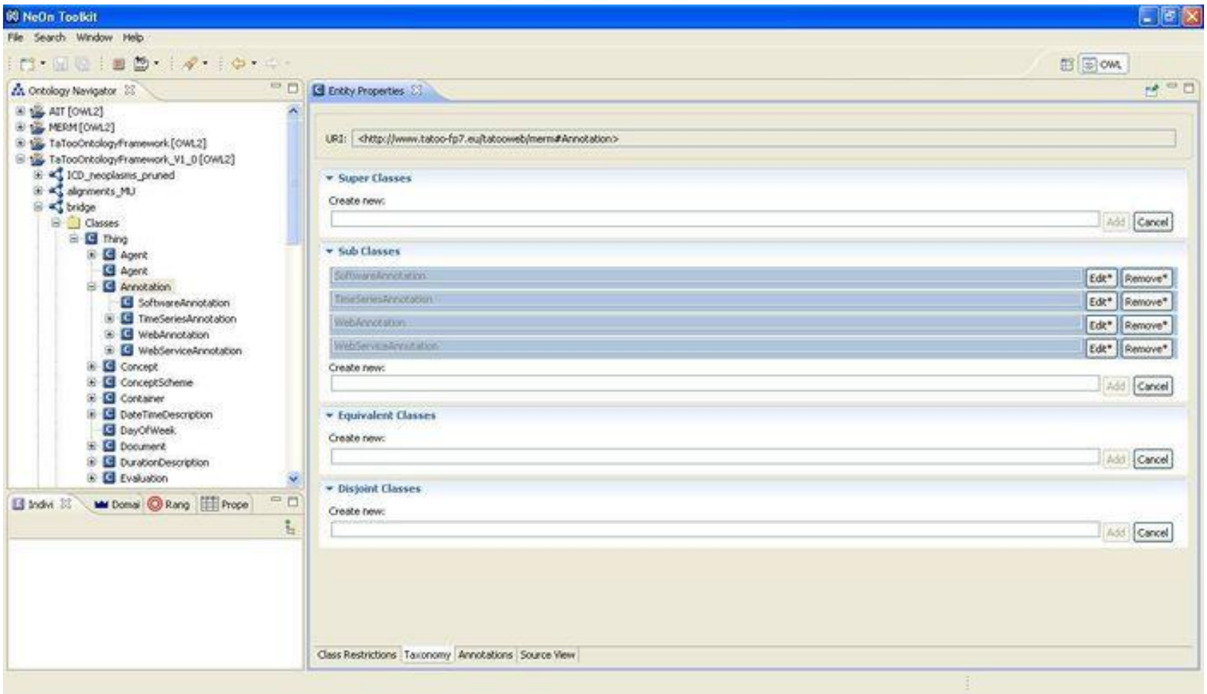


Figure SWAS-1: NeOn Toolkit screenshot

Once ontologies are developed they need to be uploaded to a knowledge base with inference capabilities to be used by business logic components. In TaToo, Sesame [Sesame] and OWLIM [OWLIM], two of the assets selected as baseline assets for this enabler, have been used as knowledge base implementation. However Sesame and OWLIM are just RDF / OWL oriented storages, so there is a lack of knowledge base management capabilities. As an example, once a ontology is loaded into a Sesame workspace it is not possible to keep track of this fact, so in case the ontology evolve in time, there is no possibility to establish a relation between the workspace and the loaded ontology in order to update the workspace. This kind of knowledge base management problems will be tackled and solved by the Semantic Support Application GE in the scope of FI-WARE.

Summarizing, a project such as TaToo might benefit from an enabler that provides an ontology and knowledge base management system integrated with an ontology engineering environment supporting strong ontology development methodology, covering the whole semantic web application lifecycle. This is clearly extensible to many different Semantic Web-based applications.

17.3 Basic Concepts

This section introduces the basic concepts related to the Semantic Support Application GE including ontologies, ontology languages and ontology development methodologies.

17.3.1 Ontologies

[Gruber 1993] introduced the concept of ontology as “a formal explicit specification of a shared conceptualization”. Thus, in the Semantic Web, ontologies play the role of a formal (machine-understandable) and shared (in a domain) backbone. Ontologies are becoming a clear way to deal with integration and exploitation of resources in the several domains.

Starting from Gruber's definition is it possible to infer some of the key features that make ontologies a valuable knowledge engineering product:

- Ontologies are formal, so they are supposed to be machine-understandable.
- Ontologies have explicit definitions, so they are storable, interchangeable, manageable, etc.
- Ontologies are shared, so they are supposed to be agreed, supported and accessible by a broad community of interest.
- Ontologies are a conceptualization, so they are supposed to be expressive enough to model wide knowledge areas.

In order to efficiently develop ontology networks that really fulfill these features, a wide range of elements are needed, ranging from appropriate methodologies, to tools supporting those methodologies and appropriate infrastructures to allow management of the ontology lifecycle. Providing such a support is the aim of this GE.

To do so, some decisions have been taken in order to limit the scope of the GE:

- To select [OWL-2 RL] as reference language for ontology formalization.
- To select NeOn Methodology as reference methodology for ontology development.

Both decisions will be discussed in the following sections.

17.3.2 OWL-2

Since the inception of the ontologies, several ontology languages, with different expressivity, serialization and underlying logic formalisms have rise and fall (OWL, WSML, F-Logic, OIL, KIF, etc.). Sometimes these languages differ in their serialization, sometimes in their background logic and sometimes they are just designed with a different purpose. Therefore, provide functionality for every single ontology language is almost an impossible task. In consequence, in the scope of the Semantic Web Application Generic Enabler OWL-RL (a decidable subset of OWL, the W3C standard and most popular ontology language) has been selected as reference for ontology definition. Some of the reasons supporting this decision are now introduced:

- Since October 2009 OWL-2 is a W3C recommendation for ontology definition.
- OWL-2 RL provides a good trade-off between expressivity and performance. Inference over OWL-2 RL guarantees that inference process will finish in a finite amount of time.
- OWL-2 RL is based in previous W3C standards such as [RDF] and [RDFs] so previous ontologies could be also managed by the proposed infrastructure.

17.3.3 Ontology Engineering

Semantic Web Application Support GE aims to provide the means for FI Applications developers to develop Semantic Web enabled applications efficiently. Ontology development, one of the key points in these applications, is a complex, expensive and time-consuming process that includes different activities, such as specifying requirements, information extraction, logical modeling, etc. In order to efficiently manage this process, it is necessary to use a methodology and its supporting tools. Due to its adoption and maturity, Semantic Application Support GE will provide the means to support the NeOn Methodology.

The NeOn Methodology defines a methodology for ontology development that covers the whole ontology lifecycle. The NeOn methodology includes extracted elements of previous methodologies like METHONTOLOGY [Fernandez 1997], On-To-Knowledge [OnToKnowledge 2001] and DILIGENT [DILIGENT 2004]. The NeOn methodology increases the level of descriptive detail, and provides two new features: ontology creation from existing resources (both ontological or not) and ontology contextualization. In this way, NeOn offers a general methodology for ontology development useful across different technological platforms and that specifies each process and activity of the methodology, defining its purpose, inputs, outputs, involved actors, applicable techniques, tools and methods, when its execution is necessary, etc.

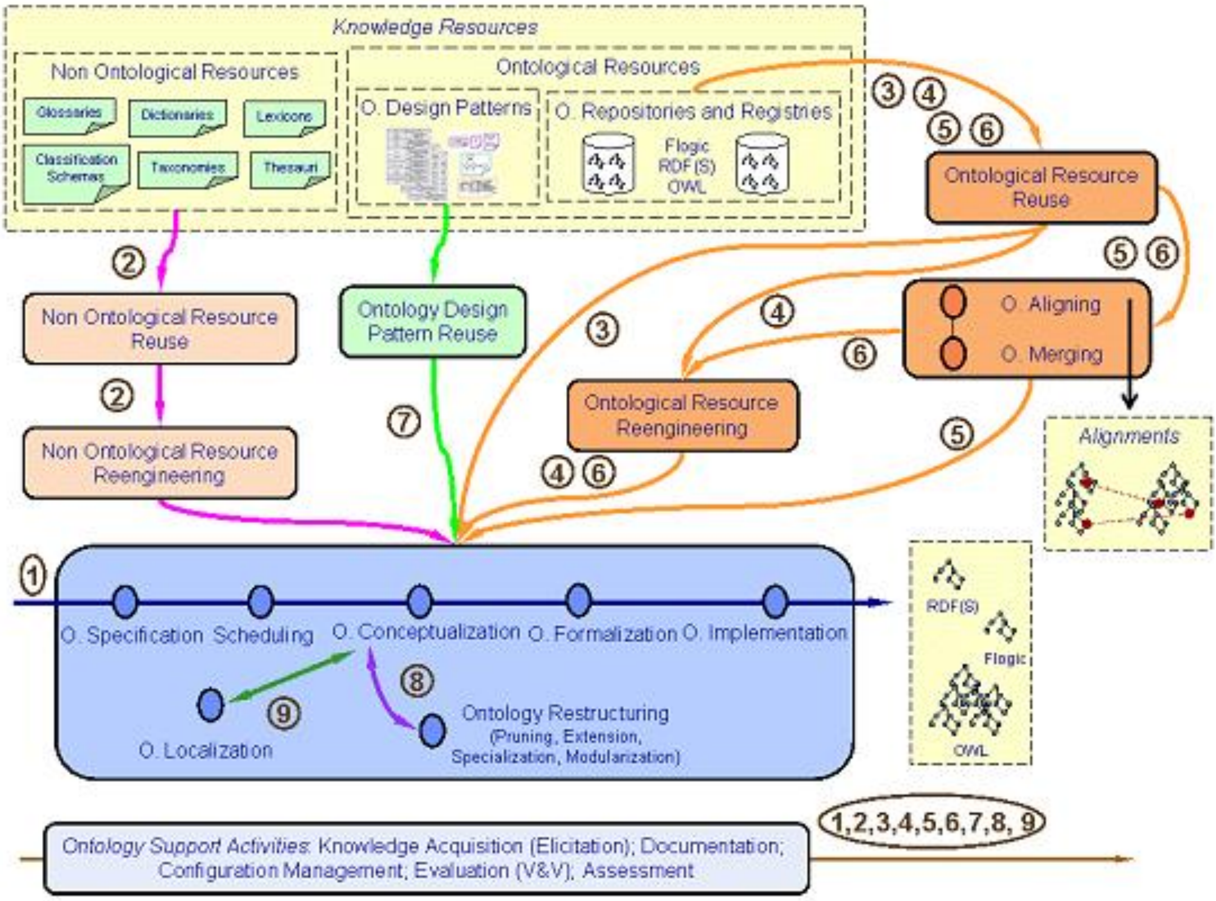


Figure SWAS-2: NeOn Methodology overview (from Suárez-Figueroa, 2008, with permission)

The NeOn methodology presents and describes nine of the most common scenarios that may arise during ontology development:

1. Specification for implementation from scratch.
2. Reusing and re-engineering non-ontological resources.
3. Reusing ontological resources.
4. Reusing and re-engineering ontological resources.
5. Reusing and merging ontological resources.
6. Reusing, merging and re-engineering ontological resources.

7. Reusing ontology design patterns.
8. Restructuring ontological resources.
9. Localizing ontological resources.

Scenario 1 represents the base case, whereas the rest of the scenarios are related to it as shown in Figure SWAS-2. For each of these scenarios, the NeOn methodology establishes detailed guidelines, tools to use, etc.

The Semantic Web Application Support GE should provide an ontology engineering environment supporting processes and activities outlined in the NeOn methodology.

17.4 Semantic Application Support GE Architecture

The objective of the Semantic Application Support GE is to facilitate the Ontology Engineering process providing a set of tools that allow the ontology reutilization using repositories to publish and share ontologies between projects. The developer can use the published ontologies to create semantic repositories to support specific needs.

In order to satisfy the previous objective, the Semantic Application Support GE is divided in a client-side Engineering Environment and a server-side Infrastructure. Figure SWAS-3 presents the SWAS Infrastructure architecture.

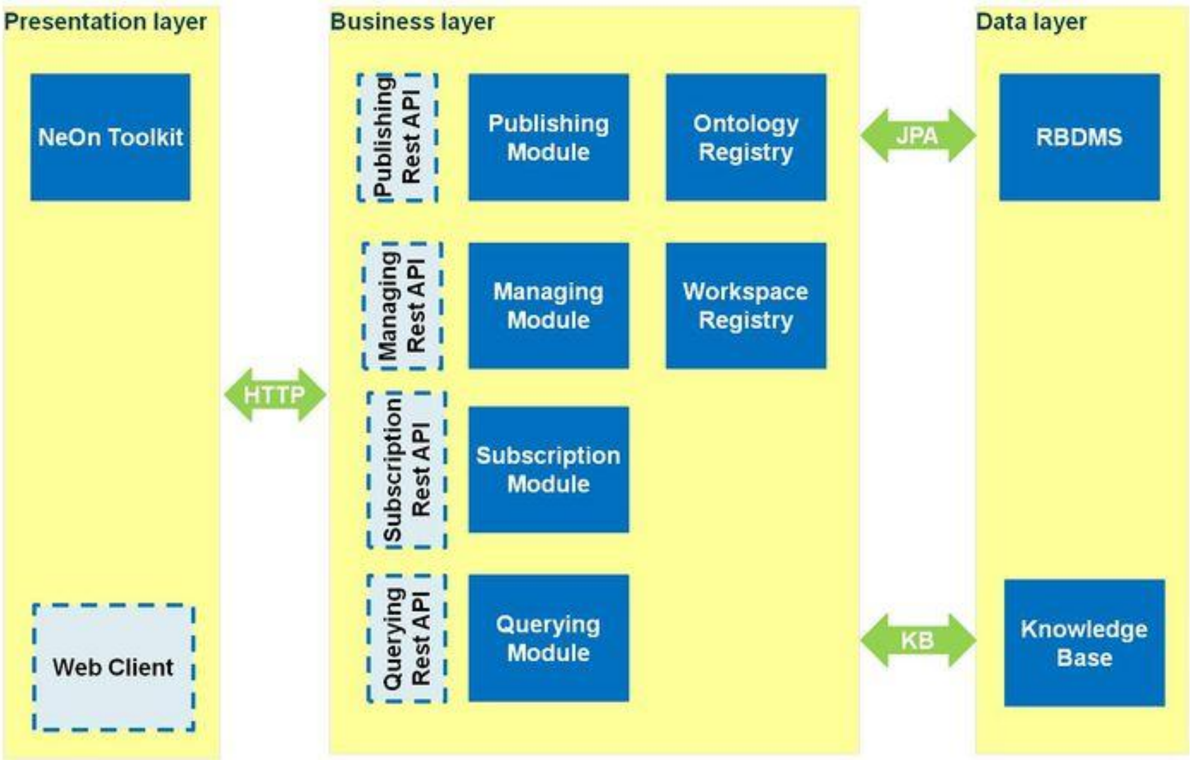


Figure SWAS-3: SWAS Infrastructure architecture

As it is shown in the diagram, it follows a typical three layer Java Enterprise Architecture. Components included in business and presentation layers are JEE based. In the data layer, two components can be found:

- A relational database, that will be used by Ontology Registry to store ontology documents loaded into the GE.

- A Knowledge Base providing OWL-2RL support. This Knowledge Base will be used by ontology and workspace registries to store ontology and workspace related metadata and by managing, querying and publishing modules to provide their functionality.

Business components will interact with data layer components by means of two different mechanisms. To interact with the relational database, business components will use JPA (Java Persistence API) that make business components database system independent. Unfortunately such an abstraction mechanism is not available for knowledge base required interaction so business components interacting with the knowledge base will be knowledge base implementation dependent. In Semantic Web Application Support reference implementation, the combination of Sesame and OWLIM has been chosen as knowledge base implementation. Knowledge base independence feature will be study for future releases.

Business Layer contains following components:

- Ontology registry that manages ontologies loaded into the system and its related metadata. Operations such us retrieving / uploading ontology, retrieving / uploading metadata, etc would be provided by this component. A description of methods provided for FI-WARE first release can be found in Backend Functionality section.
- Workspace registry that manages workspaces and their related metadata created by users to be used by their semantic enable applications. Operations such us creating / deleting a workspace, listing ontologies loaded into the workspace, etc would be provided by this component. Description of methods belonging to this component will be described in future FI-WARE releases.
- Publishing module that allow user to publish data into the GE. Data can be either ontologies or RDF serialized content. In case of ontologies, publishing module will rely on ontology registry functionality. In case of RDF serialized content, publishing module will store the content in proper knowledge base workspace in collaboration with workspace registry. In both cases publishing module will update subscription module if needed. Description of methods belonging to this component will be described in future FI-WARE releases.
- Managing module that allow users to monitor the status of the GE. Operations such as retrieving a list of available ontologies, retrieving a list of subscriptions, etc will be provided by this module. Managing module will rely on the rest of business components to provide its functionality. Description of methods belonging to this component will be described in future FI-WARE releases.
- Subscription module that allows users to subscribe to events produced in the GE. Operations such as subscribing to ontology updates or workspace modifications will be provided by this module. Description of methods belonging to this component will be described in future FI-WARE releases.
- Querying module that allows users to query their workspace following SPARQL Query Protocol. Description of methods belonging to this component will be described in future FI-WARE releases.

In order to provide GE functionality in a platform independent way, several Rest APIs will be developed. In this first FI-WARE release, a subset of methods belonging to publish and managing APIs will be provided. Therefore, clients or presentation layer applications will interact with business components by means of HTTP requests / responses.

SWAS Engineering Environment provides comprehensive support for the ontology engineering life-cycle. Concrete details about SWAS Engineering Environment functionality will be provided in Frontend Functionality section. SWAS architecture is based on Eclipse architecture, a leading development environment providing a technical layer for easy creation of new features and supported for a huge development community. Figure SWAS-4 shows the SWAS Engineering Environment architecture.

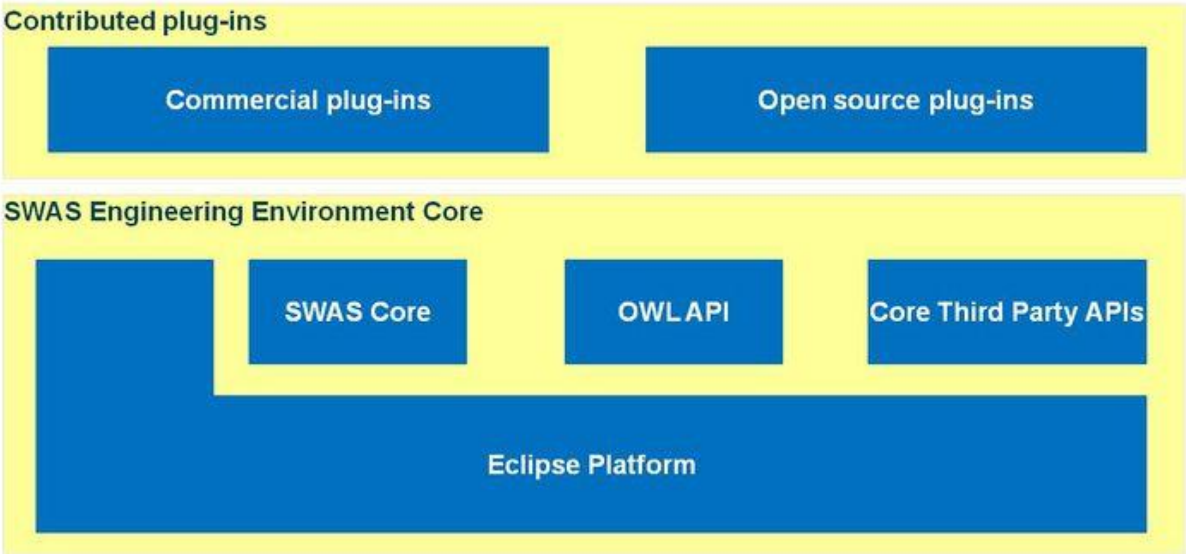


Figure SWAS-4: SWAS Engineering Environment architecture

As it shown in the diagram SWAS Engineering Environment is divided into two layers, the SWAS Engineering Environment core and the contributed plug-ins. The SWAS Engineering Core provides the core ontology editing functionality. The contributed plug-ins are extensions that provide extra functionality supporting different phases of the NeOn Methodology.

17.5 Main Interactions

17.5.1 Modules and Interfaces

This section covers the description of the Semantic Web Application Support GE main functionality. The description of this functionality is based on the functionality provided by the baseline asset in FI-WARE first release. Section Backend functionality describes functionality (methods) provided to agents in a service like style. Section Frontend functionality describes functionality provided to human users through a GUI.

17.5.2 Backend Functionality

Backend functionality describes functionality provided by the GE enabler as service invocation methods for both human or computer agents. As described in Architecture section, this functionality is accessible by means of Rest Web Services API. In this first FI-WARE release, a sub set of methods belonging to publishing and managing rest APIs will be provided:

- ***Publishing Rest API***.
    - ***Get ontology version***: Retrieves from the GE the ontology document identified by a given ontology IRI (obtained using the list ontologies service)

and version IRI. To invoke the operation, a GET http request should be sent to http://<ge url location>/ontology-registry/ontologies/<ontology IRI>/<version IRI>.

- ***Get ontology***: Similar to Get ontology version. It retrieves from the GE the latest version of the ontology document identified by a given ontology IRI (obtained using the list ontologies service). To invoke the operation, a GET http request should be sent to http://<ge url location>/ontology-registry/ontologies/<ontology IRI>.
- ***Delete ontology version***: Removes from the GE the ontology document identified by a given ontology IRI (obtained using the list ontologies service) and version IRI. To invoke the operation, a DELETE http request should be sent to http://<ge url location>/ontology-registry/ontologies/<ontology IRI>/<version IRI>.
- ***Delete ontology***: Similar to Delete ontology version. It removes from the GE the latest version of the ontology document identified by a given ontology IRI (obtained using the list ontologies service). To invoke the operation, a DELETE http request will be sent to http://<ge url location>/ontology-registry/ontologies/<ontology IRI>.
- ***Upload ontology version***: Uploads to the GE an ontology document and identifies it with a given ontology IRI (obtained using the list ontologies service) and version IRI. To invoke the operation, a PUT http request should be sent to http://<ge url location>/ontology-registry/ontologies/<ontology IRI>/<version IRI> with an file attachement including the ontology RDF/XML serialization.
- ***Upload ontology***: Similar to Upload ontology version. Uploads an ontology document to the GE and identifies it with a given ontology IRI and with the latest version IRI available. To invoke the operation, a PUT http request should be sent to http://<ge url location>/ontology-registry/ontologies/<ontology IRI> with an file attachment including the ontology RDF/XML serialization.
- ***Get ontology version metadata***: Retrieves from the GE an ontology document containing the metadata related to an ontology document identified by a given ontology IRI (obtained using the list ontologies service) and version IRI. To invoke the operation, a GET http request should be sent to http://<ge url location>/ontology-registry/metadata/<ontology IRI>/<version IRI>.
- ***Get ontology metadata***: Similar to Get ontology version metadata. It retrieves from the GE an ontology document containing the metadata related to the latest version of the ontology document identified by a given ontology IRI (obtained using the list ontologies service). To invoke the operation, a GET http request should be sent to http://<ge url location>/ontology-registry/metadata/<ontology IRI>.
- ***Delete ontology version metadata***: Removes from the GE the metadata related to an ontology document identified by a given ontology IRI and version IRI. To invoke the operation, a DELETE http request will be sent to http://<ge url location>/ontology-registry/metadata/<ontology IRI>/<version IRI>.
- ***Delete ontology metadata***: Similar to Delete ontology version metadata. Removes from the GE the metadata related to the latest version of the ontology document identified by a given ontology IRI. To invoke the operation,

a DELETE http request should be sent to http://<ge url location>/ontology-registry/metadata/<ontology IRI>.

- ***Upload ontology version metadata***: Uploads to the GE an ontology document containing metadata related to an ontology document identified by a given ontology IRI (obtained using the list ontologies service) and version IRI. To invoke the operation, a PUT http request should be sent to http://<ge url location>/ontology-registry/metadata/<ontology IRI>/<version IRI> with an file attachement including the metadata RDF/XML serialization. Metadata uploaded must complain to OMV (Ontology metadata vocabulary).
- ***Upload ontology metadata***: Similar to Upload ontology version metadata. It uploads to the GE an ontology document containing metadata related to the latest version of an ontology document identified by a given IRI (obtained using the list ontologies service). To invoke the operation, a PUT http request should be sent to http://<ge url location>/ontology-registry/metadata/<ontology IRI> with an file attachement including the metadata RDF/XML serialization. Metadata uploaded must complain to OMV (Ontology metadata vocabulary).

- ***Managing Rest API***.
  - ***List ontologies***: Retrieves an XML document containing the list of ontology documents and their versions loaded into the GE. To invoke the operation, a GET http request should be sent to http://<ge url location>/ontology-registry/mgm/list. As a result, an xml encoding the requested information will be sent as response.
  - ***List ontology versions***: Similar to List ontologies. Retrieves an XML document containing the versions of an ontology document identified by a given ontology IRI loaded into the GE. To invoke the operation, a GET http request should be sent to http://<ge url location>/ontology-registry/mgm/<ontology IRI>. As a result, an xml encoding the requested information will be sent as response.

All methods described can be invoked by mean of regular HTTP requests either using a web browser (for those ones who rely on GET requests) or by a programmatic APIs such as Jersey. The query and subscription modules will be provided in the next releases of the Semantic Web Application Support GE.

17.5.3 Frontend Functionality

SWAS Engineering Environment functionality is based on the functionality provided by the baseline asset NeOn Toolkit. The NeOn Toolkit is a state-of-the-art, eclipse based, open source multi-platform ontology engineering environment, which provides comprehensive support for the ontology engineering life-cycle. Due to its nature, it wouldn't be possible to describe all SWAS Engineering Environment provided functionality in a service like manner. Anyway, an overview of the functionality required for the SWAS Engineering Environment is now introduced. To try to better describe functionality, some screenshots from baseline asset NeOnt Toolkit will be used in this section. Figure SWAS-6 presents an overview of NeOn Toolkit GUI.

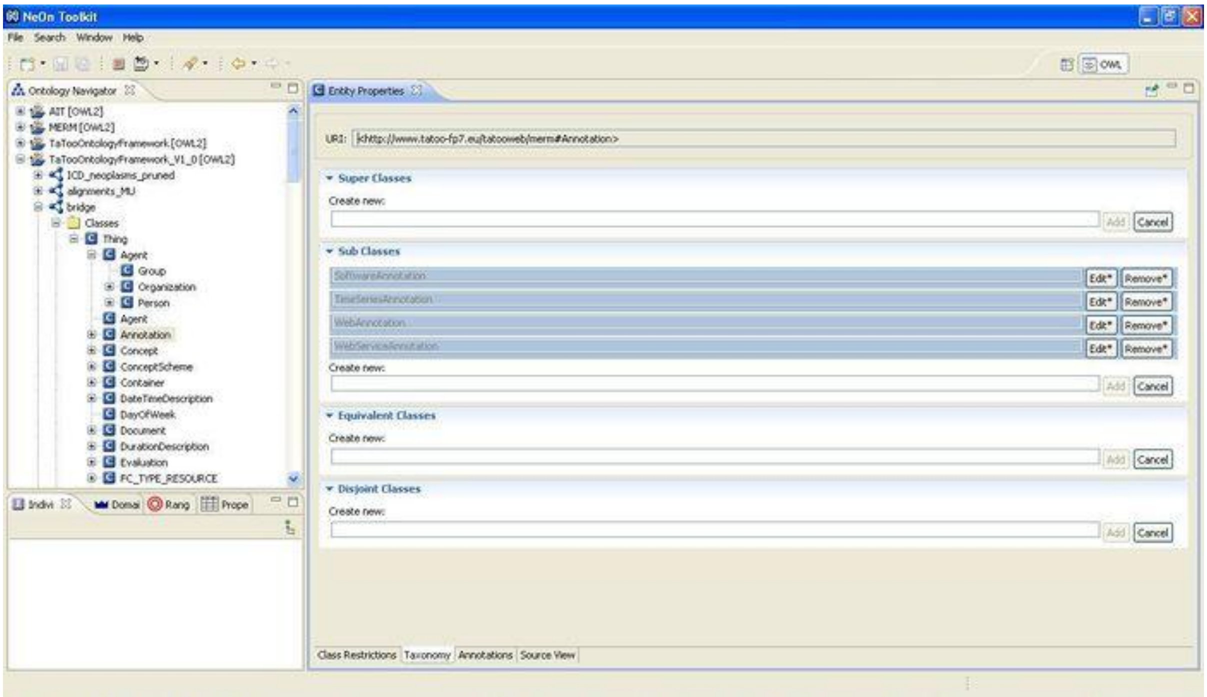


Figure SWAS-6: NeOn Toolkit main window

SWAS GE Engineering Environment will follow and take advantage some of the paradigms introduced by [Eclipse], one of the leading development environments, including:

- Using workspaces, projects, folders and files as containers to organize and store development artifacts.
- Using workbench, editors, views and perspectives to provide functionality to the user by means of GUI.

Therefore, most of the functionality provided by SWAS Engineering Environment is provided as editors, views and perspectives. Figure SWAS-7 presents the Ontology navigation perspective.

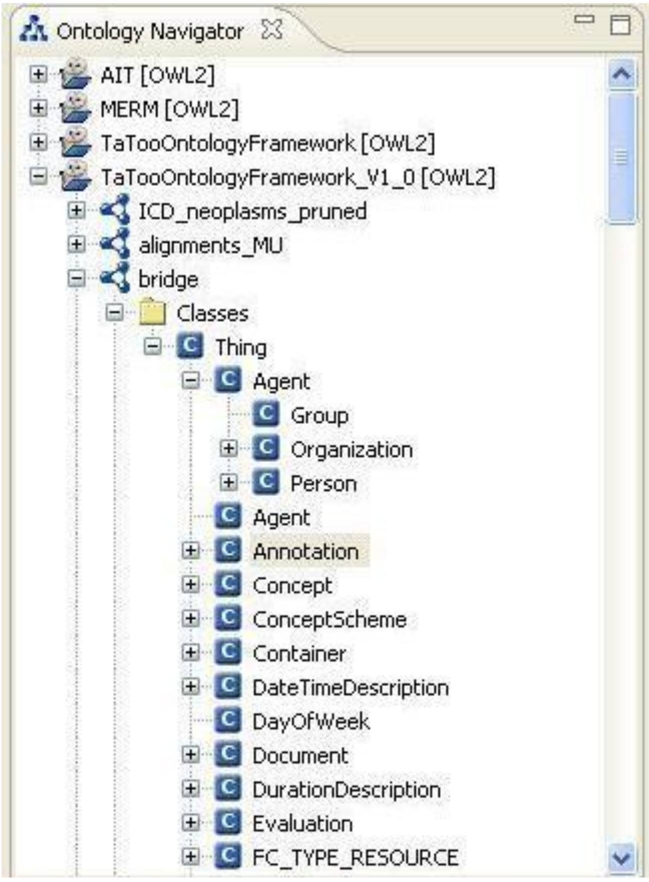


Figure SWAS-7: Ontology navigation perspective

Under this perspective users are able to manage their projects and ontologies, creating or removing projects, loading or creating new ontologies, etc. In the scope of a given ontology, users are able to manage (adding, removing, etc) main ontology contents such as classes, object properties and data properties. Once selected, ontology contents can be edited by means of a proper editor. Figure SWAS-8 presents the class editor.

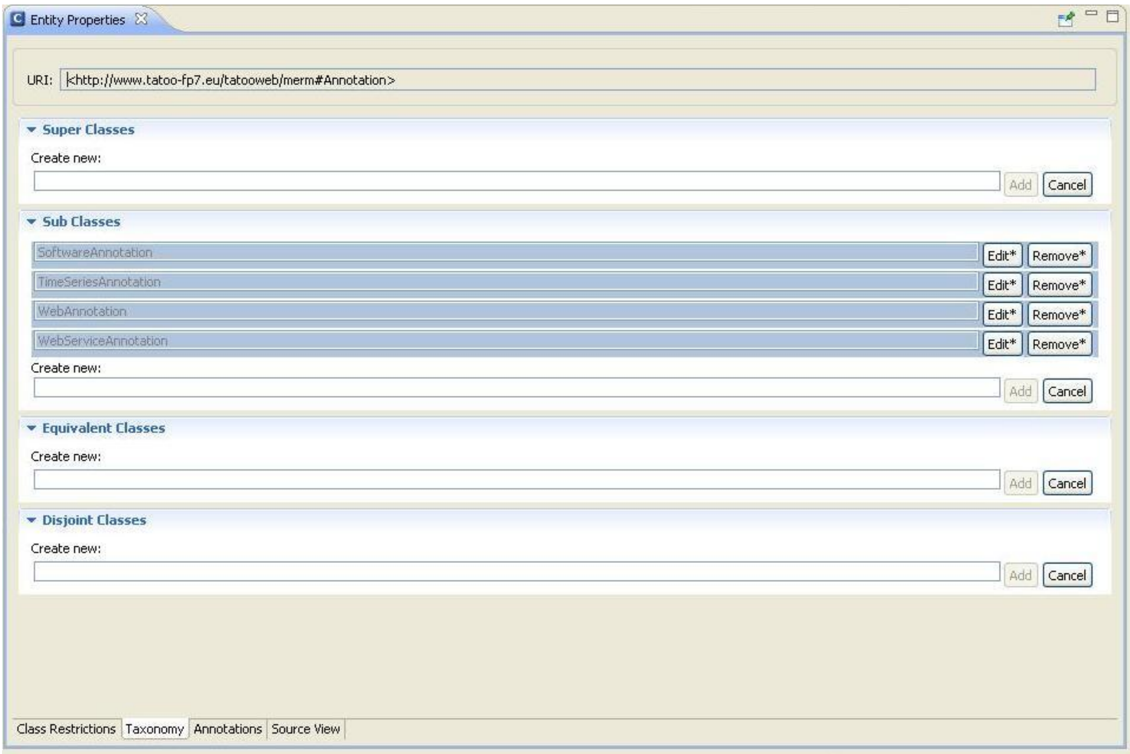


Figure SWAS-8: Class editor

Class editor is composed of four tabs:

- Class restrictions tab that allow the user to modify restrictions applicable to the class.
- Taxonomy tabs that allow the user to modify the class ancestors, successors or siblings.
- Annotation tab that allows the user to annotate the class con textual descriptions.
- Source tab that presents the user the OWL code generated for the described class.

Data property and object property editor provide similar functionality for data and object properties. Finally, views present additional information about the items selected in the ontology navigation perspective. Figure SWAS-9 presents the range view.

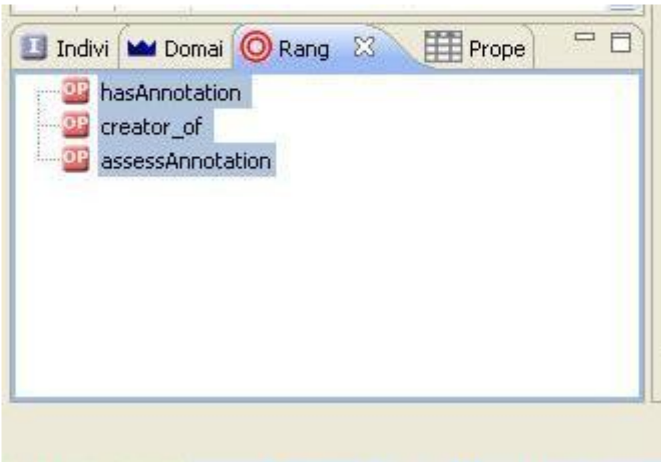


Figure SWAS-9: Range view

Range view presents for each class, the set of object properties that has the selected class as range. As mentioned in Semantic Application Support GE Architecture section, Engineering Environment functionality can be extended by means of plug-ins. Nowadays there are more than 30 active plug-ins for NeOn Toolkit covering a wide range of functionality covering several steps of the NeOn Methodology. Some of this plug-ins functionality may inspire Engineering Environment functionality in the future if needed.

17.6 Design Principles

The main goal of the Semantic Web Application Enabler is to provide a framework for ontology engineers and developers of semantically-enabled applications offering RDF/OWL management, storage and retrieval capabilities. This goal will be achieved by providing an infrastructure for metadata publication, retrieval and subscription that meets industry requirements like scalability, distribution and security, plus a set of tools for infrastructure and metadata-data management, supporting most adopted methodologies and best practices.

The Semantic Web Application enabler is based on the following design principles:

- Support standards: Support for RDF/OWL, the most common standards used in Semantic Web applications.
- Methodological approach: GE is strongly influenced by methodological approaches, so it will adopt and support, as far as possible, most adopted methodologies to achieve its goals.
- Semantic repository features: Provide high-level common features valid for most of the existing solutions in the semantic web in terms of RDF / OWL storage and inference functionalities.
- Ontology management: The enabler will provide an ontology registry and the API to control it, including some high-level ontology management functionalities.
- Knowledge Base management: The enabler will provide a knowledge base registry and the API to control it, including some high level knowledge base management functionalities.
- Extensibility: The most important part of the architecture design of the enabler is to define interfaces that allow the extensibility of the system. Where applicable the design should also be modular, to facilitate future extensions and improvements. The reference implementations should comply with this common design.

In the scope of FI-WARE first release, the work in the Semantic Application Support GE enabler has been mostly related to the ontology registry component. Some decisions regarding this component design has been taken, including: selection of an ontology metadata format, definition of a format for ontology identifiers, definition of an interface for exposing ontology registry functionality and decision on how to storage ontologies.

In order to provide advanced ontology management functionalities, ontologies should be annotated with extended metadata. In order to do so, the selection of a suitable ontology metadata format is needed. In this case, the Ontology Metadata Vocabulary [OMV] has been selected. Some of its key features are:

- OWL-2 ontology developed following NeOn Methodology by consortium members.
- Designed to meet NeOn Methodology reusability use case requirements.
- Extensible, reusable, accessible and interoperable.

OMV describes some metadata regarding ontologies that should be provided by users while loading ontologies into the GE. This metadata include information about ontology developers, ontology language, ontologies imported by the ontology, etc. A class diagram showing OMV main classes and attributes can be found in **Figure X**.

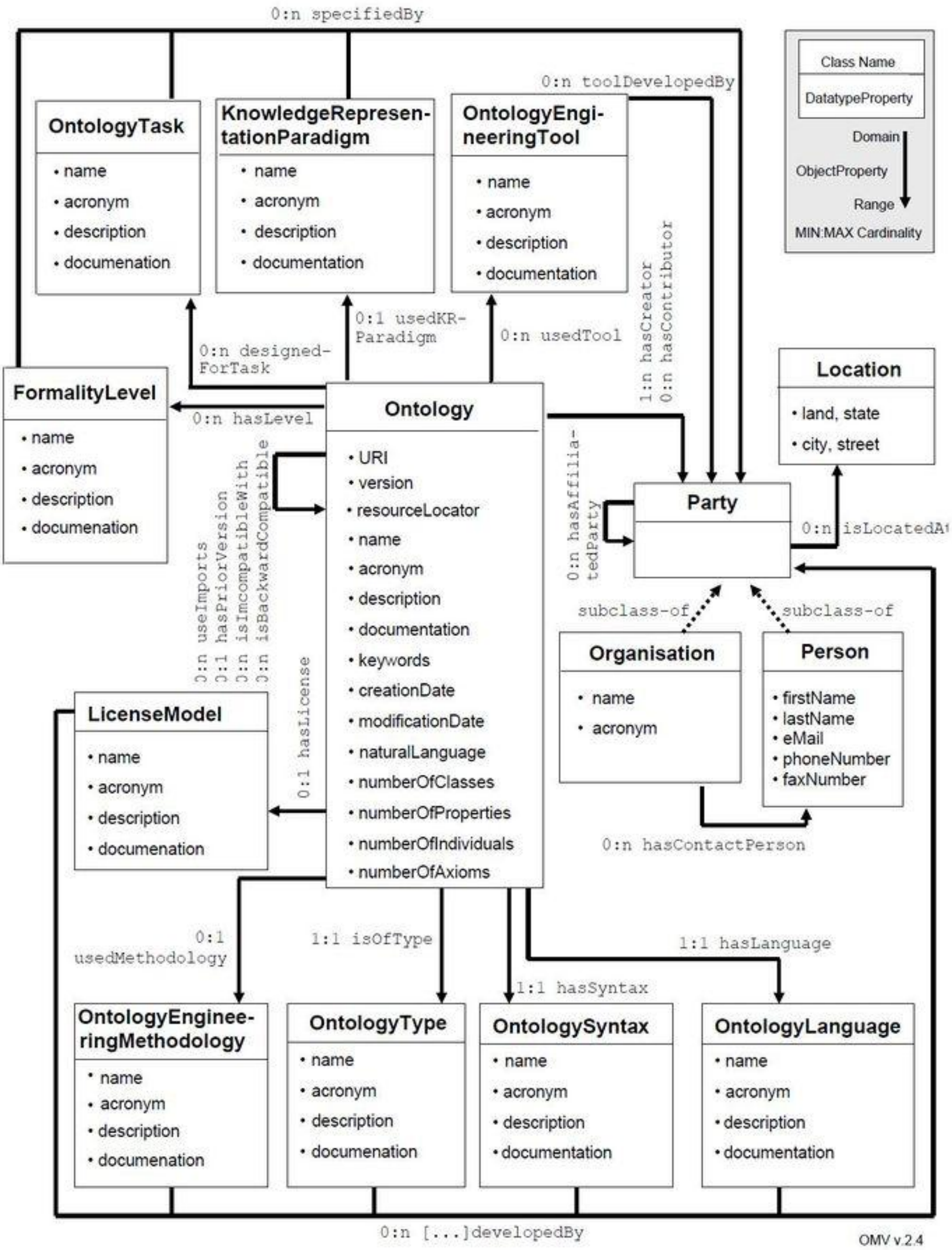


Figure X: Ontology Metadata Vocabulary UML diagram

In order to be stored into the ontology registry, it would be needed to assign to the ontology a unique identifier. Identifying ontologies may look an easy task but it is not even completely tackle even in OWL-2 specification. Taking a look to OWL-2 specification it can be found that:

- *Each ontology may have ontology IRI, which is used to identify an ontology. If an ontology has an ontology IRI, the ontology may additionally have a version IRI, which is used to identify the version of the ontology*
- *The ontology document of an ontology O should be accessible via the IRIs determined by the following rules:*
    - *If O does not contain an ontology IRI (and, consequently, it does not contain a version IRI either), then the ontology document of O may be accessible via any IRI.*
    - *If O contains an ontology IRI OI but no version IRI, then the ontology document of O should be accessible via the IRI OI.*
    - *If O contains an ontology IRI OI and a version IRI VI, then the ontology document of O should be accessible via the IRI VI; furthermore, if O is the current version of the ontology series with the IRI OI, then the ontology document of O should also be accessible via the IRI OI.*

For the sake of the implementation, in the scope of the Semantic Application Support, ontologies must have ontology IRI and version IRI. Ontology IRI must be provided by the user while version IRI may be provided by the GE in some cases. Moreover, ontology documents will be accessible using its ontology IRI plus version IRI, being this last one optional. In case no version IRI is provided, latest version of the ontology identified by the ontology IRI will be provided while accessing.

The Semantic Application Support GE will need to store then two kinds of resources: ontologies and ontology metadata. Having selected OMV as ontology metadata format, ontology metadata would need to be stored into a RDF triple store with OWL capabilities. In case of ontologies, they will be managed as plain text objects and stored in a regular relational data base. This would avoid potential problems of performance while serving ontologies for developers with editing purposes.

Finally, an interface for accessing the ontology registry should be provided. In this case, Semantic Application Support GE will follow [SPRQL Query protocol]: *If a service supports HTTP bindings, it must support the bindings as described in specification. A SPARQL Protocol service may support other interfaces such as SOAP*. In the case of this GE, a RESTful based service will implement the interface of the ontology registry. This interface is described in main interactions section.

17.7 References

[Pariente 2011]	Lobo, T. P., Lopez J. M. F., Sanguino M. A., Yurtsever S., Avellino G., Rizzoli A. E., et al. (2011). A Model for Semantic Annotation of Environmental Resources: The TaToo Semantic Framework. ISESS 2011.
[Suarez-Figueroa 2008]	Suárez-Figueroa, M. C., et al. NeOn D5.4.1. NeOn Methodology for Building Contextualized Ontology Networks. February 2008.
[NeOn-Toolkit]	The NeOn Toolkit, http://neon-toolkit.org/
[Sesame]	Sesame RDF framework, http://www.openrdf.org/
[OWLIM]	OWL Semantic Repository, http://www.ontotext.com/owlim/
[Gruber 1993]	Gruber, T.: A translation approach to portable ontology specifications. Knowledge Acquisition 5, 1993
[OWL-2 RL]	http://www.w3.org/TR/owl-profiles/#OWL_2_RL
[RDF]	Beckett D., RDF/XML Syntax Specification (Revised). W3C Recommendation, 10 February 2004
[RDFs]	Dan Brickley D, Guha R.V., RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, 10 February 2004
[Fernandez 1997]	Fernández-López M., Gómez-Pérez, A. & Juristo, N.: Methontology: from ontological art towards ontological engineering. Proc. Symposium on Ontological Engineering of AAAI, 1997
[OnToKnowledge 2001]	Broekstra, J., Kampman, A., Query Language Definition. On-To-Knowledge (IST-1999-10132), 2001
[DILIGENT 2004]	Pinto H.S., Tempich C., Staab S., Sure Y.: Diligent: Towards a fine-grained methodology for distributed, loosely-controlled and evolving engineering of ontologies. In de M´antaras L.R., Saitta L., editors, Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004), August 22nd - 27th, pages 393–397, Valencia, Spain, AUG 2004. IOS Press.
[OMV]	Jens Hartmann, Raúl Palma, York Sure, María del Carmen Suárez-Figueroa, Peter Haase, Asunción Gómez-Pérez, Rudi Studer: Ontology Metadata Vocabulary and Applications. OTM Workshops 2005: 906-915
[SPRQL Query protocol]	http://www.w3.org/TR/rdf-sparql-protocol/
[Eclipse]	http://www.eclipse.org/

18 Internet of Things (IoT) Services Enablement Architecture

You can find the content of this chapter as well in the wiki of fi-ware.

18.1 Introduction

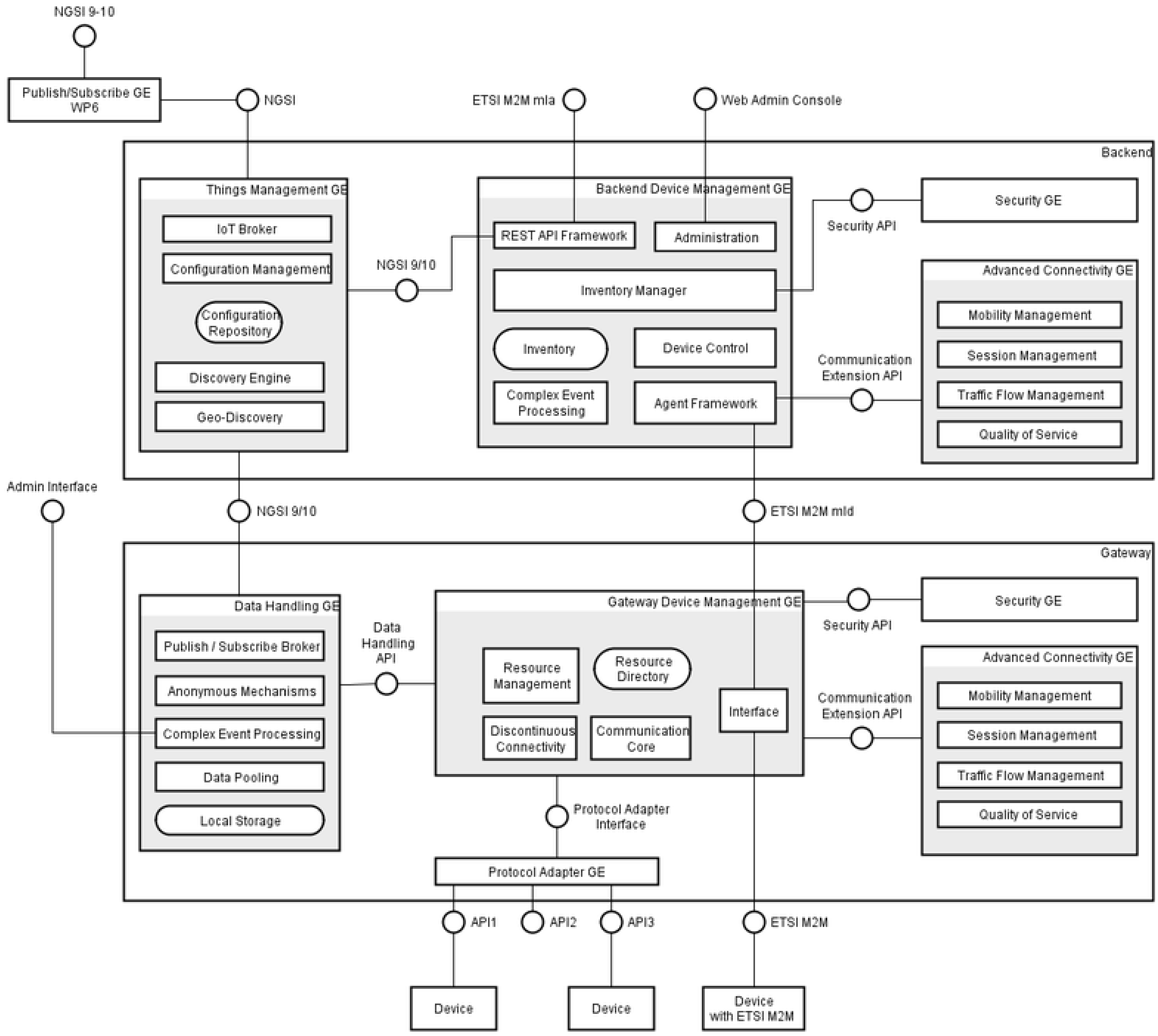
FI-WARE will build the relevant Generic Enablers for Internet of Things Service Enablement, in order for things to become citizens of the Internet – available, searchable, accessible, and usable – and for FI services to create value from real-world interaction enabled by the ubiquity of heterogeneous and resource-constrained devices.

Nota Bene: For the reader, we are using in the following chapters the same vocabulary than in the FI-Ware Product Vision chapter

- **Thing.** Physical object, living organism, person or concept interesting from the perspective of an application.
- **Device.** Hardware entity, component or system that either measures properties of a thing/group of things or influences the properties of a thing/group of things or both measures/influences. Sensors and actuators are devices.
- **IoT Gateway.** A device hosting a number of features of one or several Generic Enablers of the IoT Service Enablement. It is usually located at proximity of the devices to be connected.
- **IoT Resource.** Computational elements (software) that provide the technical means to perform sensing and/or actuation on the device. The resource is usually hosted on the device.

18.2 Architecture Overview

The deployment of the architecture of the IoT Service Enablement chapter is typically distributed across a large number of Devices, several Gateways and the Backend. The Generic Enablers described in this chapter, shown in the figure below, implement functionalities distributed across IoT resources hosted by devices, IoT Gateways and in the IoT Backend.



Architecture Diagram: https://forge.fi-ware.eu/scmrepos/svn/iot/trunk/documents/FMC/IoT-Streamlined.graphml

18.2.1 Device and IoT Resource

A device is a hardware entity, component or system that either measures properties of a thing/group of things or influences the properties of a thing/group of things or both measures/influences. Sensors and actuators are devices. Devices can further be categorized into IoT compliant (i.e., devices with the full-blown FI-WARE capabilities and supporting the standard ETSI M2M interface) and non-compliant (legacy devices with propietary protocols).

IoT Resources are computational elements (software) that provide the technical means to perform sensing and/or actuation on the device. The resource is usually hosted on the device.

18.2.2 Gateway

A gateway is providing inter-networking and protocol conversion functionalities between devices and the IoT backend. It is usually located at proximity of the devices to be connected. An example of an IoT gateway is a home gateway that may represent an aggregation point for all the sensors/actuators inside a smart home. The IoT gateway will support all the IoT backend features, taking into consideration the local constraints of gateway devices such as the available computing, power, storage and energy consumption. Gateways are connected northbound to the backend via IP connectivity and southbound to

- IoT compliant devices without IP connectivity
- Legacy devices that needs protocol conversion

As IP devices will now appear on the market, the gateway will also be able to manage some of them using IETF Core CoaP protocol but one of the main role of the gateway is to bridge different technologies with IP connectivity. The second main role is deployment of smart services as close as possible of the things to enphazise smart applications development.

18.2.3 Backend

The backend provides management functionalities for the devices and IoT domain-specific support for the applications. It supports access at both IoT resource and thing-level. The backend can be connected southbound to gateways and/or IoT compliant devices (devices that will implement the standardised interface i.e. ETSI M2M).

Current developments are focusing on Backend and Gateway interactions.

18.3 Basic Concepts

This section elaborates on those basic concepts that need to be explained in order to understand the multiple Generic Enabler descriptions.

18.3.1 Interface Abstraction Levels

The resources and devices may be addressed on different abstraction levels. This section explains these levels.

18.3.1.1 *Device level*

A device is a hardware unit having the capability to either perform a measurement or an actuation. Device-level data is the raw data provided by the device. Management functionality like sensor calibration, firmware updates, and battery status monitoring is also taking place on the device level.

18.3.1.2 *Resource level*

A resource is a computational element providing access to sensor/actuator devices. An information model for the description of resources can include context data like location, accuracy, status information, etc. Resource level data consists not only of the measured data, but also context information like the data type, a time stamp, accuracy of measurement, and the sensor by which the measurement has been performed. Resources can be addressed using a uniform addressing scheme.

18.3.1.3 *Thing level*

A thing can be any object, person, or place in the real world. Things are represented as virtual things having a type and several attributes. Sensors can be modelled as things, but other real-world objects like rooms, persons, etc. can be modelled as things as well. So thing level data consists of descriptions of things and their attributes, while information on how the data has been obtained might be contained as meta data, but is in general out of scope.

18.3.2 ETSI Machine-to-Machine communications

ETSI Machine-to-Machine communications is an application agnostic standard which is supported by more than hundred companies around the world and has now reached a stable version for reference implementation. This standard deals with connectivity concerns over classical communication networks.

It contains an overall end to end M2M architecture, identifies the functional entities and the related reference points. The ETSI M2M specifications are available at the ETSI website.

The high-level functional architecture consists of two main parts, the Network and Applications domain (Network) and the M2M Device Domain containing M2M Gateways and M2M Devices. M2M Devices are basically devices that run applications using the service capabilities defined by the M2M standard. M2M Gateways are appliances using service capabilities to ensure Devices interworking and interconnection to the Network and Application Domain. FI-Ware gateways will host more services than ETSI-M2M gateways especially defining several levls of gateways hosting data handling capabilities.

- A Service Capability is a central concept. It provides functions that are to be shared by different applications, expose functions through a set of open interfaces as well as simplify and optimise application development and deployment through hiding of network characteristics.
- The standard defines three reference points: mId between the device/gateway and the M2M core, mIa between the M2M core and applications and dIa between the device and gateway. The Backend Device Management GE exposes the mIa interface northbound. All three uses a resource-oriented design with obvious RESTful binding over HTTP. ETSI M2M is applicable to a wide range of network technology, so when specific potential bindings are defined, these bindings should not limit the applicability to other networks (e.g., using different protocols).

References:

- ETSI M2M Architecture Overview
- ETSI M2M Specifications rel. 1

18.3.3 OMA NGSI – Context Management

A thing which provides some data in different situation from the point of view of the application using these data can be described as a "context element", providing data which describe a context. This approach is relevant in the FI-Ware architecture which targets all type of things, by extension several "context elements".

The OMA NGSI Context Management component provides the NGSI-9 and NGSI-10 interfaces to manage Context Information about Context Entities. The purpose of NGSI-9 is to exchange information about the availability of Context Information, while NGSI-10 is desiged for exchanging the Context Information itself.

A Context Entity is any principal and object, which has a state. This state can be described using Context Information. Context Entities could be users, devices, places, buildings, and many other (including virtual objects). Context Information is any volatile or persistent information, which describes a state of a Context Entity. Context Information can be measured by sensors, manually set by humans, derived from operations on handsets or terminals, inferred from other information, or requested from databases.

References:

- NGSI Context Management - Open Mobile Alliance V1.0 Aug 3, 2010
- NGSI-9/NGSI-10 Information Model
- Context Information interface OMA NGSI-10

18.3.4 Interfaces to other Chapters

18.3.4.1 *I2ND*

Interfaces to Networks and Devices architecture

Things will be connected to communication networks and could be hosted into classical mobile devices connected to telecommunication networks. These networks will provide in a near future new services which will be useful for the Internet of Things.

An information exchange is expected to happen between S3C and IoT GEs, this interface exchanges messages in XML like format (under specification). Device Management interfaces are also expected between CDI and IoT GEs. The interface specification is currently under definition.

18.3.4.2 *Data*

An interaction to exchange information is expected between Things Management GE and Publish/Subscribe Broker GEs. This interface is NGSI-9/NGSI-10-based.

18.3.4.3 *Security*

Security GE drawn in the figure belongs to the security chapter and interfaces to the Device Management GE.

18.4 Architecture Description

18.4.1 Backend

- FIWARE.ArchitectureDescription.IoT.Backend.ThingsManagement
- FIWARE.ArchitectureDescription.IoT.Backend.DeviceManagement

Out of scope for the first release:

- FIWARE.ArchitectureDescription.IoT.Backend.AdvancedConnectivity
- FIWARE.ArchitectureDescription.IoT.Backend.Security

18.4.2 Gateway

Gateway's Generic Enablers will be introduced in two updates of the FIWARE testbed in July and September 2012.

- FIWARE.ArchitectureDescription.IoT.Gateway.DeviceManagement
- FIWARE.ArchitectureDescription.IoT.Gateway.DataHandling
- FIWARE.ArchitectureDescription.IoT.Gateway.ProtocolAdapter

Out of scope for the first release:

- FIWARE.ArchitectureDescription.IoT.Gateway.AdvancedConnectivity
- FIWARE.ArchitectureDescription.IoT.Gateway.Security

19 FIWARE ArchitectureDescription IoT Backend ThingsManagement

You can find the content of this chapter as well in the wiki of fi-ware.

19.1 Copyright

19.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

19.3 Overview

19.3.1 Data model outline

The IoT Things Management GE (TM GE for short) is the part of the IoT Backend which is responsible for Thing-level interaction. The underlying data model of this GE is based on the OMA NGSI Context Management Information Model which relies on the concept of context entities, which are generic entities whose state is described based on values of attributes and associated metadata. In the context of IoT, context entities and context entity attributes can be used to model IoT resources and the variables they measure, respectively, but also - and more importantly - arbitrary physical objects (Things) like rooms, persons, etc. and their attributes like temperature, geo-location, etc.

19.3.2 Functionality outline

The TM GE is the point of contact for accessing information about entities and their attributes. The northbound interface supports one-time queries and continuous subscriptions for that kind of context information. Applications or other GEs can further receive unsolicited information updates from the TM GE about both the availability of context information and actual context information. In the FI-WARE architecture, the Publish/Subscribe Broker GE takes the role of the Application consuming context information from the TM GE.

Using the FI-WARE NGSI-9 interface that the TM GE provides, applications will be able to register, query for and subscribe to updates on context availability information, that is:

● Information about IoT resources and the variables they measure

● Information about Things and their attributes

● Information about Associations establishing how attributes of Things can be derived from attributes of other Things or from variables measured by IoT resources

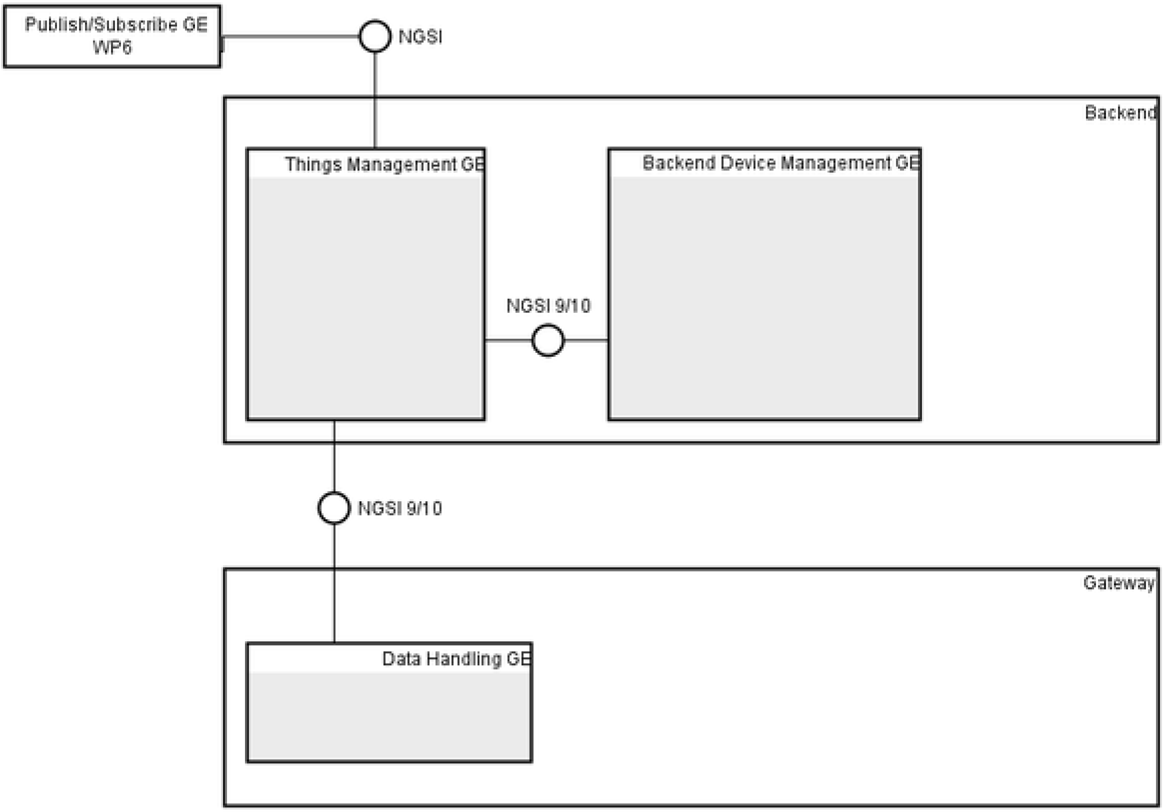
The TM GE implements the subset of the FI-WARE NGSI interfaces required to support integration with the FI-WARE Publish/Subscribe Broker GE so that context information about Things becomes accessible to applications. Actually, using the FI-WARE NGSI-10 interface that the Publish/Subscribe Broker GE provides, applications will be able to:

● Query about attributes of Things

● Subscribe about updates on attributes of Things

Note that the Publish/Subscribe Broker GE may manage context information not necessarily provided by the TM GE, therefore linked to the Internet of Things, but gathered from other parts of the application.

The context information and context availability information provided by the TM GE is either (a) forwarded from IoT Agents exporting FI-WARE NGSI interfaces, or (b) derived from lower-level context information inside the TM GE. In case (a), IoT Agents can be both the Data Handling GE in IoT Gateways, or the Backend Device Management GE. It is also possible that the context information is provided by other IoT Backend systems. In case (b), the TM GE applies basic mapping rules derived from associations registered through the FI-WARE NGSI-9 interface that the TM GE exports. It may also apply more complex rules based on inference engine capabilities it will support.



Things Management GE and the interacting components

19.4 Basic Concepts

The Things Management GE is based on the OMA NGSI context data model.

19.4.1 FI-WARE NGSI

The TM GE the FI-WARE NGSI Context Management specifications (FI-WARE NGSI specifications for short).

FI-WARE NGSI Context Management specifications are based on the NGSI Context Management specifications defined by OMA (Open Mobile Alliance). They take the form of a RESTful binding specification of the two context management interfaces defined in the OMA NGSI Context Management specifications, namely NGSI-9 and NGSI-10 (see FI-WARE NGSI Open RESTful API Specification (PRELIMINARY)).They also solve some ambiguities in the OMA specs and extend them when necessary to implement the FI-WARE Vision.

You can visit the FI-WARE NGSI Context Management tutorial to learn the main concepts underlying FI-WARE NGSI Context Management specifications.

19.4.2 Associations in FI-WARE NGSI-9

19.4.2.1 *Background*

OMA NGSI-9 is an interface for exchanging information on the availability of context information. The central information container of NGSI-9 is a data structure called ContextRegistration. Each ContextRegistration instance contains a list of entities, a list of attributes, the URI of an application where context information about these entities/attributes is available, and a list of metadata.

In NGSI-9 is implicitly assumed that the providing application exposes an NGSI-10 interface, where the context information can be queried. It is also implicitly assumed that the context information is provided using the same entity/attribute combinations. For example, a ContextRegistration instance containing the entity "Car_2", attribute "speed", and URI "http://cars.info" represents the information that the speed of "Car_2" can be queried at "http://cars.info" by an NGSI-10 queryContext operation asking for attribute "speed" of entity "Car_2".

19.4.2.2 *Enhanced Associations*

In the following paragraphs we describe a generalization of this principle. We introduce a special type of metadata which modifies the query that has to be invoked at the providing application in order to obtain the desired information. In NGSI, each piece of metadata has a name, a type and a value. The metadata introduced at this point has name "knownAs" and type "QueryContextRequest". The metadata value represents the necessary query message for the providing application: When invoking the query context operation on the providing application using this message, the returned result will be the context information described in the ContextRegistration structure.

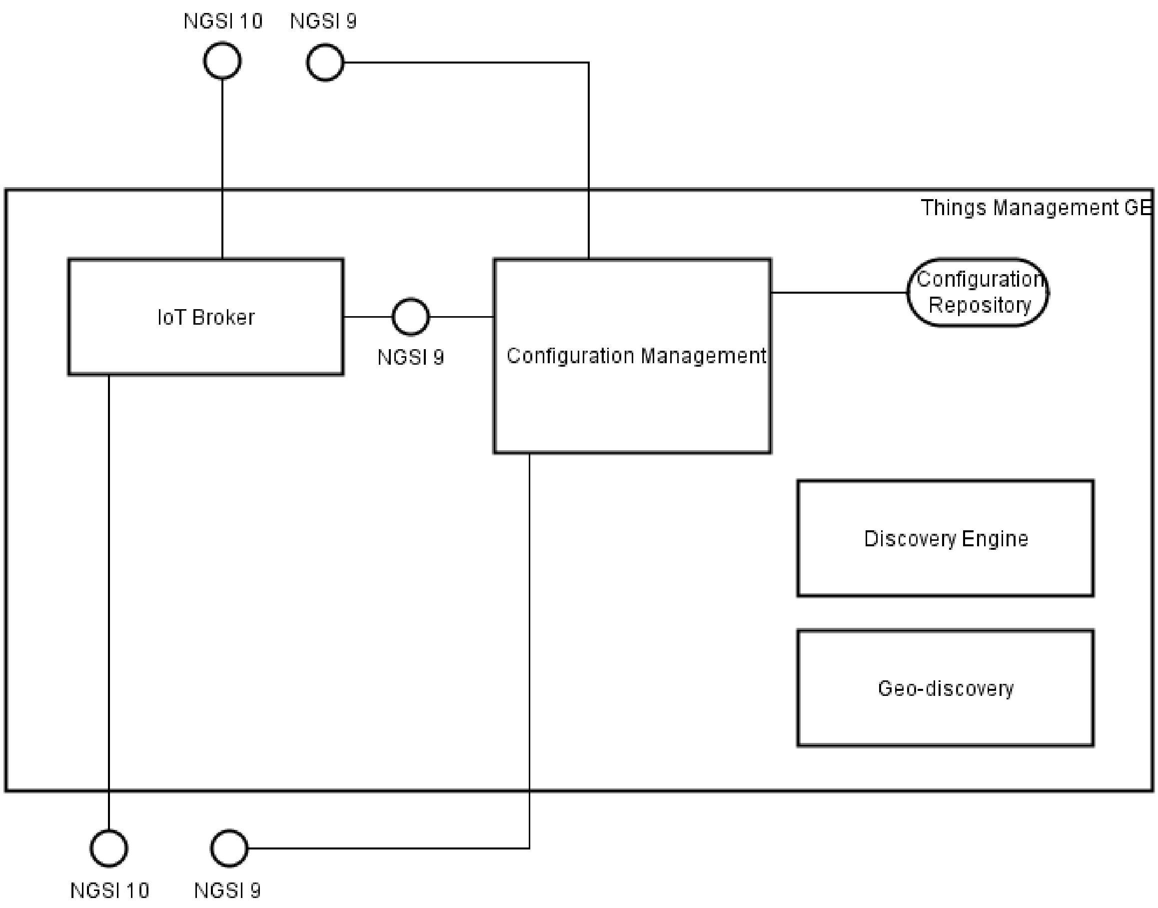
Before formally defining this special type of ContextRegistration instances, let us walk through an example. Assume that a ContextRegistration contains one entity ID "Car_2", one attribute "speed", providingApplication is "http://cars.info", an there is a piece of metadata as described above. The metadata value is an instance of QueryContextRequest which contains one entityId "speedometer_38" and one attribute "measurement". The information described by the ContextRegistration instance just described is as follows: in order to find out the speed of Car_2, one has to go to "http://cars.info" and query for the "measurement" of "speedometer_38".

An instance of ContextRegistration as the one just described is called an *Association*. Formally, a ContextRegistration instance is an Association, if

- It contains exactly one entity ID and exactly one attribute. This attribute is not a domain name, and the entity ID is no pattern.
- It contains a piece of metadata with name "knownAs" and type "QueryContextRequest".
- The value of the "knownAs" metadata is an instance of QueryContextRequest which contains exactly one entity ID and one attribute. The entity ID is no pattern, and the attribute is no domain.

19.4.3 TM GE Architecture

This section describes the internal architecture of the FI-WARE TM GE implementation.



IoT TM GE internal architecture

The main components of the TM GE are described in the following sections.

19.4.3.1 ***IoT Broker***

The IoT Broker is the point of contact for accessing Thing-level information via the northbound NGSI-10 interface.

When requests arrive through this interface, the IoT Broker first discovers the IoT resources that can provide the required information. The discovery is typically realized by interacting with the Configuration Management via the NGSI-9 interface. Following the discovery, the IoT Broker queries the discovered information sources through its southbound NGSI-10 interface.. In the FI-WARE Architecture, these information sources are Data Handling GE instances on IoT Gateways or the information of interest is provided by the Backend Device Management GE. Subscriptions through the northbound interface are handled similarly.

The IoT Broker can also receive NGSI-10 updates via the southbound interface. In this case, the received information is forwarded through the northbound interface to a fixed information sink. In the FI-WARE architecture, this information sink is the Publish/Subscribe Broker GE.

19.4.3.2 ***Configuration Management***

The Configuration Management component implements a context information registry in which context provider applications can be registered. In addition, components interacting with the Configuration Management can perform discovery operations on that context registration information or subscribe to changes on it.

It detail, the component provides the following functionality;

● Allow IoT Broker discovery and subscription of context information, through the internal NGSI-9 interface exposed by the Configuration Management..

● Announcing the availability of context information in the towards the Publish/Subscribe Broker GE (or any other component supporting the NGSI-9 interface) through the northbound NGSI-9 interface.

The Configuration Management component stores the context information in a Configuration Repository described in next section. This repository is used to answer NGSI-9 request in the exposed interfaces described above.

Receiving registrations from IoT Gateways and the Thing-level Adapter through the NGSI-9 southbound interface and storing this information in the Configuration Repository , thus updating the context information registry.

19.4.3.3 ***Configuration Repository***

The Configuration Repository stores information on the availability of context information and can be accessed through the Configuration Management. When the IoT Broker receives a context query, its first step is to use the queryContextAvailability operation on the Configuration Management in order to find out where the desired context information can be found. The Configuration Management returns a number of ContextRegistration instances, which can possibly be associations.

By this approach, the TM GE can maintain higher-level context information that is not available in the IoT Gateways or Devices. For example, a Gateway might not know the concept of cars, but only maintains a list of sensors and their measurements. The information about which sensors provides information about which car could be maintained only by the TM GE (moreover, the relationship between sensors and cars could have been automatically derived by the Discovery Engine component described in next section). Still taking as example the cars, when an application queries the status of a certain car, the TM GE determines what sensors provide that information, queries those sensors' measurements, and returns the measured values to the application.

19.4.3.4 ***Discovery Engine (PRELIMINARY)***

The discovery engine is a component that makes use of machine learning (ML) techniques for resolving object descriptions, whereby objects can refer to Things or Resources. The component uses probabilistic ML techniques to group objects with similar descriptions. Its purpose is to allow automated discovery, ranking and recommendation of objects. The probabilistic machine-learning approach is completely unsupervised. It provides an efficient mechanism for publication and discovery by: · Automatically clustering all the object descriptions within a repository, hence allowing the Discovery Engine to be scalable to large object repositories. · And automating the ranking of results in order of relevance. It is also Interoperable between different types of description technologies as it bases its technique on text analysis, as long as wrappers are provided for a particular description format. With regards to the Discovery Engine’s role in the Things Management GE, it will first retrieve available NGSI-9 descriptions from the repository, which in this case is the Configuration Repository. The Discovery Engine will use the NGSI-9 descriptions to extract the link for a semantic description of the object. This is description is termed as the Advanced Description. The Advanced Description is essentially an RDF description based on W3C SSN and IoT-A IoT models. It will be used for extracting semantic concepts which will be used for training the engine. The discovery engine will also host a repository based on the “Linked IoT Data Platform” for storing the Advanced Descriptions. The Advanced Description will also enable the dynamic binding of spatial, temporal, and thematic associations between Things and Resources.

When upon startup, there are no descriptions; the Discovery Engine will wait until an acceptable number of descriptions have been stored in order to start the training process and defining the clusters. In the case when discovery requests are received by the Configuration Management from the IoT Broker via NGSI-9, the request can be forwarded to the discovery engine to provide a recommended, ranked set of results, especially in the case where a request providing an ID/attribute is not exactly matched with the descriptions in the repository.

19.4.3.5 ***Geo-Discovery (PRELIMINARY)***

Geo-Discovery is an optional component which is specialized to perform fast and robust discovery of all entities (possibly of a certain type) within given geographic scopes. To this end it communicates with the IoT Broker and/or the Configuration Management. Details are still under discussion.

19.5 Additional Concepts

19.5.1 Enhanced Associations by OMA NGSI-9

19.5.1.1 ***Background***

OMA NGSI-9 Normal 0 21 false false false FR ZH-CN AR-SA is an interface for exchanging information on the availability of context information. The central information container of NGSI 9 is a data structure called ContextRegistration. Each ContextRegistration instance contains a list of entities, a list of attributes, the URI of an application where context information about these entities/attributes is available, and a list of metadata.

In NGSI-9 is implicitly assumed that the providing application exposes an NGSI-10 interface, where the context information can be queried. It is also implicitly assumed that the context information is provided using the same entity/attribute combinations. For example, a ContextRegistration instance containing the entity "Car_2", attribute "speed", and URI "http://cars.info" Normal 0 21 false false false FR ZH-CN AR-SA represents the information that the speed of "Car_2" can be queried at Normal 0 21 false false false FR ZH-CN AR-SA "http://cars.info" by an NGSI-10 queryContext operation asking for attribute "speed" of entity "Car_2". .

19.5.1.2 *Enhanced Associations*

In the following paragraphs we describe a generalization of this principle. We introduce a special type of metadata which modifies the query that has to be invoked at the providing application in order to obtain the desired information. In NGSI, each piece of metadata has a name, a type and a value. The metadata introduced at this point has name "knownAs" and type "QueryContextRequest". The metadata value represents the necessary query message for the providing application: When invoking the query context operation on the providing application using this message, the returned result will be the context information described in the ContextRegistration structure.

Before formally defining this special type of ContextRegistration instances, let us walk through an example. Assume that a ContextRegistration contains one entity ID "Car_2", one attribute "speed", providingApplication is "http://cars.info", an there is a piece of metadata as described above. The metadata value is an instance of QueryContextRequest which contains one entityId "speedometer_38" and one attribute "measurement". The information described by the ContextRegistration instance just described is as follows: in order to find out the speed of Car_2, one has to go to "http://cars.info" and query for the "measurement" of "speedometer_38".

An instance of ContextRegistration as the one just described is called an Association. Formally, a ContextRegistration instance is an Association, if

● It contains exactly one entity ID and exactly one attribute. This attribute is not a domain name, and the entity ID is no pattern.

● It contains a piece of metadata with name "knownAs" and type "QueryContextRequest".

● The value of the "knownAs" metadata is an instance of QueryContextRequest which contains exactly one entity ID and one attribute. The entity ID is no pattern, and the attribute is no domain.

19.5.1.3 *Usage in the IoT TM GE*

The Configuration Repository stores information on the availability of context information and is made accessible via the Configuration Management. When the IoT Broker receives a context query, its first step is to use the queryContextAvailability operation on the Configuration Management in order to find out where the desired context information can be found. The Configuration Management returns a number of ContextRegistration instances, which can possibly be associations.

By this approach, the TM GE can maintain higher-level context information that is not available in the IoT Gateways or Devices. For example, a Gateway might not know the concept of cars, but only maintains a list of sensors and their measurements. The information about which of the sensors provides information about which car could be maintained only in the backend, automatically derived by the Discovery Engine. When applications query the status of a certain car, the TM GE will find determine the sensors providing that information and, query for these sensors' measurements, and return the measured values to the application.

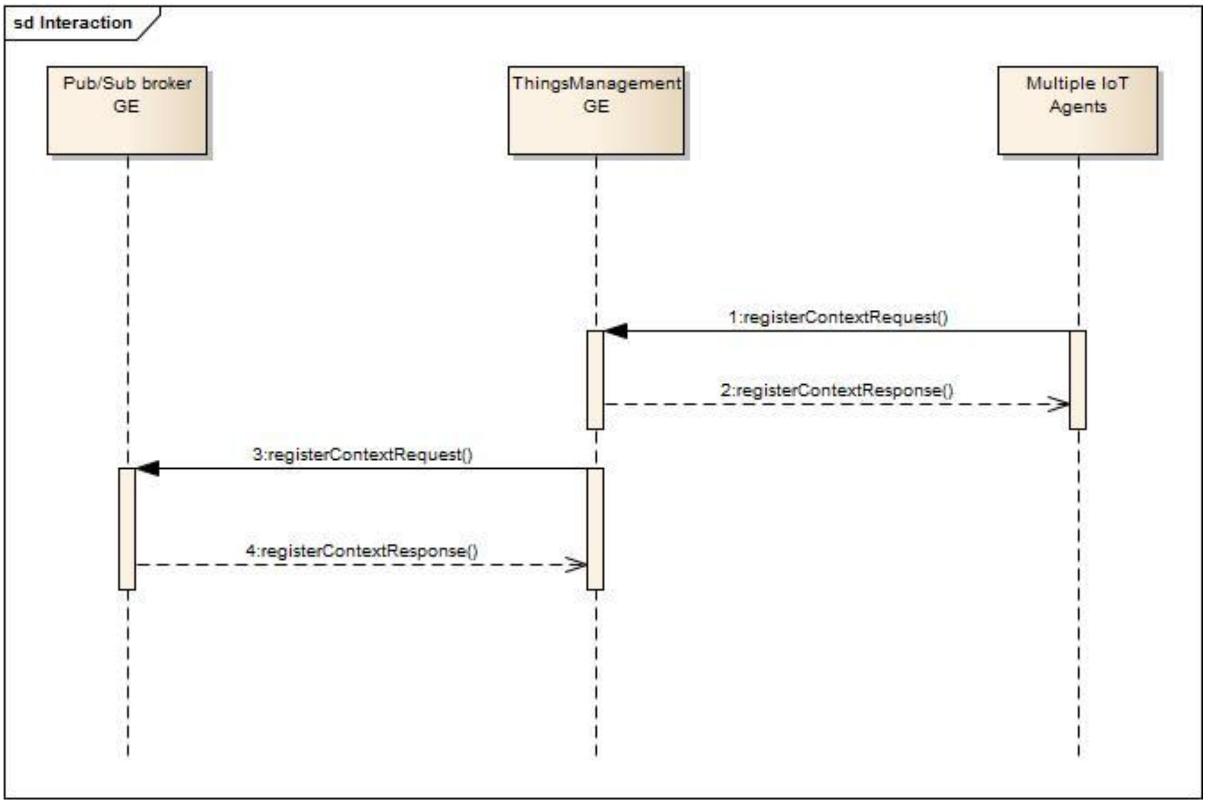
More examples of how associations are used can be found below .

19.6 Main Interactions

The TM GE has a number of interfaces for interacting with applications, users, and other GEs of the FI-WARE architecture. In that context, the GE communicates with the Publish/Subscribe Broker GE via the Northbound Interface and with the IoT Agents on the Southbound Interface. The role of IoT Agent can be played by either the Backend Device Management GE, or by the Gateway Data Handling GE.

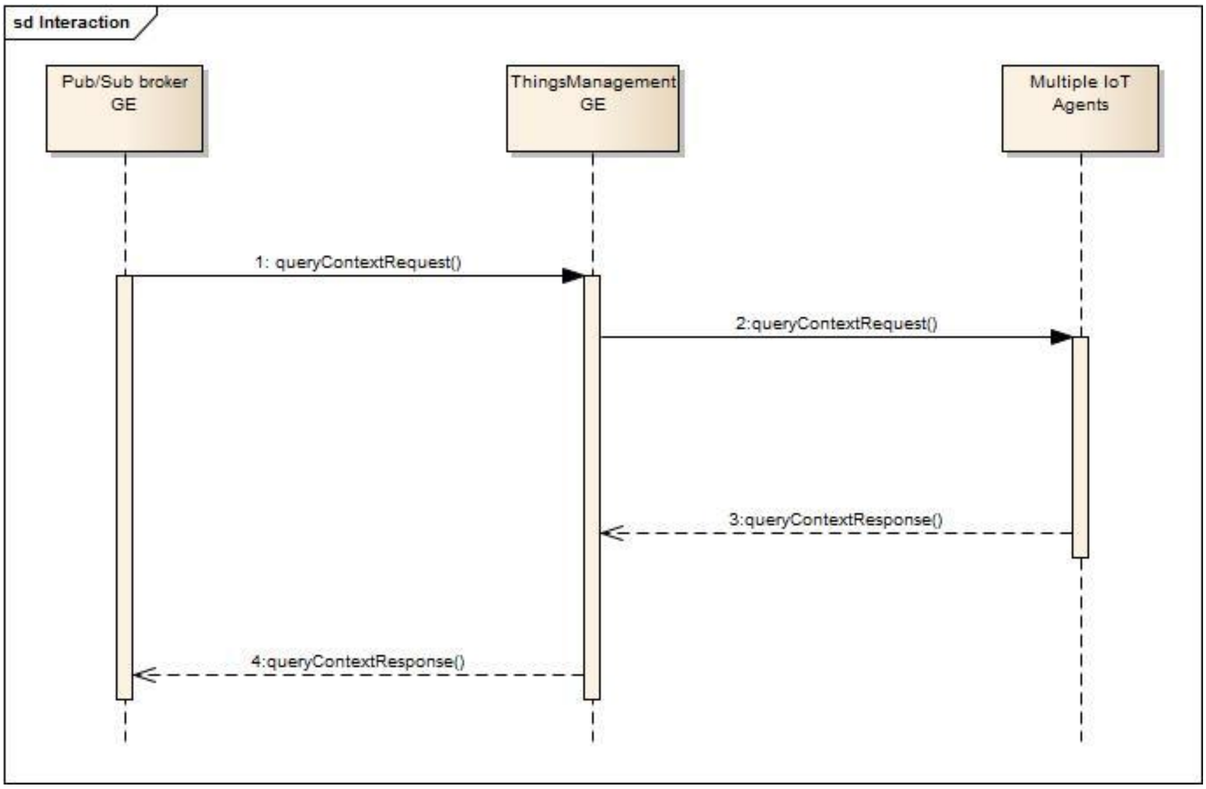
19.6.1 Reception of RegisterContext operations from Agents

In order for their information on Things and/or Resources to be available at the Backend, IoT Agents need to register their information to the TM GE. This is done via the RegisterContext operation. Note that the registration can be performed on various levels of granularity: registering specific Entity/Attribute combinations, registering that information about certain entity type is available, or even in general registering that some entity information is available (without getting more specific) are all possible registrations. Also note that in absence of a Publish/Subscribe GE instance the registration is still stored in the TM GE.



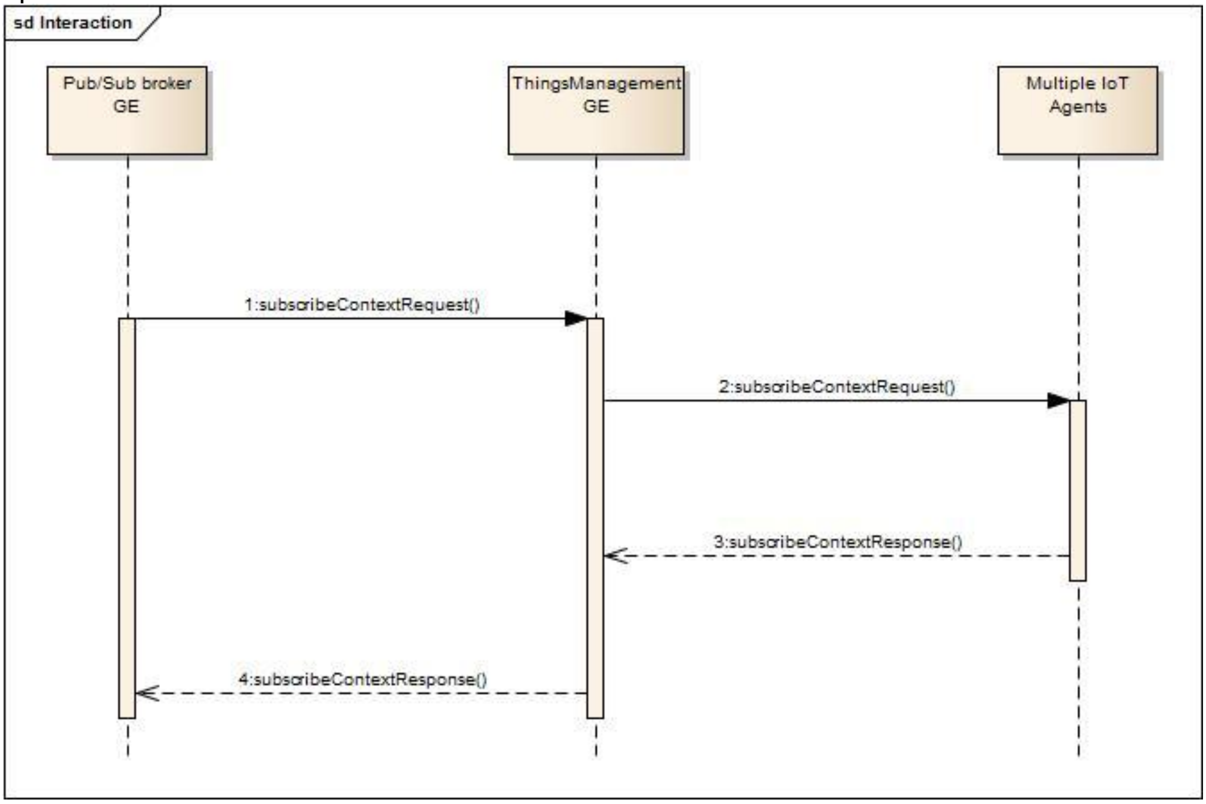
19.6.2 Query Handling

Queries are one-time requests for information. They are realized by the queryContext operation of OMA NGSI-10.



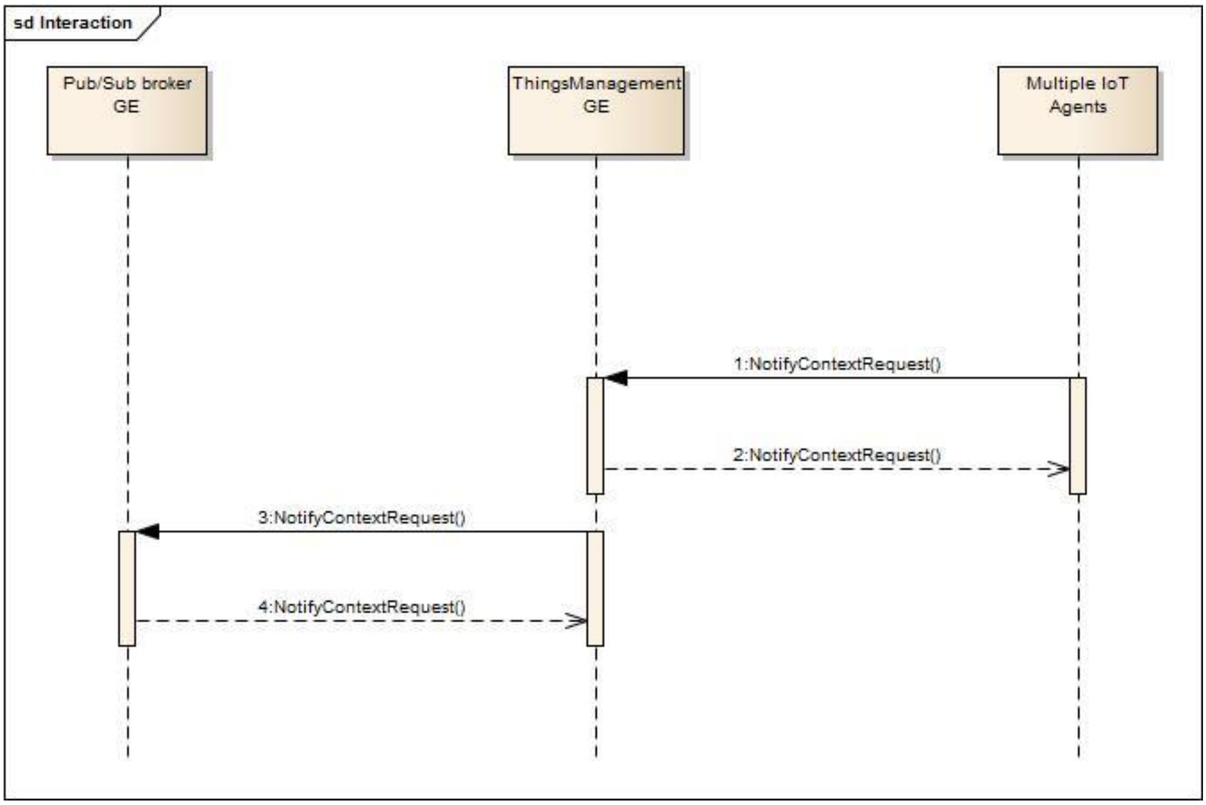
19.6.3 Subscription Handling

Subscriptions are requests for information updates the issuer wishes to receive under conditions that have to be specified in the request message. The picture below shows the interaction diagram for the OMA NGSI-10 subscribeContext operation, but the same interaction pattern applies to the updateContextSubscription and unsubscribeContext operations.


sd Interaction
Pub/Sub broker GE
ThingsManagement GE
Multiple IoT Agents
1:subscribeContextRequest()
2:subscribeContextRequest()
3:subscribeContextResponse()
4:subscribeContextResponse()

19.6.4 Notification

Notifications are the counterpart of subscriptions. A notification is sent whenever the condition for it (that has been specified in the subscription) is satisfied.



20 FIWARE ArchitectureDescription IoT Backend DeviceManagement

You can find the content of this chapter as well in the wiki of fi-ware.

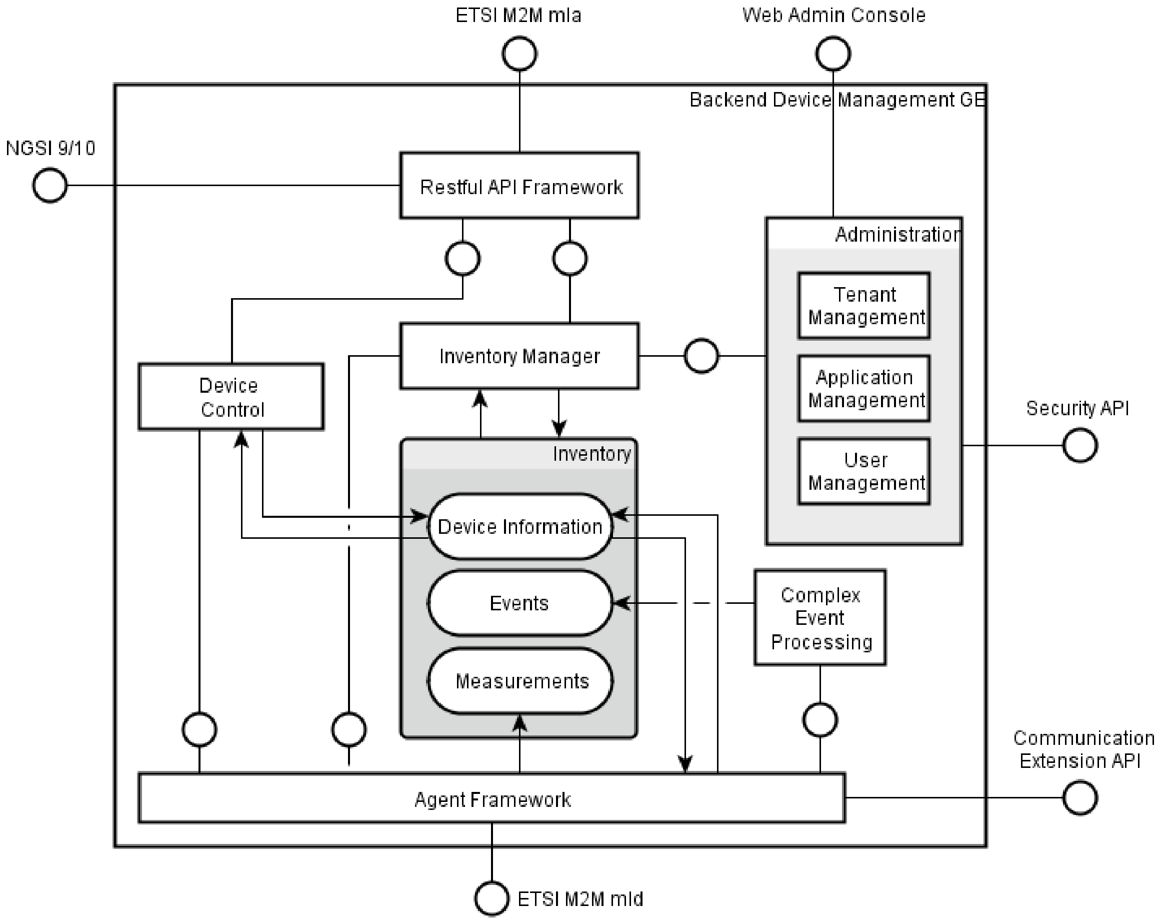
20.1 Copyright

20.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

20.3 Overview

The Backend Device Management GE is the central component for the IoT backend. It provides the resource-level management of remote assets (devices with sensors and/or actuators) as well as core communication capabilities such as basic IP connectivity and management of disconnected devices.



20.3.1 Main Components

20.3.1.1 *Inventory Manager*

Backend applications will want to maintain an inventory of connected M2M devices and their relationship to remote assets (such as rooms, cars or machinery). The Inventory Manager provides the basic business logic for this task allowing the to access data of remote sensors.

Each managed object in the inventory has an own, "global" identifier that is synthetically generated by the Inventory Manager when the object is created. This identifier can be used to reliably reference the object, regardless of, for example, restructuring of networks or replacement of hardware parts.

20.3.1.2 *Inventory*

The Inventory component allows:

- Access data of remote sensors and use remote controls independent of device manufacturer, but still capture manufacturer-specific data where required.
- Capture application- or vertical-specific data.
- Capture tenant-specific data.

This is facilitated through Managed Objects and Fragments. The Inventory component consists of subcomponents for capturing device information, events, readings, alarms.

Device Information

The Device Information stores devices and other assets or business objects known to the Backend Device Management GE, referred to as *Managed Objects*. Each managed object in the inventory has an own, "global" identifier that is synthetically generated by the Backend Device Management GE when the object is created. This identifier can be used to reliably reference the object, regardless of, for example, restructuring of networks or replacement of hardware parts.

Events

Events are used to pass real-time information. Three types of events are distinguished: base events, alarm signals and audit records. A base event signals when something happens. An event could, for example, be sent when a switch is switched on or off. An alarm signals an event that requires action, for example, when a meter has been tampered with or the temperature of a fridge increases above a particular threshold. An audit record stores events that are security relevant and should be stored for auditing. For example, an audit log should be generated when a user logs into a gateway.

Measurements

Measurements represent regularly acquired readings and statistics from sensors. Measurements consist of a time when the measurement was taken, the unique identifiers of the source of the measurement

20.3.1.3 *Device Control*

Devices need to be remote controlled and managed. The main scenarios are the following:

- Device control: Setting a switch, regulating a heating control.
- Device configuration: Setting a tariff table in a smart meter.
- Device operation: Requesting a home security camera to take a picture.
- Device maintenance: Uploading a new firmware binary.

These use cases are implemented by the Device Control component via sending operations to the devices.

20.3.1.4 *Agent Framework*

To shield the backend applications from the diversity of IoT protocols, parameters and network connectivity options, the Backend Device Management GE uses so-called agents. An agent is a function that fulfills three responsibilities for a given vendor and type of devices:

Protocol translation Configuration parameters, readings, events and other information are either send to an agent ("push") or queried by the agent ("poll") through a device-specific protocol on one side. The agent will convert these messages into the protocol that the Backend Device Management GE understands on the other side. It will also receive device control commands from the GE ("switch off that relay") and translate these to whatever protocol the device understands.

The Backend Device Management GE uses a simple and secure reference protocol based on REST (i.e., HTTP) and JSON, which can be used from a wide variety of programming environments down to small embedded systems. To support near-real-time scenarios, the protocol is designed around a "push" model, i.e., data is sent as soon as it is available.

Model transformation Configuration parameters, readings, events asf. all have their device-specific name (and possibly units). An agent for a particular device will transform this device-specific model to the reference model. For example, an electricity meter may provide the main reading as a parameter "Received Wh", so the agent will transform this reading into a reference "Total active energy" in kWh.

Secure remote communication Devices may provide a protocol that is unsuitable for secure remote communication, in particular in public cloud environments. The protocol may only support local networking, it may not pass through firewalls and proxies and it may carry sensitive data over clear text. To overcome such situations, an agent can be co-located to the device and provide a secure, internet-enabled link to the remote device.

20.3.1.5 *Complex Event Processing*

Complex Event Processing is a component responsible for processing events and discovering complex patterns among multiple streams of event data.

20.3.1.6 *Administration*

The Administration component is a set of web-based admin interfaces responsible for Tenant, Application and User Management.

Tenant Management

The Backend Device Management GE supports multiple tenants. Several enterprises, or tenants, share the same instance of the GE. Each tenant has

- A dedicated URL for accessing the instance.
- An own user database storing the tenant's users and their passwords.
- A dedicated storage area keeping the data that is received from the tenant's devices and that is entered by the tenant users. This storage area is, by default, invisible to other tenants on the same instance.
- A set of subscribed backend applications that the tenant can use.

Application Management

The usage of applications directly connected to the backend will typically be a paid service that tenants subscribe to. Hence, these applications are registered with the Backend Device Management GE. This allows the Backend Device Management GE to check subscriptions, make the applications visible in the user interface, monitor them and possibly charge on usage base.

User Management

The Backend Device Management GE uses a standard authentication and authorization model based on realms, users, user groups and authorities. A *realm* is a database of users and user groups that follow the same authentication and authorization policy. A *user* is a person or an external system entitled to access protected resources in the GE. Access is controlled through permissions, or authorities.

The Backend Device Management GE creates a new realm for each tenant to store the users of that tenant. Realms provide own name space for user names, allowing users to keep the names that they are familiar with from their own enterprise IT or other IT systems. There is no conflict between user names – a user "smith" of one particular tenant is different from a user "smith" of another tenant. Each new realm is automatically populated with an initial administrator user who can create further users and user groups, and who can assign permissions to these users and user groups.

20.3.1.7 ***Restful API Framework***

The Backend Device Management GE provides a set of REST interfaces for the applications. Application developers are free to choose the technologies like Java, JQuery, Ruby etc. for the application development.

20.3.2 Basic Concepts

20.3.2.1 ***Managed Objects***

Managed Objects are devices and other assets or business objects known to the Backend Device Management GE. Managed objects could be actual "smart objects" such as smart electricity meters, home automation gateways and GPS devices. They could be supervised assets, such as rooms in which sensors are installed, or cars containing GPS devices. The can also be related business objects, such as households or driving routes.

The following JSON code shows a minimal example of a managed object in the inventory, in this case a simple switch.

```
{
  "id": "47635",
  "type": "com_ge_45609",
  "com_cumulocity_model_control_Relay": {
    "state": "OPEN"
  },
  ...
}
```

An example for another asset stored in the inventory could be a room in which the switch is installed. (Compare the "id" property of the switch with the "managedObject" reference.)

```
{
  "id": "47636",
  "type": "com_resortenergymgmt_model_Room",
  "name": "Sauna",
  "childAssets": {
    "references" : [
      {
        "managedObject": {
          "id": "47635",
          ...
  },
  "com_resortenergymgmt_model_RoomProperty": {
    "size": 56,
    ...
  }
}
```

In general, each managed object consists of

- A unique identifier that can be used to reliably reference the object.
- A type string that defines the most specific type of the object.
- A time stamp of the last update.
- Additional so-called fragments.

20.3.2.2 *Fragments*

Fragments are used to identify capabilities of a managed object. For example, you may want to describe electricity meters from different vendors. Depending on the make of the meter, it may have a relay and it may be able to measure a single phase or three phases. These capabilities are identified by storing a fragment for each of the capabilities as follows:


```
{
  "id": "47635",
  "type": "com_elstermetering_AS220",
  "lastUpdated": "2010-11-13T18:28:36.000Z",
  "com_cumulocity_model_Coordinate": {
    "latitude": 63.2857346747758,
    "longitude": 28.03634548187256
  },
  "com_cumulocity_model_energy_sensor_ThreePhaseElectricitySensor":
{},
  "com_cumulocity_model_control_Relay": {
    "state": "CLOSED"
  }
}
```


In this example, a fragment "com_cumulocity_model_energy_sensor_ThreePhaseElectricity Sensor" identifies a three phase electricity meter. In addition, the device includes a relay, which can be used to turn the power supply on and off.

Fragments use a naming convention similar to fully-qualified Java class names to avoid conflicts between different parties contributing fragment information. A set of standard fragments are defined by the platform. See the client library documentation for more details.

Note that the Backend Device Management GE follows a document-oriented approach for storing data. All capabilities of an object can be inferred from the document with the object data itself. There is no explicit separate metadata model that needs to be configured and managed. However, applications can, of course, add own metadata and store that as well in the inventory. For example, a vending application can maintain metadata about slot configuration of the diverse vending machine types in the inventory.

20.3.2.3 *ETSI M2M*

- ETSI M2M Architecture Overview
- ETSI M2M Specifications rel. 1

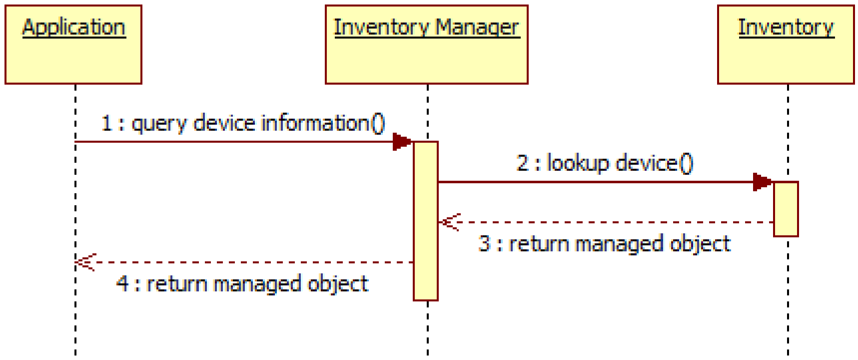
20.3.2.4 *OMA NGSI*

- OMA NGSI Context Management - Open Mobile Alliance V1.0 Aug 3, 2010
- FI-WARE NGSI9/10 Information Data Model
- FI-WARE NGSI Open RESTful API Specification (PRELIMINARY)

20.4 Main Interactions

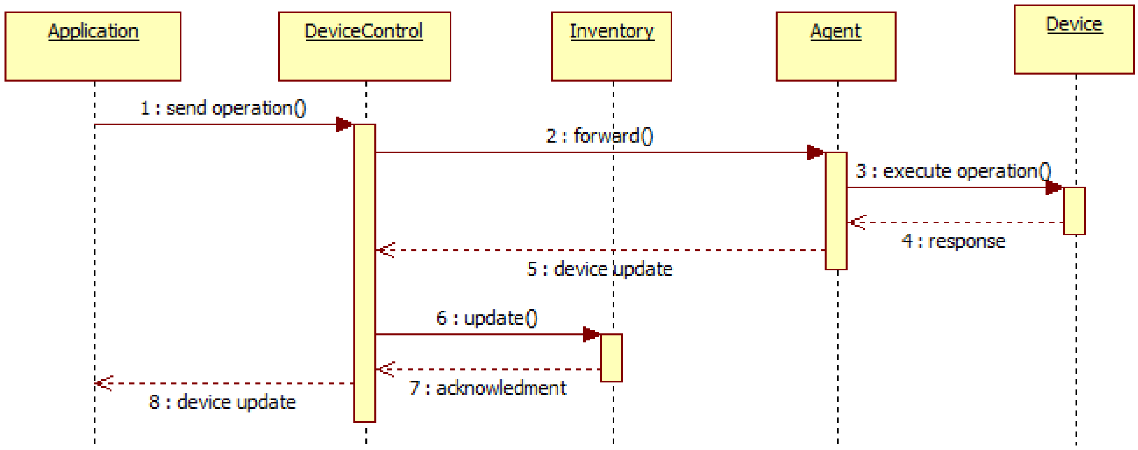
20.4.1 Retrieve Device information

To retrieve information from device(s), first the application sends the query to the Inventory Manager. The Inventory Manager queries the Inventory based on the received criteria and return the result to the application.



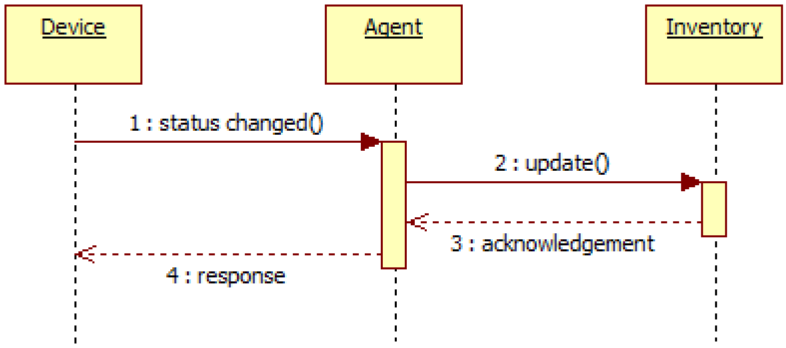
20.4.2 Sending control operations to Device

To pass an operation from a backend application to a device, a process of several steps is required: First, the application sends the operation to the Device Control. Then, the Device Control routes the operation to the agent managing the target device. The agent translates the operation to the protocol required by the device and sends it to the device. Finally, any responses are returned to the system.



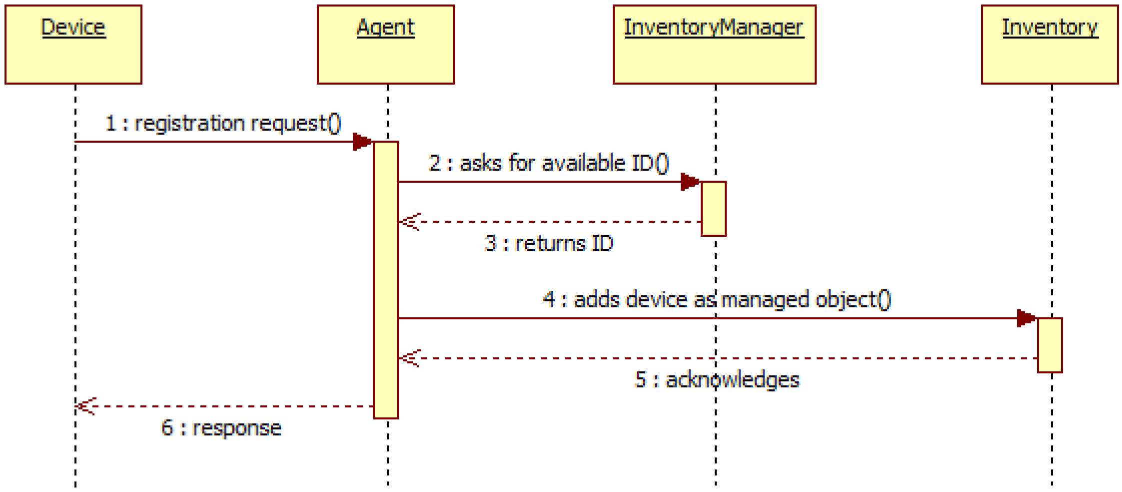
20.4.3 Device Push Update

When the status of the Device is changed, it send the status update to the Agent. The agent then updates the device information in the inventory.



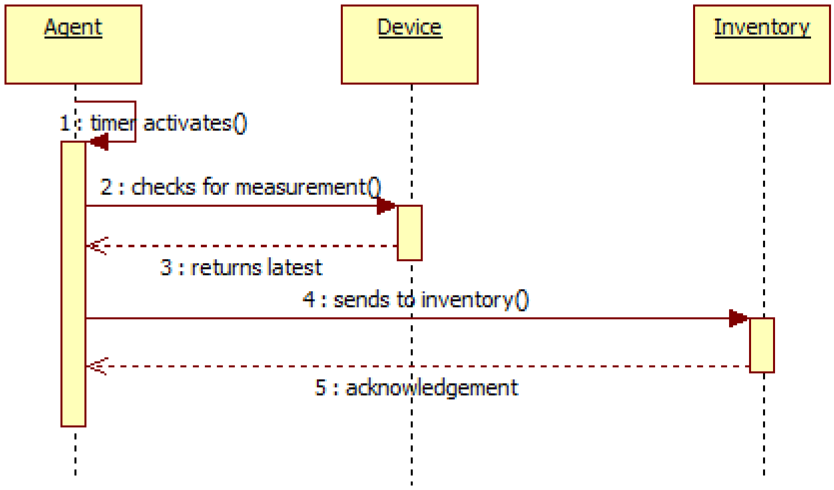
20.4.4 Device Registration Southbound

This scenario depicts the a registration initiated by the device itself. We have to assume that the device either uses the ETSI M2M mId interface, or the Backend Device Management instance has an Agent that translates the protocol of the device. First, the device sends a registration request to the Agent. The address of the Agent/Backend Device Management GE may come in several ways. The two basic possibilities consists of being added during the device manufacturing process or inserting later by the user via a smartcard. Once the Agent receives the registration request, it asks for an available ID within the GE from the Inventory Manager. Once the ID is obtained, the Agent adds the new managed object to the Inventory.



20.4.5 Measurement Collection

In this scenario the Agent periodically checks the device for new measurement information. First, the previously set internal timer activates the process. Then, the Agent checks for new measurement. In case a new measurement is ready, the device sends it to the Agent. Finally the Agent stores the new measurement in the Inventory.



20.5 Basic Design Principles

- The concept of *managed objects* and *fragments* were introduced in the Inventory to access data of remote sensors and use remote controls independent of device manufacturer, but still capture manufacturer-specific data where required. Using this approach, modeling devices can be split into modeling the elementary sensors and controls as fragments, and modeling the entire device as a combination of sensors, controls and possibly proprietary aspects of the device. It also enables developing generic application components. For example, as soon as a managed object has a coordinate fragment, it can be placed on a map. As soon as it has a relay, it can be switched on and off using the respective device control command.
- Today's IoT standards rarely define exactly how to access particular readings of particular sensors or manipulate particular controls. Devices may be connected through mobile networks and gateways. Protocols to devices/gateways range from low-level serial links to full-blown IT protocols such as web services. Thus, the Backend Device Management GE introduced the Agent Framework along with the *agent* concept similar to the Protocol Adapter GE on the gateway level to mediate between the various connectivity options and protocols.

21 FIWARE ArchitectureDescription IoT Gateway DeviceManagement

You can find the content of this chapter as well in the wiki of fi-ware.

21.1 Copyright

21.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

21.3 Overview

The Gateway Device Management GE is contains much of the "core" gateway functionality. It is responsible for the communication with the Backend and IoT and non-IoT devices. The Gateway Device Management GE includes the functional components to handle the registration/connection phases towards the Backend/Platform, to translate the incoming data or messages in an internal format and to send the outgoing data or messages in the ETSI M2M format (marshall/unmarshall). It is also capable of managing the communication with the IoT Resources, i.e. the devices connected to the IoT Gateway (that may be online or offline), and resources hosted by the gateway. The GE also contains Resource Management capabilities, i.e. to keep track of IoT Resource descriptions that reflect those resources that are reachable via the gateway. These can be both IoT Resources, or resources hosted by legacy devices that are exposed as abstracted IoT Resources. In addition, any IoT resource that is hosted on the gateway itself if also managed by this GE. The GE makes it possible to publish resources in the gateway, and also for the backend to discover what resources are actually available from the gateway.

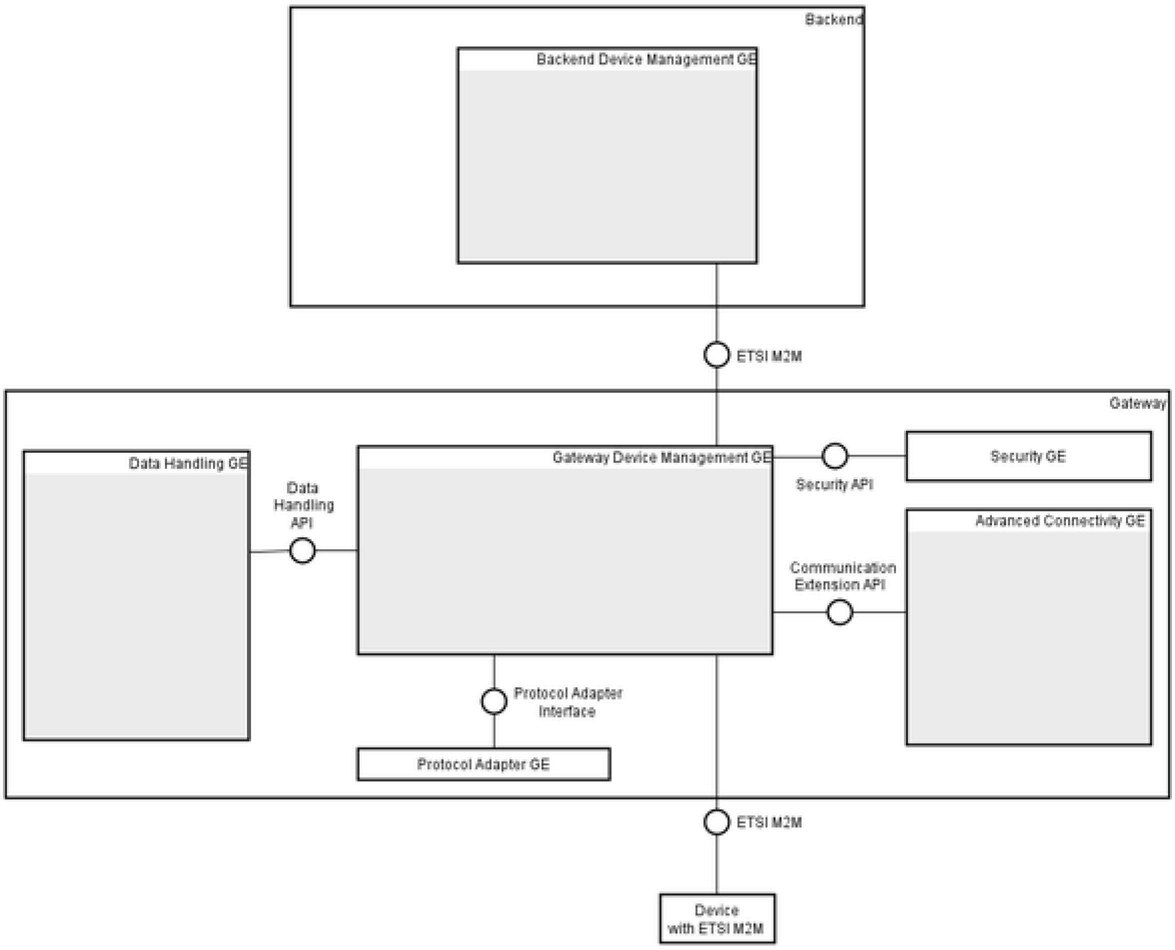


Figure 1: Gateway architecture

21.4 Basic Concepts

Figure 1 shows an overview of the IoT gateway and the position of the Gateway Management GE. The Gateway Device Management GE currently has four interfaces:

- A northbound interface towards the backend is used for retrieving information from IoT and non-IoT resources and gateway-hosted resources. This interface is also used for resource discovery. This is currently based on IETF CoRE.
- A southbound interface towards native IoT resources used for retrieving sensor data, managing subscriptions etc. Essentially the same as the backend northbound interface. Currently based on IETF CoRE.
- A southbound interface towards the Protocol Adapter GE used for communicating with non-IoT resources
- An interface towards the Data Handling API used by the Publish/Subscribe broker for creating, updating and deleting subscriptions to resources. Also used to retrieve locally stored data from IoT resources.

The Security API and Communication Extension API's may be supported in future releases of Fiware.

Figure 2 shows the internal architecture of the Gateway Device Management GE.

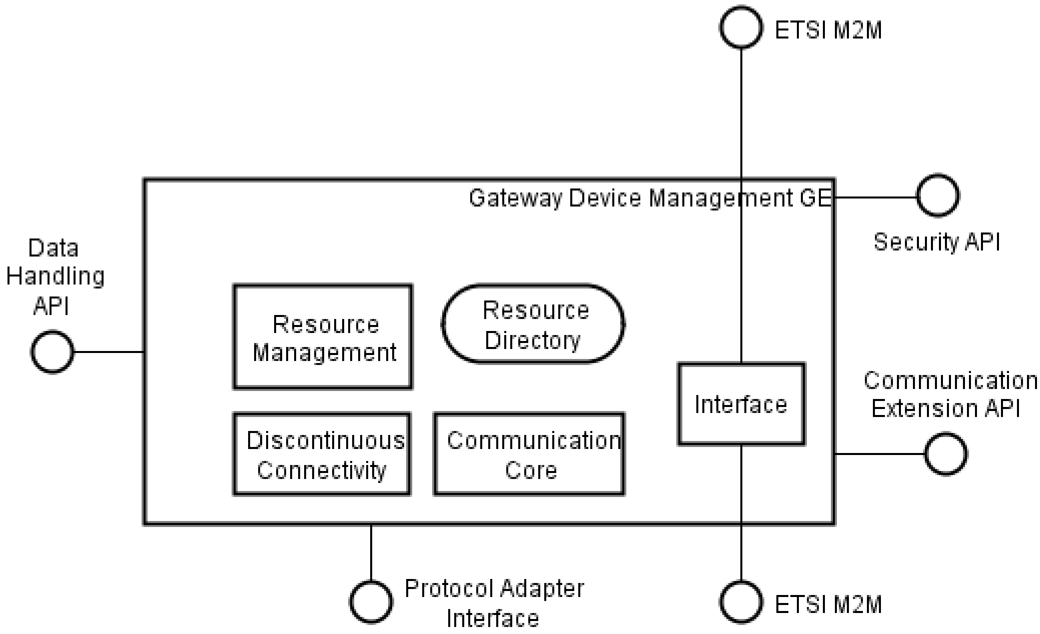


Figure 2: Gateway Device Management GE internal architecture

21.4.1 Resource Management

Resource Management is about handling information of connected devices and the resources they host. The Resource Management component makes it possible to discover resources by doing lookup searches, and store resource descriptions.

The Resource Directory is a data store for IoT and legacy resource descriptions used for discovering which device that hosts a particular resource. Typically these resource descriptions can contain information about the endpoints that host the resources (address, port, etc), the type of resource (e.g. temperature), and contextual data such as position. The Resource Directory supports looking up resource descriptions, as well as publishing, updating and removing resource descriptions to it.

21.4.2 Discontinuous connectivity

Discontinuous Connectivity Management deals with the connectivity with the Backend/Platform and with the Protocol Adapter GE for the connectivity with the IoT Resources. When the Backend/Platform wants to communicate with an IoT Resource it sends its message to the Discontinuous Connectivity Management module so that it can check if the IoT Resource is currently connected. If it is connected the Discontinuous Connectivity Management forwards the message directly to the IoT Resource, otherwise it stores the message into a Connectivity Cache. When the IoT Resource connects to the

Gateway then the messages stored in the Connectivity Cache will be forwarded. When an IoT Resource wants to communicate with the Backend/Platform it sends its message to the Discontinuous Connectivity Management module so that it can check if the Gateway is currently connected to the Backend/Platform. If it is connected the Discontinuous Connectivity Management forwards the message directly to the Backend/Platform, otherwise it stores the message into the Connectivity Cache. When the Gateway connects to the Backend/Platform the messages stored in the Connectivity Cache will be forwarded.

Connected Device List [store] is a repository that keeps all the information of the different IoT Resources registered and connected to this IoT Gateway, providing information about:

- The identifier of the IoT Resources
- What are the properties/capabilities of the IoT Resources
- The registration status of the IoT Resources
- The connectivity status of the IoT Resources

21.4.3 Communication Core

The Communication Core contains the basic communication capabilities of the gateway to setup communication to the IoT Backend and IoT Devices.

21.5 Main interactions

21.5.1 Resource Management

IoT Resources can be both contained on devices outside the gateway and locally stored (ex. cached data) within the gateway. Figure 3 shows how devices outside the gateway can:

- Create an entry in the Resource Directory exposing the availability of a particular IoT Resource.
- Delete an entry in the Resource Directory, e.g. when the IoT Resource entry is not available any more.
- Update an entry in the Resource Directory, e.g. to reflect a change in the service description of the IoT Resource.
- Publish entries in the Resource Directory, i.e. resources that are exposed by the gateway, to the backend Resource Directory

Devices can be both "native" IoT devices that communicate directly with the Gateway Device Management GE or legacy devices where communication pass through the Protocol Adapter GE as pictured below. The Gateway Device Management GE can also publish resource descriptions to the backend on behalf on devices. Notable in this figure is that the Discontinous Connectivity Management component will check if the backend is online. Otherwise this request will be cached and sent at a later stage when the backend/gateway is back online.

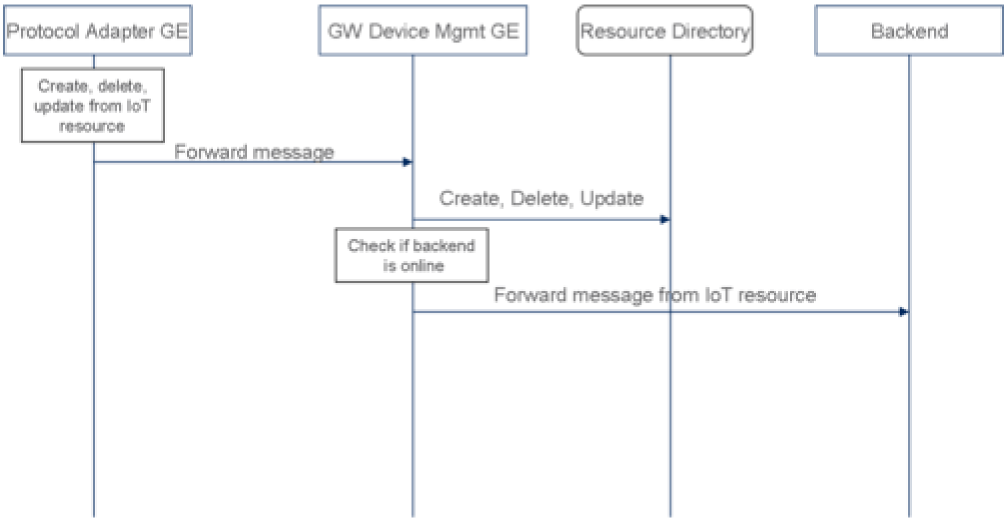


Figure 3: Resource Management (external resources)

The IoT gateway can also contain locally stored resources. In this case these resource descriptions are contained within the Data Handling GE. Figure 4 shows how to:

- Create an entry in the Resource Directory exposing the availability of locally stored data for a particular IoT Resource.
- Delete an entry in the Resource Directory, e.g. when the IoT Resource entry in the Data Store is not available any more.
- Update an entry in the Resource Directory, e.g. to reflect a change in the service description of the IoT Resource.

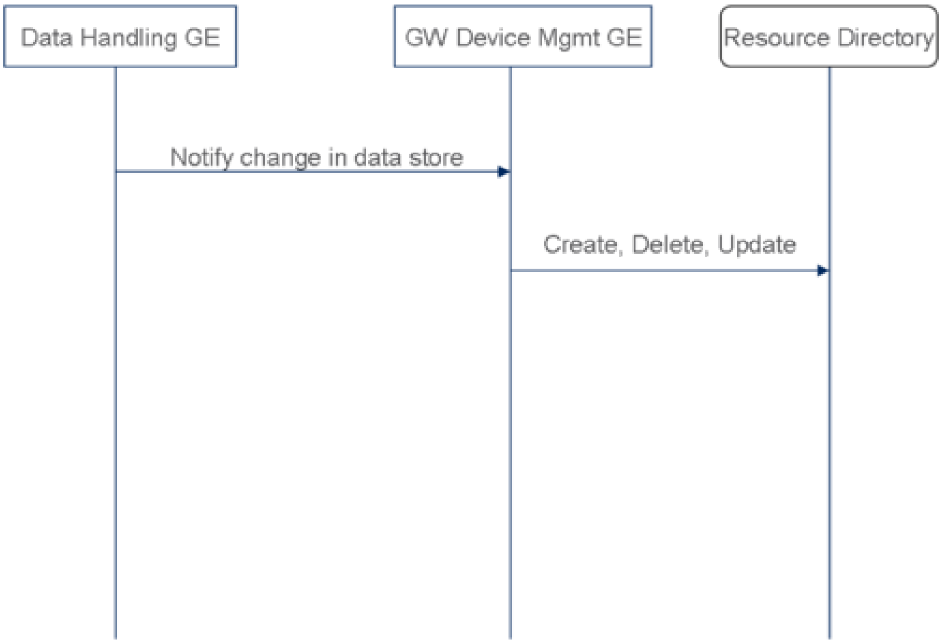


Figure 4: Resource Management (internal resources)

The Gateway Device Management GE exposes an API so that it is possible for the backend to read entries in the Resource Directory, ie. discover which IoT resources that are exposed by the gateway.

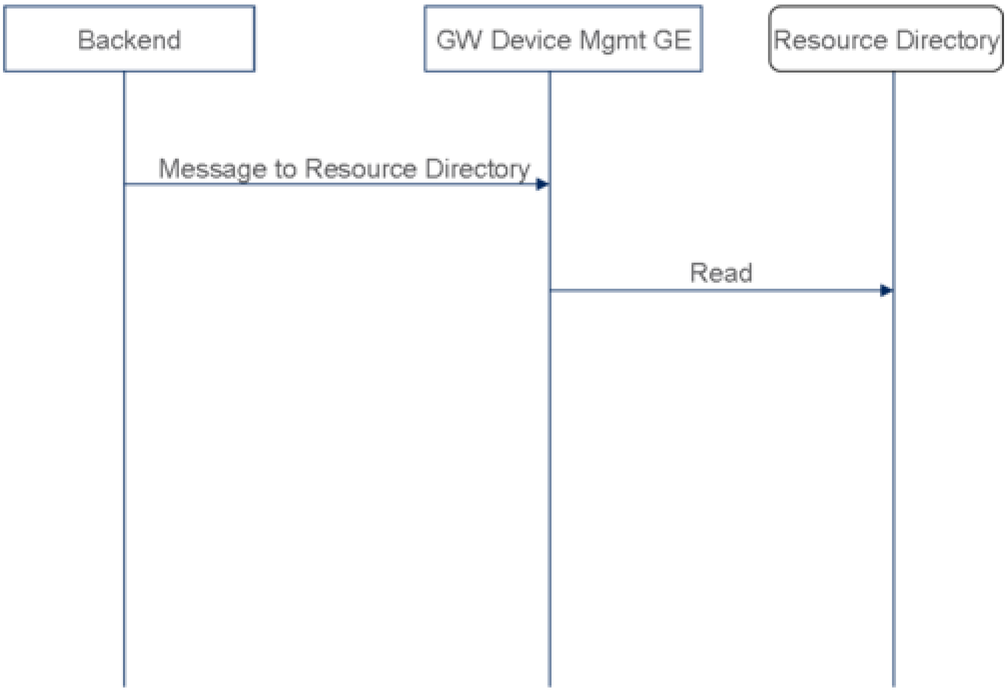


Figure 5: Lookup resources

21.5.2 Accessing resources

The Gateway Device Management GE enable basic communication with resources hosted on devices outside the gateway as well as resources hosted by the gateway.

Figure 6 shows how the backend can manipulate an IoT resource by Create, Read, Update, Delete (CRUD) operations on an IoT resource. Notable in this figure is that the Discontinous Connectivity Management component will check if the device hosting the resource is online. Otherwise this will be cached and sent at a later stage when the device is back online.

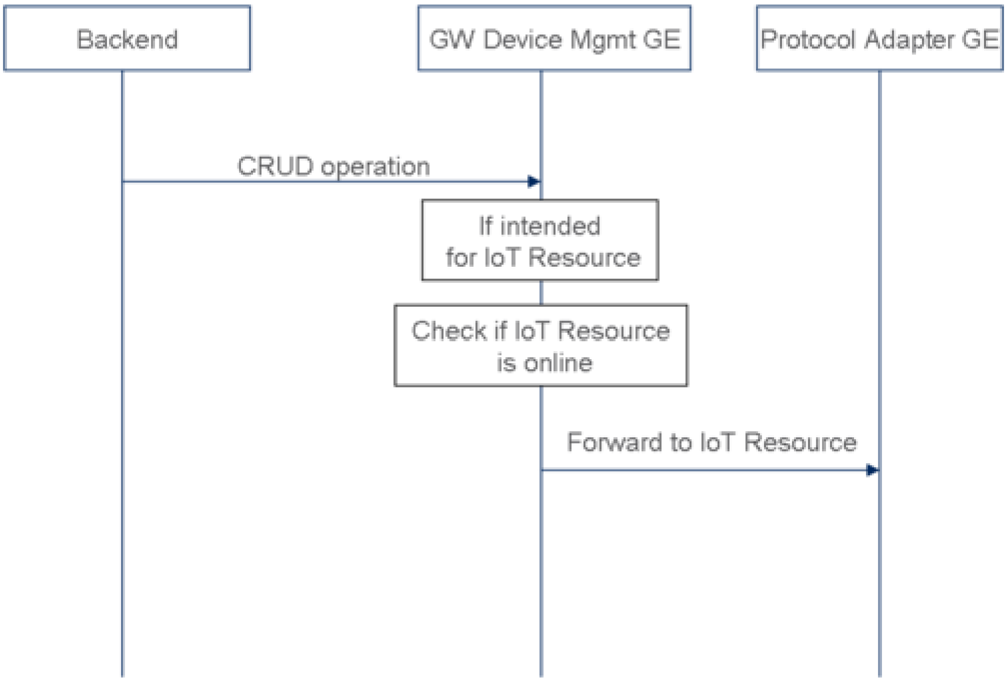


Figure 6: Access to resource hosted by device

Figure 7 shows the corresponding Create, Read, Update, Delete operations on a resource hosted by the gateway.

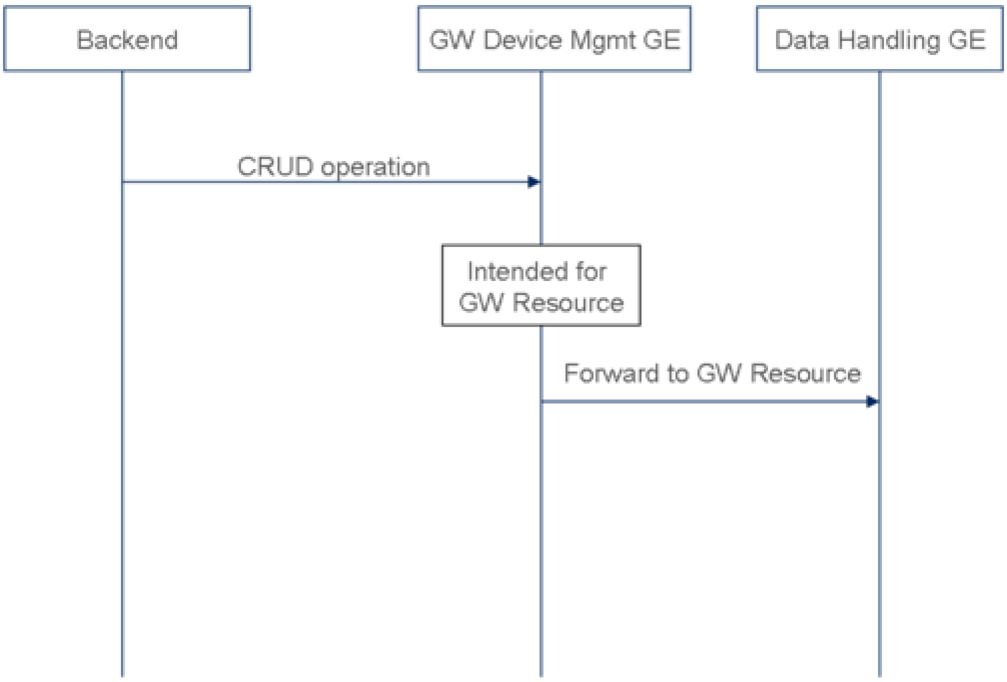


Figure 7: Access to resource hosted by the gateway

21.6 Basic Design Principles

The Ericsson gateway is designed to be "lean and clean", in the sense that it is adapted for constrained environments and following a simple RESTful web service approach. Therefore the IETF CoRE standards has been selected as the base for the Ericsson Gateway. The CoRE RESTful approach is also not limited to the CoAP protocol. On the contrary, this also means that for the interface towards the Service Enablement there can be bindings to multiple protocols, e.g. HTTP or CoAP. Specifically for HTTP, work is ongoing on an implementation of web sockets to accomodate asynchronous request handling for observing sensor values.

22 FIWARE ArchitectureDescription IoT Gateway DataHandling

You can find the content of this chapter as well in the wiki of fi-ware.

22.1 Copyright

22.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

22.3 Overview

The IoT world mainly consists of a huge amount of resources, creating a lot of events to be processed.

The Data Handling GE addresses the need of filtering, aggregating and merging data from different sources; merging data is considered to be the main value-added feature. Applications should receive value-added data that are relevant to their needs thanks to the Complex Event Processing technology (CEP). This is also referred to as event stream analysis. Events are subscribed to by an application, depending on the access rights that have been granted. The application is then able to receive published events.

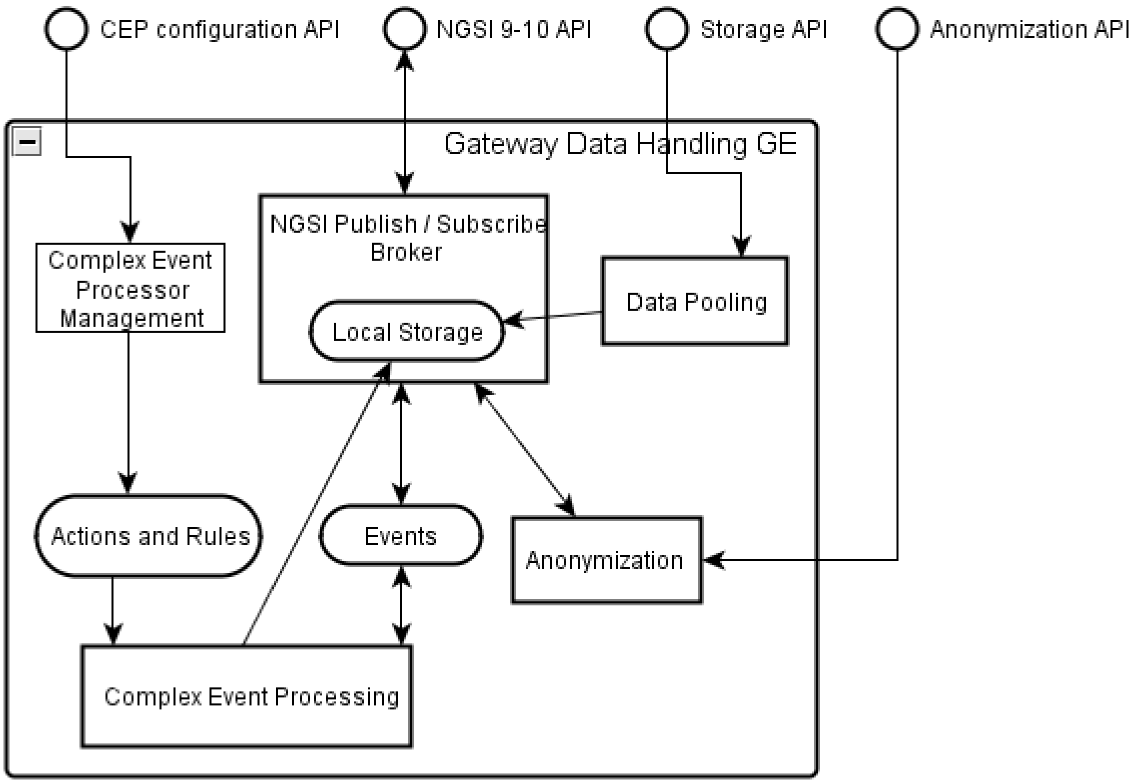
The Data Handling GE comes in two flavours : the Esper4FastData for mobile and fixed gateways, and SOL/CEP which targets more constrained environments.

Typical applications that require Data Handling GE are sensor network applications, RFID reading, supply chains, scheduling and control of fabrication lines, air traffic and so on. What these applications have in common is the requirement to process events (or messages) in real time or near real time. Key considerations for these types of applications are throughput, latency and the complexity of the logic required. The Data Handling CEP component typically processes input data in order to generate a smaller set of output data that avoids backend and network flooding.

The Data Handling GE also provides support for IoT hosting devices that for various reasons cannot be continuously online. The main reason is that the devices work under resource constraints such as being battery operated. This typically requires asynchronous communication with the IoT resources and for this purpose, local storage is required. To this aim, the Local Storage component provides two support functions. The first is to store cached IoT resource requests from the backend that cannot be processed due to the relevant IoT device being off-line. The second is to locally store retrieved data from IoT resources for subsequent retrieval by the backend and ultimately the applications.

It is essential to handle the data and events from the IoT Resources (i.e. the devices connected to the IoT Gateway) or from the Backend/Platform (i.e. the IoT Services Enablement Platform) in a store. In particular the Data Handling GE deals with the Things Management GE events, that are propagated from and to the Backend/Platform. It also deals with the Gateway Device Management GE events that are propagated from and to the IoT Resources. For that purpose, the Data Pooling component provides functionality to discover what is stored in the data store, to retrieve, to create, to delete, to update the data and events of the data store.

For privacy and security concerns, access and storage rights are defined at the closer level of data production, to devices or to IoT resources. The Data Access Policy component is a dedicated additional component to the Security, Privacy and Trust which is providing features for authentication and centralized anonymazing mechanisms.



Data Handling GE architecture

[1] IoT-DataHandlingGE.graphml

22.3.1 Main components

22.3.1.1 *Southbound, NGSI 9-10 API (link between Data Handling GE and any other GE)*

This is part of the Data Handling API. It is compliant with the New Generation Service Interface (NGSI), and allows events to typically flow from the Gateway Device Management GE towards the Data Handling GE.

Device Management GE registers itself as a context provider, thanks to the NGSI-9 registerContext method provided by Data Handling GE. After context registration, Device Management GE sends events by calling updateContext method of the Data Handling GE.

22.3.1.2 *Northbound (link between Data Handling GE and Backend)*

Northbound interface uses the same interface as southbound and as such it is NGSI 9-10 compliant. It means southbound and northbound use the same entry point.

Data Handling GE registers itself as a context provider, by calling the registerContext method of the upper level GE, which is typically the Things Management GE. After registration, Data Handling GE calls the updateContext method of the Things Management GE in order to send events.

Optionnaly, third party software and other external GEs can also subscribe to Data Handling GE output events through the NGSI-10 registerContext method of the Data Handling GE. For every subscriber, events are propagated by calling its notifyContext method, which means subscribers should at least implements this method, otherwise they could not receive events.

22.3.1.3 *Publish / Subscribe Broker component*

This component is an immplementation of NGSI. It allows the CEP / Data Aggregation Filter component to communicate with the outer world and the other GEs, in an ingoing and outgoing way. It provides NGSI methods like queryContext, updateContextSubscription, unsubscribeContext, updateContext, registerContext...

The PubSub Broker relies on the Anonymization component in order to manage subcription grants. It will interwork with centralized Publish/Subscribe Broker GEs both supporting the interfaces defined in the Data/Context Management chapter. This way, it will be feasible to create a reliable network of Publish/Subscribe Brokers that propagates subscription service conditions down to potential IoT resources.

22.3.1.4 *Complex Event Processing component*

As a demonstrator, two different Complex Event Processing components are provided; one by Orange, aimed at a smart phone and another by Atos, aimed at small, embedded sensor gateways that can be installed at various points in a sensor network infrastructure, e.g. a Smart City.

The role of both CEP is to play simultaneously the role of filtering/aggregating data and processing events, both roles being based on rules. The CEP understands NGSI-10 on its input and processes all events coming from the Data Pooling component, through the PubSub Broker component. It's the role of Data Pooling component to translate various protocols into NGSI-10.

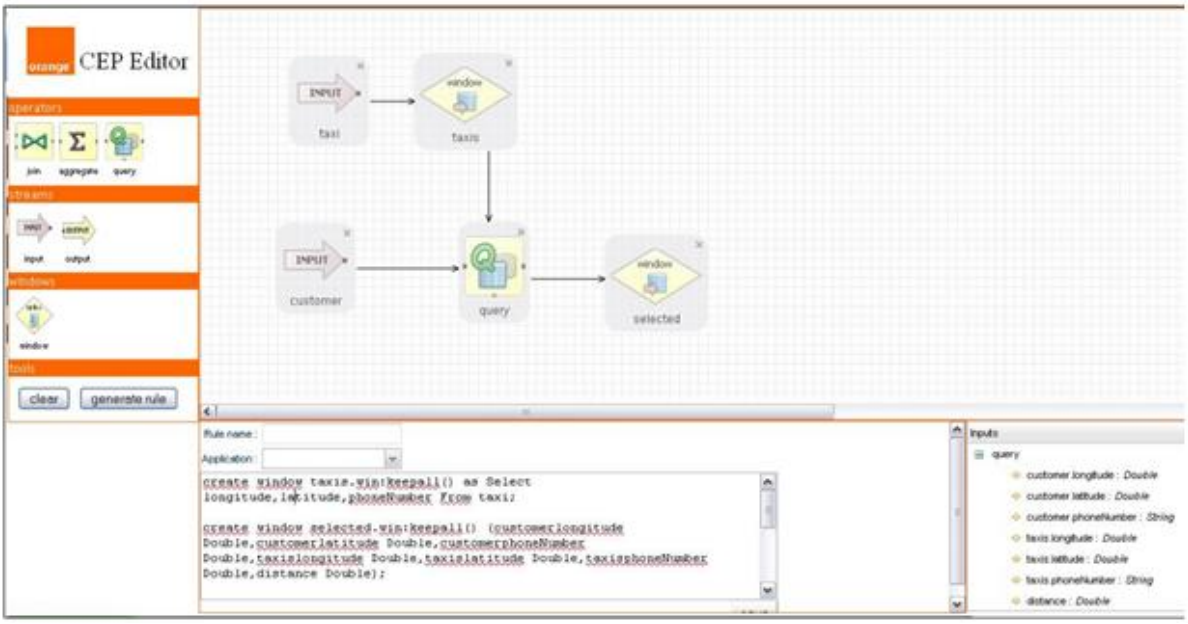
After translating a protocol-dedicated event into NGSI-10, the Data Pooling component stores this raw information into the Micro DB component. Each event at the CEP output is also stored in the Micro DB. Therefore events can be tagged "raw" or "CEP processed". The amount of stored events is up to the Local Storage component, and depends of course on this component memory capacity.

CEP output events are subscribed by external GEs through the NGSI-10 compliant PubSub broker. Data structures travelling between CEP component and PubSub Broker component are NGSI-10 compliant.

The following is only applicable to the Fast Data4Esper CEP flavour. SOL/CEP does not provide the same features :

This feature displays an API that allows CEP management with a dedicated application. Ideally, the displayed CEP API methods have to be consistent between chapters and context. They should be the same, whether the implementation concerns a device, a mobile phone, a gateway or the backend.

For example, using a portal, the admin user is able to start, stop, configure and monitoring the CEP engine. A graphic editor allows to add, delete or modify rules and to check the syntax.



CEP rules editor

22.3.1.5 *Anonymazing component*

The Anonymazing feature enables the devices or applications to return Anonymazing data, i.e. elements on outgoing messages can be blanked out based on internal policies. At the extreme it may blank out all elements, in other words, not return any data, depending on who is asking for it. Alternatively, enforce the "random noise" which is equivalent to "decrease the accuracy of the data accoring to agreed SLA".

22.3.1.6 *Data Pooling component*

The Data Pooling component provides methods to access and manage history records of events.

Architecture

Next follows a description of the different functional blocks of this architecture.

Data Models Exposure deals with both the Device Handler GE and the Device Frontend GE to allow the following operations on the data and events stored in the data store:

- create,
- store,

- retrieve
- update

When an incoming request gets to this component it interacts with the Data Store in order to fulfill the action requested and interacts with the Data Handling GE to send a copy of events and data in case of an update operation.

Local Data Discovery deals with both the Device Handler GE and the Abstract Protocol Layer GE to allow the search within the Data Store.

The Backend/Platform, via Device Handler GE, may find data and events stored by the IoT Resources and the IoT Resources, via Device Frontend GE, may find data and events stored by the Bakend/Platform.

Interfaces

An internal interface is useful to both the Backend/Platform and the IoT Resources to create, store, retrieve, update the data and events via Device Level API (for the Backend/Platform) and via Device Frontend GE (for the IoT Resources).

An internal interface is useful to both the Backend/Platform and the IoT Resources to discover the data and events which are stored in the data store, via Device Level API (for the Backend/Platform) and via Device Frontend GE (for the IoT Resources).

22.3.1.7 *Local Storage component*

Architecture

The Local Storage component contains two functional blocks, the Connectivity Cache and the Data Store.

The Connectivity Cache contains IoT Resource requests that have not been able to be processed due to the IoT Resource in question being off-line. A request is stored for a limited time period, after which the request is automatically removed if not consumed by an IoT Resource going online.

The Data Store contains cached data from IoT Resources. This means that the Data Store can hold historical records of e.g. sensor readings. Necessary metadata is added to the actual stored data, such as source identification and time stamps. Retrieval can be done by single item or in bulk.

Interfaces

The Local Storage has the following main interfaces.

- Direct access to the Data Store which exposes explicitly addressable data via the Device level API in the Exposure GE.
- An interface to the Data Handling GE for the purpose of storing IoT Resource data readings coming from IoT devices via the Publish/Subscribe Broker. This is used for Observation requests tasked to the IoT Resource in question.
- An interface the the Resource Directory in the Resource Management GE. This is a resource publication interface so that the Resource Directory can be populated with information about IoT resource data available in the Data Store.
- An interface used by the Discontinuous Connectivity Management function in the Connectivity Management GE for caching requests directed to offline IoT Resources.

22.3.2 Basic concepts

22.3.2.1 *Complex Event Processing component*

Event concepts

Similar to many topics in science and engineering, the term event has different meanings based on who is observing the event and the context of the observation. Generally speaking, event is something notable that happens. It can be represented as an object that encodes or records an event, generally for the purpose of computer processing. Events can contain data, are immutable, but more than one event may record the same activity. It should be noted that events are highly context dependent.

Event preprocessing is a general functionality that describes normalizing data in preparation for upstream, ”higher level,” event processing. Event preprocessing is often referred to as data normalization, validation, prefiltering and basic feature extraction. When required, event preprocessing is often the first step in a distributed, heterogeneous complex event processing solution. Heterogeneous, distributed event processing applications normally require some type of event preprocessing for data normalization, validation and transformation.

Complex event processing (CEP) consists of processing many events happening across all the layers of an organization, identifying the most meaningful events within the event cloud, analyzing their impact, and taking subsequent action in real time.

Complex event processing refers to process states, the changes of state exceeding a defined threshold of level, time, or value increment or just of a count as the event. It requires the respective event monitoring, event reporting, event recording and event filtering. An event may be observed as a change of state with any physical or logical or otherwise discriminated condition of and in a technical or economical system, each state information with an attached time stamp defining the order of occurrence and a topology mark defining the location of occurrence.

Complex Event Processing in the Data Handling GE will be provided with two different interchangeable assets, Esper (Orange) and SOL/CEP (Atos).

Esper Complex Event Processing engine

Esper is a CEP engine software, with its own rules language. It's Java oriented.

Esper [2] is a component for Complex Event Processing (CEP) available for Java as Esper, and for .NET as NEsper. Esper CEP engine is a open-source software under the GNU General Public License GPL. Esper CEP engine is mainly used for:

- Complex computations - applications that detect event correlation, filter events, aggregate time or length windows of events, join event streams, trigger based on absence of events etc.
- High throughput - applications that process large volumes of messages (between 1,000 to 100k messages per second)
- Low latency - applications that react in real-time to conditions that occur

Events format can be plain Java objects, XML and java.util.Map. Adapters allowed are CSV, JMS, DB, HTPP Input and output and Socket input adapters. Esper offers Event Processing Language (EPL) for Event Processing Rules. EPL is an SQL like language, which is an optimized language for dealing with high frequency time-based event data. EPL language allows grouping, aggregation, sorting, filtering, merging splitting and duplicating of event stream, time and length windows, event pattern matching, creation of new complex events.

Three main concepts drive the CEP component structure, which can be divided into three sub-components:

- The generators receive events from the PubSub Broker, unmarshal them and transfer the received data to the CEP core
- The CEP core applies rules to the received data. If the data matches the rules an event is internally generated and transfered to the appropriate listener
- In reaction to a core event, the listener propagate an NGSI event to the PubSub broker

EPL statement structure

[insert into insert_into_def] select select_list from stream_def [as name] [, stream_def [as name]] [,...] [where search_conditions] [group by grouping_expression_list] [having grouping_search_conditions] [output output_specification] [order by order_by_expression_list]

Example of rules statement:

- Temporally store event: CREATE WINDOW…

```
       CREATE WINDOW
StoredTaxiPresenceEvents.std:groupwin(phoneNumber).win:length(1)
(phoneNumber string,status boolean)
```

- Insert in Windows: INSERT INTO…

```
       INSERT INTO StoredTaxiPresenceEvents SELECT
phoneNumber,status FROM PresenceEvent where userType='Drive'
```

- Correlates events: SELECT evt1.prop, evt2.prop…

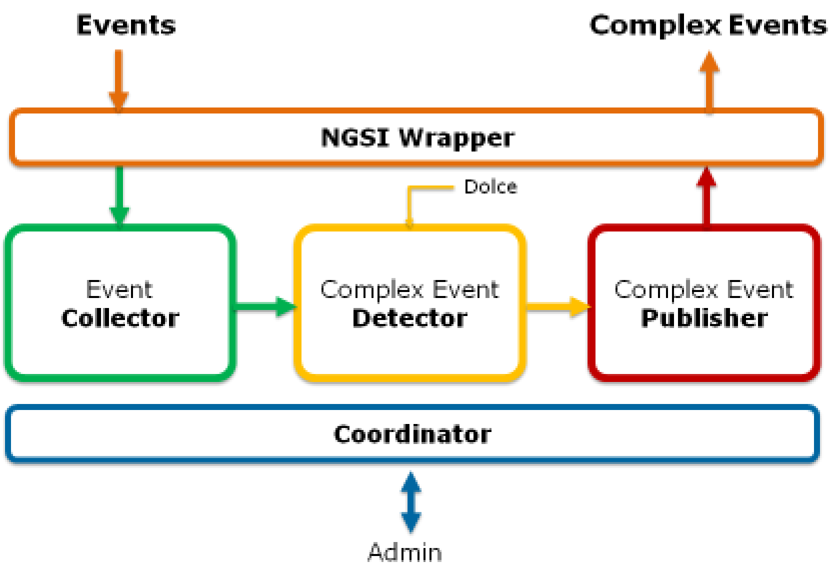
```
       select fraud.accountNumber as accntNum, fraud.warning as
warn, withdraw.amount as amount,

       MAX(fraud.timestamp, withdraw.timestamp) as timestamp,
'withdrawlFraud' as desc

       from FraudWarningEvent.win:time(30 min) as fraud,

       WithdrawalEvent.win:time(30 sec) as withdraw

       where fraud.accountNumber = withdraw.accountNumber
```

- Instantiation of listener: ON event DO…
- Product new Events: INSERT INTO Event2…

SOL/CEP Complex Event Processing engine

SOL/CEP is a Complex Event Processor characterised by scaleability and performance. It accepts complex events that are described in a dedicated, declarative language, called Dolce.

With Dolce it is possible to specify what individual events need to be detected, and how and which of these events are composed into Complex Events.



SOL/CEP High level architecture

The CEP Engine reads events from the Event Collector, analyses them in the Complex Event Detector and finally emits them through the Complex Event Generator. Events are accepted from a variety of sources and protocols (Database, file system, TCP/IP), translated to an internal format by means of adpaters and processed. Likewise, Complex Events can be published towards different channels in a variety of formats.

A special wrapper will provide compatibility with FI-WARE NGSI.

Characteristics

SOL/CEP is designed for the following characteristics:

- Sliding Time Window

  Applies a time window over the evaluation of the complex event.

  Example: all transactions during the last 10 hours. The "last 10 hours" is called the sliding time window and always refers to the events that happened in this time frame.

- Correlation between Event attributes and Complex Events

  This means that the values of the attributes that are specified in the Event can be reused and evaluated in the Complex Event.

  Example: the customer name from an incoming event can be copied to the Complex Event.

- Complex Event Functions

Aggregation allows totalizing the values of the incoming events, Averaging calculates the average of all values of a time windows.

Example: the sum of all the money that was withdrawn in the last 5 days.

Example: The average temperature during the last 20 minutes.

- Temporal Awareness

  A complex event is a combination of simple, individual events. The temporal order in which these events should happen for the complex event to be raised can be specified using the relative “after” or “during” specifiers

  Example: WarningLightsOff before LandingGearOut LandingGearUp during Approach

- Evaluation of attributes

  Attributes in the Complex Event can be evaluated using a free formula.

  Example: Average(temperature) >= 20

- Event Filtering

  Incoming Events can also be filtered based on a formula.

  Example: TransactionAmount > 1000

Technical aspects

The CEP runs as a process and is language agnostic. Accepts input and generates output through Format Adapters (explained before)

- Application level language binding is C++.
- Low footprint: designed for performance and small resource usage, but scalable to high end servers.
- Operating system support:

BSD, Linux (Debian).

Dolce Language

In Dolce, a user can specify Events that need to be detected as follows:

```
event RainfallMeasurement
{
  use { int Amount, int SensorId };
  accept { SensorId == 5 );
}
```

Tells the CEP to detect RainfallMeasurement events, with the attributes amount and sensorId. The event is only accepted for Sensor number 5. Other sensors are ignored. The amount represents the mm of rain that fell since the last measurement.

NOTE: the Format Adapter needs to map the event type and assign a number to the different incoming events. This is a convention that is decided at design time by the developer – in this case, the number 5 was chosen to identify RainfallMeasurement events.

A Complex Event could be specified as follows:

```
complex FloodAlarm
{
  type { 1600 };
  payload
  {
    Total { int Sum(RainfallMeasurement.Amount) }
  };

  detect RainfallMeasurement
  where sum(RainfallMeasurement.Amount) > 100
  in [ 10 minutes ];
}
```

Tells the CEP to raise a complex event when the total amount of rain exceeds 100mm in 10 minutes. The event will be of type 1600 and contains a field called "Total" which is an integer and contains the amount of rain that fell

NOTE: For the first release, only partial implementation of the Dolce language is implemented, implying also that features such as temporal and spatial awareness will be available in the second release.

22.3.2.2 *NGSI*

[3] Ref. NGSI Context Management - Open Mobile Alliance V1.0 Aug 3, 2010

[4] NGSI9/10 Information Data Model

[5] Context Information interface OMA_NGSI_10

22.3.2.3 *ETSI M2M*

[6] ETSI M2M communications

22.3.2.4 *XACML*

Data Access Policy component is based on XACML[7] (eXtensible Access Control Markup Language)supported by OASIS[8].

Based on XML this standard provides access rights control based on policy rules without to manage directly authentication which could be assume with SAML [9].

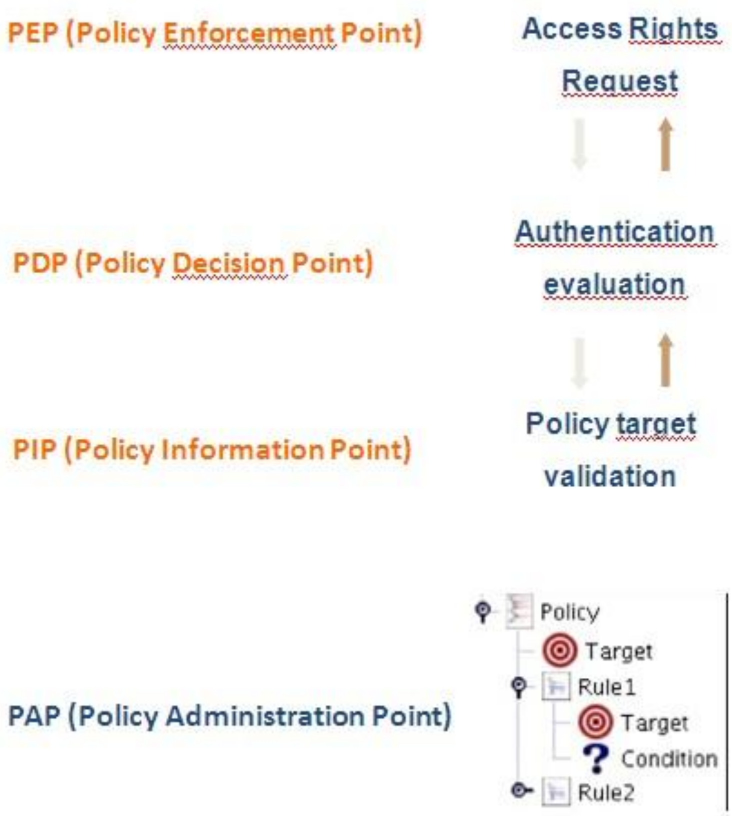
XACML policies are the required format for device or IoT resource owner to define access rights to related data or events.

This GE implement three main functionalities:

PEP (Policy Enforcement Point): to create the XACML request. Each things or application which needs an authorization for a dedicated action to reach IoT resource or device sends some parameters to support decision making process. All parameters are groued in an XML file.

PDP (Policy Decision Point): to evaluate XACML request. This functionnality provides three type of answer (Permit / Deny / not applicable) based on its evaluation of valuable policy rules for the related device or IoT resource.

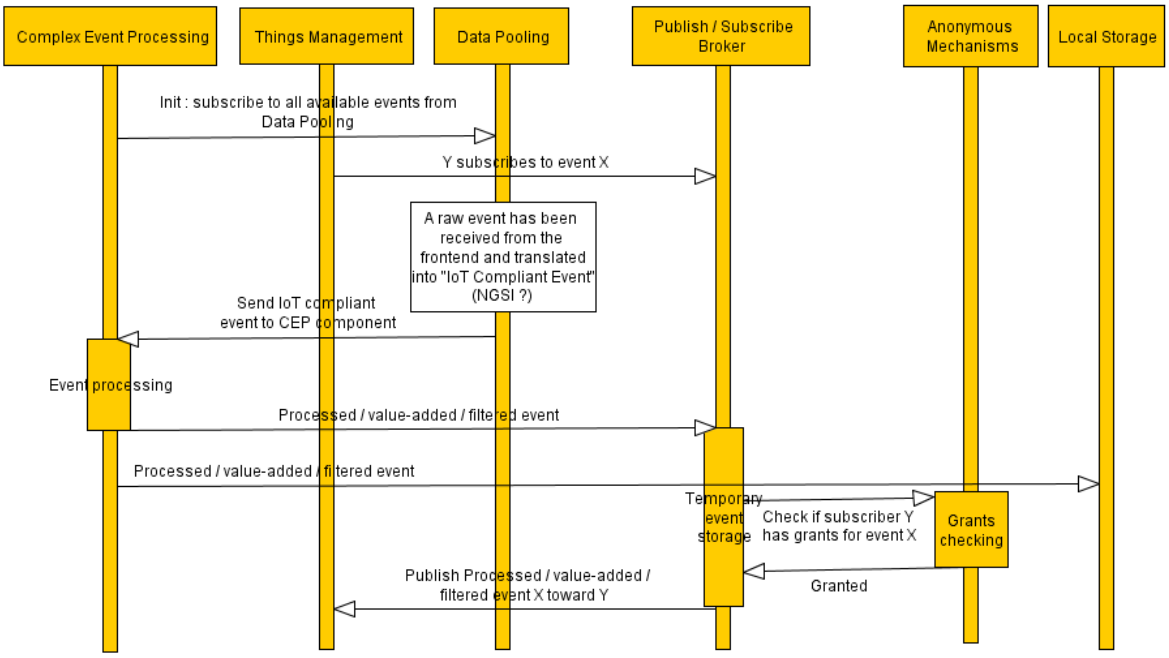
PAP (Policy Access Point): to evaluate XACML request. PAP is a key functionnality to create, modify and analyze politcy rules and proposes and management module to define access control rights for each users (applications).



22.4 Main Interactions

- The PubSub broker has to be compliant with its counterpart in WP6. Compliance means consistency in terms of data format (NGSI) and GE API.
- With Data Pooling component, which is the NGSI data provider for the CEP component
- With Things/Resources Resolution, from Things Management : Data Handling interacts with Things Directory Configuration Management in order to obtain a Things/Resources Resolution to resolve the Things whose properties have to be changed according to the observations received from Resources.

22.4.1 Typical interactions



22.4.2 Local Storage Component

22.4.2.1 *Device level API to Data Store*

- Read stored IoT Resource data from the Data Store. Schematically, the gateway receiving a GET to http://authority/datastore/device/resource will return stored data from the IoT resource identified by http://device/resource.

22.4.2.2 *Discontinuous Connectivity Management to Connectivity Cache*

- Create IoT Resource requests into the Connectivity Cache for later processing.
- Read cached IoT Resource requests.
- Delete cached IoT Requests.
- Update cached IoT Requests.

22.4.2.3 *Discontinuous Connectivity Management to Data Store*

- Read stored IoT Resource data from the data store.

22.4.2.4 *Publish/Subscribe Broker to Data Store*

- Create an IoT Resource data entry.
- Update an IoT Resource data entry when new data has been received e.g. by a notification from an IoT device.

- Delete an IoT Resource data entry, e.g. when an observation expires.

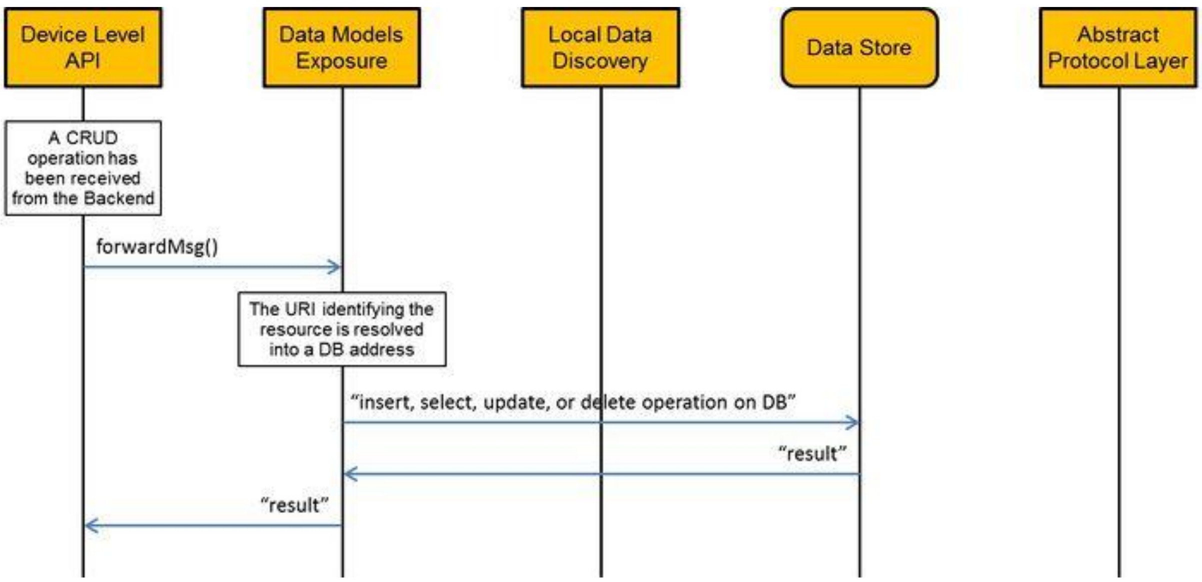
22.4.2.5 *Data Store to Resource Directory*

- Create an entry in the Resource Directory exposing the availability of locally stored data for a particular IoT Resource.
- Delete an entry in the Resource Directory, e.g. when the IoT Resource entry in the Data Store is not available any more.
- Update an entry in the Resource Directory, e.g. to reflect a change in the service description of the IoT Resource.

22.4.3 Data Pooling component

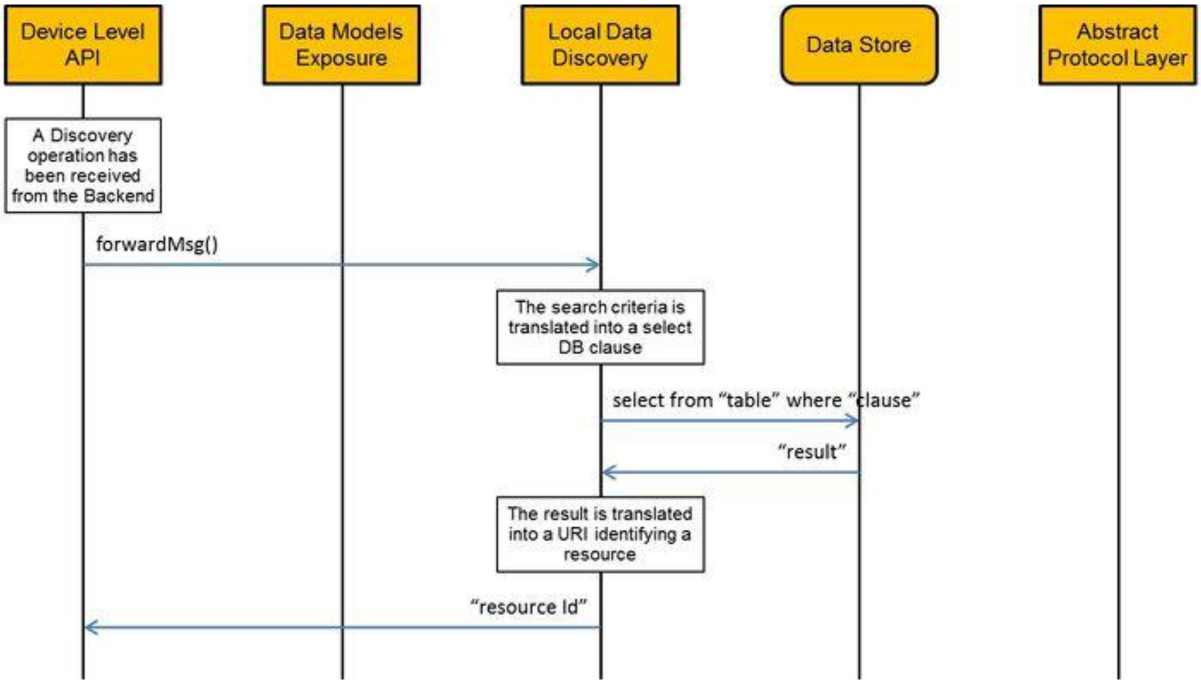
In the following pictures the main interactions occuring within the Gateway which involve mainly the Data Pooling GE are described. In particular it is depicted the activity when a message originated from the Backend/Platform or from an IoT Resource comes to the Gateway in order to work on ETSI M2M Resources stored at gateway level.

In the diagram shown by figure 5.1.3 it is described the call flow when a CRUD operation coming from the Backend/Platform (via the Device Level API component) is directed to an ETSI M2M resource contained in the Gateway (a CRUD operation is an operation on an ETSI M2M resource, where CRUD stands for "Create, Read, Update, and Delete"). The Data Models Exposure component resolves the resource ID into an identifier of a record in the Data Store store that contains the resource data, and translates the requested operation in the data format common to FI-WARE IoT, and then sends the corresponding operation to the Data Store. The result coming from the Data Store is translated back by the Data Store component and sent back to the Device Level API component.



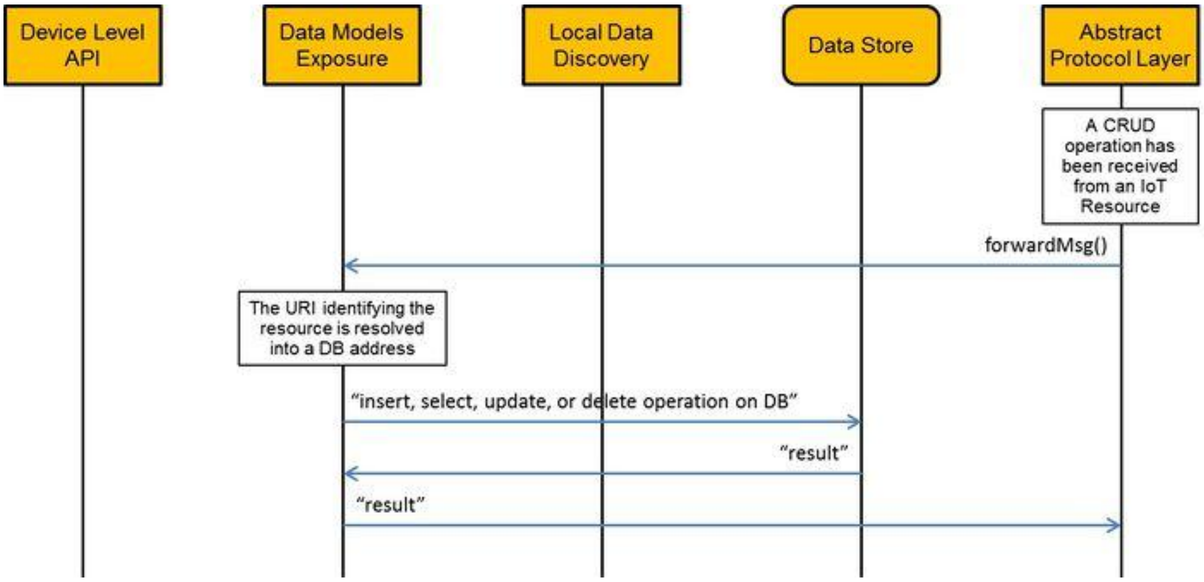
CRUD operation received from the Backend/Platform

In the diagram shown by figure 5.1.4 it is described the call flow when a Discovery operation coming from the Backend/Platform (via the Device Level API component) wants to search within the ETSI M2M resources contained in the Gateway. The Local Data Discovery component translates the search criteria contained in the request in the data format common to FI-WARE IoT, and then sends the corresponding operation to the Data Store . The result coming from the Data Store is translated into a resource ID by the Local Data Discovery component and sent back to the Device Level API component.



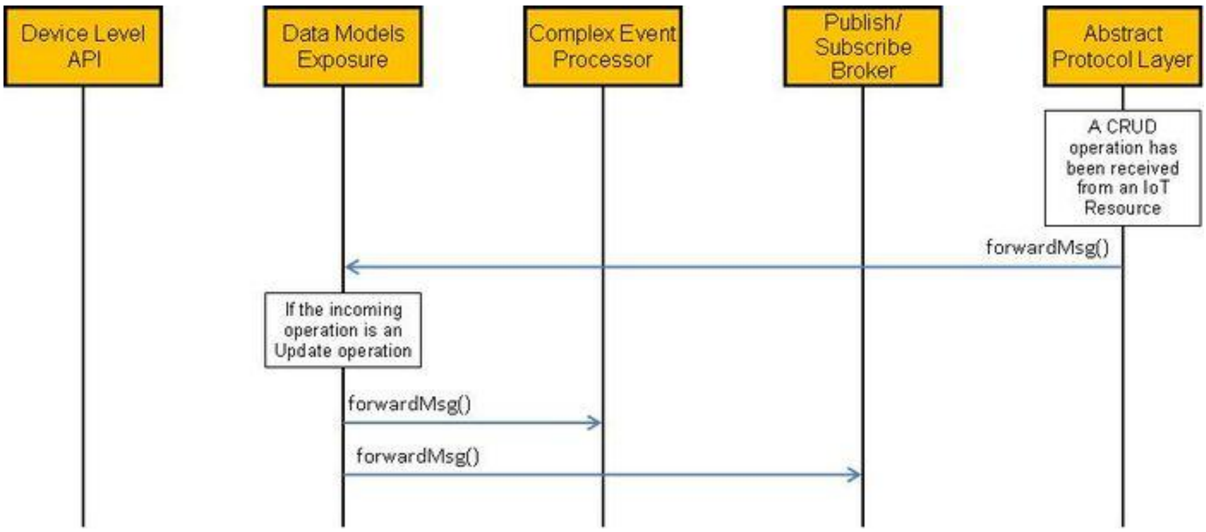
Discovery operation received from the Backend/Platform

In the diagram shown by figure 5.1.5 it is described the call flow when a CRUD operation coming from an IoT Resource (via the Abstract Protocol Layer component) is directed to an ETSI M2M resource contained in the Gateway (a CRUD operation is an operation on an ETSI M2M resource, where CRUD stands for "Create, Read, Update, and Delete"). The Data Models Exposure component resolves the resource ID into an identifier of a record in the Data Store store that contains the resource data, and translates the requested operation in the data format common to FI-WARE IoT, and then sends the corresponding operation to the Data Store. The result coming from the Data Store is translated back by the Data Store component and sent back to the Abstract Protocol Layer component.



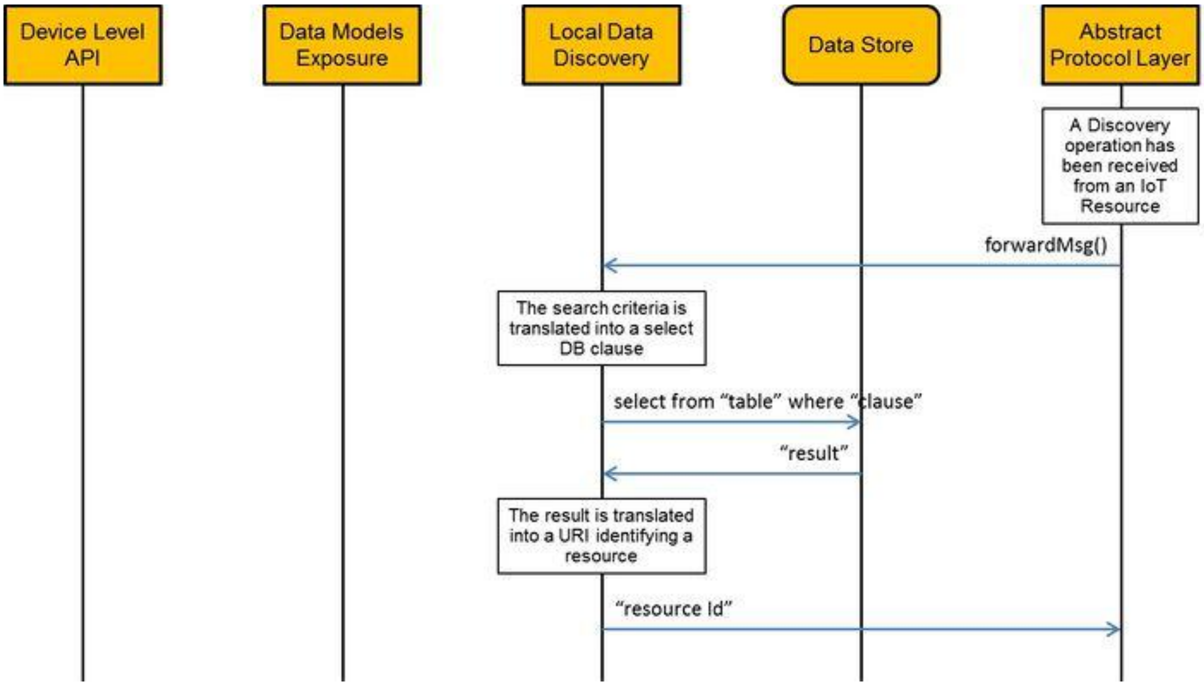
CRUD operation received from an IoT Resource

In the diagram shown by figure 5.1.5bis it is described the call flow when an Update request coming from an IoT Resource (via the Abstract Protocol Layer component) is directed to an ETSI M2M resource contained in the Gateway. The Data Models Exposure component executes the logic described in the diagram shown by figure 5.1.5, moreover it forwards the events and data to the Complex Event Processor/Data Aggregation Filter component and to the Publish/Subscribe Broker component which will treat these events and data as described in the Data Handling GE.



Update operation received from an IoT Resource

In the diagram shown by figure 5.1.6 it is described the call flow when a Discovery operation coming from an IoT Resource (via the Abstract Protocol Layer component) wants to search within the ETSI M2M resources contained in the Gateway. The Local Data Discovery component translates the search criteria contained in the request in the data format common to FI-WARE IoT, and then sends the corresponding operation to the Data Store . The result coming from the Data Store is translated into a resource ID by the Local Data Discovery component and sent back to the Abstract Protocol Layer component.



Discovery operation received from an IoT Resource

22.5 Basic design principles

- The Complex Event Processing API was designed with the main CEP concepts : CEP engine state, managing rules, and handling actions executed on rule triggering
- The API was thought to be ressource oriented, given the very nature of the Internet of Things. Thus The API was designed in a restful manner.
- NGSI interface is also part of the API and it was designed in a restful manner. The Data Handling GE has to keep technical consistency on its whole API.

23 FIWARE ArchitectureDescription IoT Gateway ProtocolAdapter

You can find the content of this chapter as well in the wiki of fi-ware.

23.1 Copyright

23.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

23.3 Overview

The Protocol Adapter GE deals with the incoming and outgoing traffic and messages between the Gateway and Devices registered to be served by the gateway. It is capable of serving non-IoT devices, i.e. devices that do not support ETSI M2M (the specifications may be found at the following link [1]). These devices can be IP-based devices that communicates using the IP stack (IPv4 or IPv6), and in particular uses Embedded Web Services at the application layer. Embedded Web Services include REST communication with a binding to either CoAP or HTTP as application layer protocols. Devices can also be "legacy devices", meaning devices communicating using non-IP based protocols, for instance ZigBee Pro, or Z-Wave.

The Protocol Adapter GE terminates these device specific protocols and translates them to a uniform internal API. The API exposed handle capabilities to read and write to the resources, as well as IoT specific management and configuration services such as resource discovery consisting of both look-up and publication.

In particular the ZigBee Protocol Adapter provides a communication conduit into a PAN(s) (Personal Area Network). It supports a mechanism whereby gateway can interact with individual ZigBee nodes to exert control over or to obtain data from those nodes or conversely a mechanism whereby the nodes can communicate some information to the gateway.

23.4 Basic Concepts

An overview of the Protocol Adapter GE is provided below, and is followed by an identification of the interfaces.

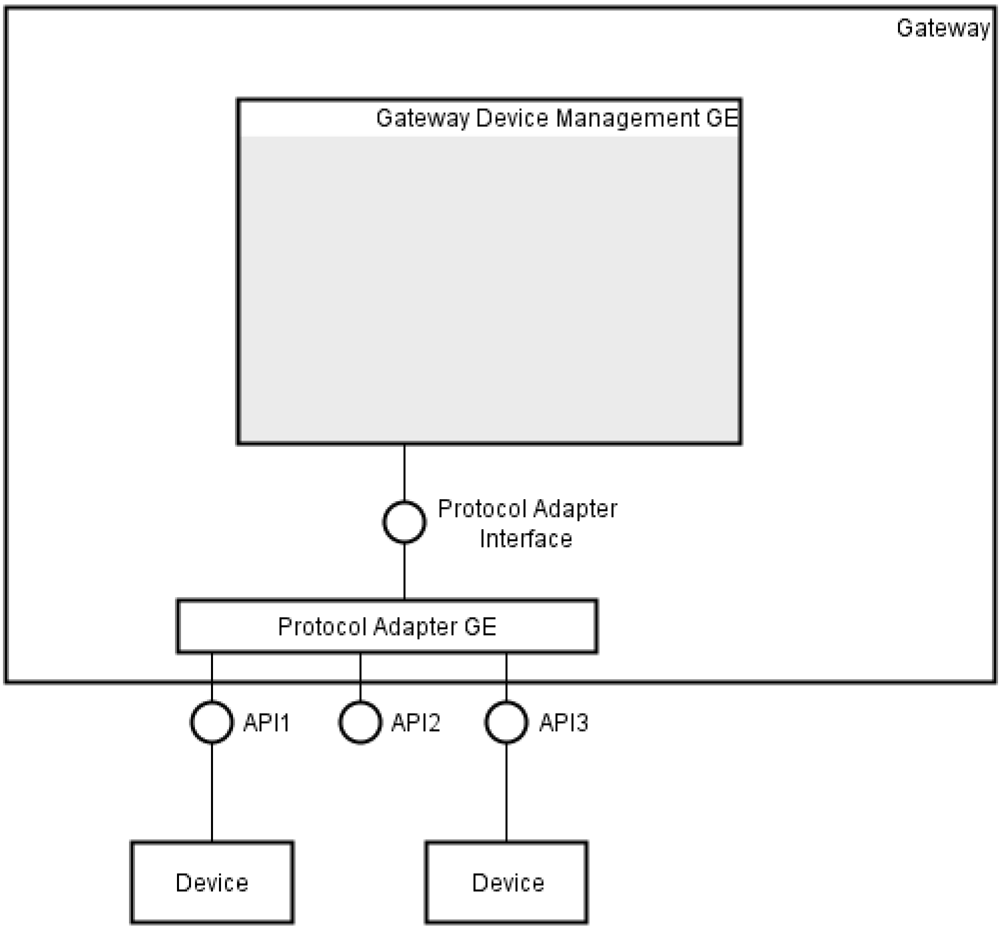


Figure 1: Protocol Adapter Architecture

There are two interfaces to the Protocol Adapter GE.

The southbound APIs provides the gateway external interface to non-ETSI M2M devices hosting sensor and actuator resources. The supported interface for the first release is IETF CoRE. For the next release will be added the ZigBee interface.

The Protocol Adapter Interface is a communications protocol to the Gateway Device Management GE, based on IETF CoRE. This interface can be used for initiating subscriptions to resources and receiving notifications from resources that have been tasked with subscriptions, read or write resources that are determined to be online, publishing resource capabilities in the Resource Directory, or querying devices for its resources.

Figure 2 shows the Fiware implementation of the Protocol Adapter GE.

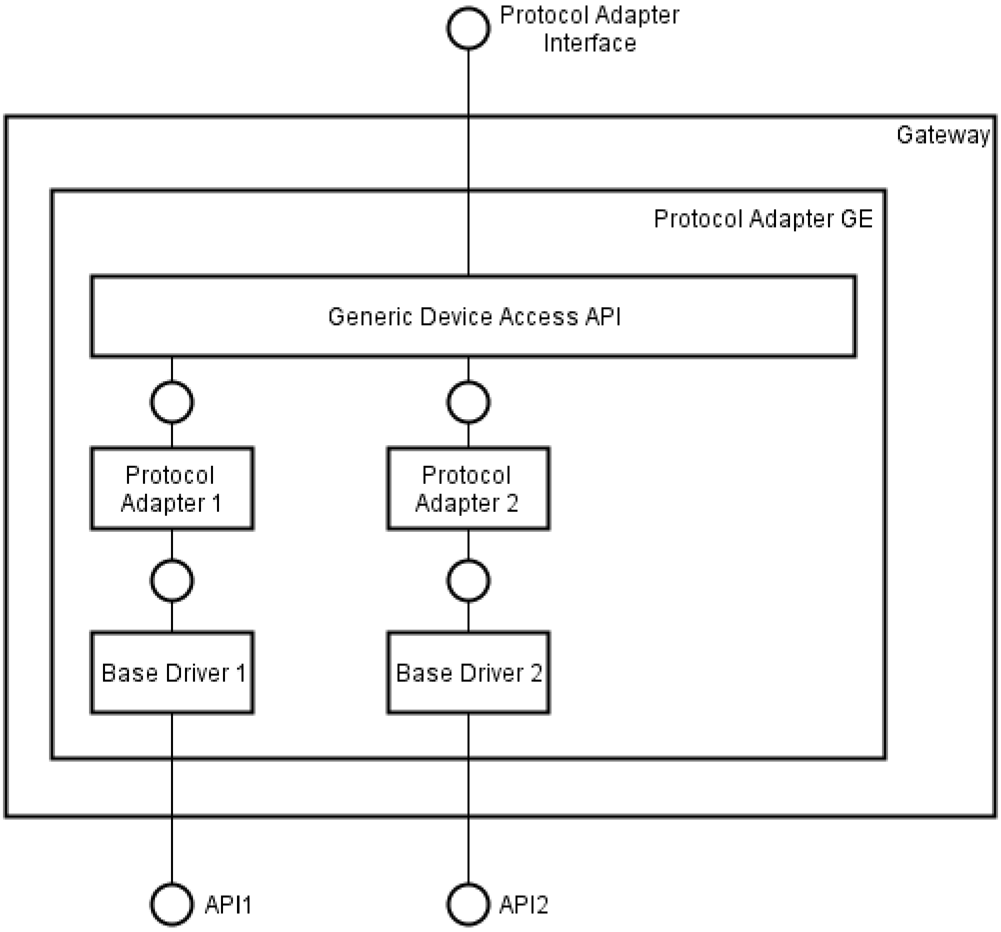


Figure 2: Protocol Adapter GE internal architecture

23.4.1.1 ***Basedriver***

The Base Driver is the low-level API for legacy devices (implementation of the protocol stack). Base Drivers handle device discovery and access to sensor and actuator resources in a protocol specific way.

In particular the ZigBee Base Driver is based on the Network Device Gateway Specification defined into the ZigBee Alliance (ZigBee document 075468r35 may be found at this link [2]). The included operations in this specification are:

- operations to read and write attributes, and configure and report events;
- macro operations for network and service discovery;
- endpoint management;
- flexible start-up and network join operations;
- bi-directional communication mechanisms between ZigBee Base Driver and gateway

23.4.1.2 *Protocol Adapter*

A protocol adaptor is the glue between the generic device access API and a base driver. Via the base driver, it discovers devices, tracks events occurred on them and executes commands to actuate them. Therefore, a protocol adaptor is necessary for each protocol that the generic device API supports. Whether the protocol is standardized or proprietary does not matter. As far as the base driver is available and the protocol adaptor is implemented on top, the generic device access API is able to support the protocol and provide a unified way to access devices with the protocol. The Protocol Adapter shall be able to support: Device discovery. When a new device is discovered and gets available, it shall create and register this device as a service within the Protocol Adapter framework. When a previously discovered device gets unavailable, it shall unregister the corresponding service. Device parameter update. When a service parameter update occurs on a device, it shall update the corresponding variable on the device in the Protocol Adapter framework and trigger update for the corresponding device service. Device actuation. For each action in a service that a device supports, it should implement a protocol specific logic and put it as an action in device service that is registered in the Protocol Adapter framework.

23.4.1.3 *Generic Device Access API*

The Generic Device Access API (GDA) exposes a high-level, protocol agnostic API towards applications. For Fiware this is implemented so that the GDA exposes an IETF CoRE interface (web service interface) to the Gateway Device Management GE. GDA uses service schemas which are XML-files that describe the supported resources, i.e. the application profiles. This schema based approach make it possible to cover wide range of applications from media to health care.

23.5 Main Interactions

Figure 3 shows an example of device and resource discovery, and how to subscribe to resources using the Protocol Adapter GE. The Device Management GE and Protocol Adapter register listeners with the gateway framework (OSGi service bus) for new devices. When the basedriver finds a new device in the network it will register this device in the framework which in turn notifies the listening Protocol Adapter. The Protocol Adapter can now query the device for resources which are then mapped to a service schema. These resources are then made available to the Generic Device Management GE.

Subscriptions are also registered in the framework which then triggers the Protocol Adapter to start a subscription to the requested resource. A new update (e.g. change in sensor value) will send an update to the Generic Device Management GE.

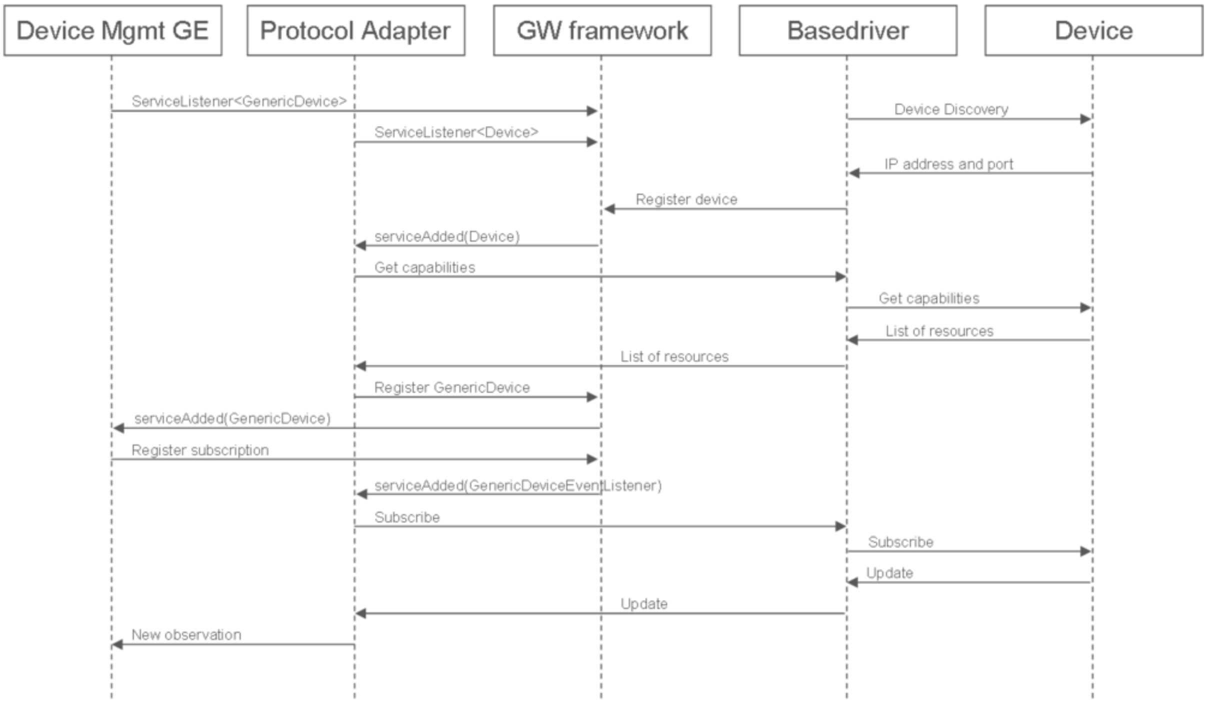


Figure 3: Device discovery and subscribe

23.6 Basic Design Principles

Projects deciding to implement support for additional protocols should make sure that there implementations cover the following functionality:

- Device discovery
- Device parameter update
- Device actuation

24 Architecture of Applications and Services Ecosystem and Delivery Framework

You can find the content of this chapter as well in the wiki of fi-ware.

24.1.1 Introduction

The Generic Enablers for the Apps Chapter together build an ecosystem of applications and services that is sustainable and fosters innovation as well as cross-fertilization. In particular the Apps Generic Enablers supports managing services in a business framework across the whole service live cycle from creation and composition of services to monetization and revenue sharing.

A couple of basic enablers are important to realize the vision of such a service business framework which enables new business models in an agile and flexible way:

- A Store, which allows to offer services for consumers as well as developers of future internet applications.
- A Marketplace, which allows to find and compare offerings from different stores and provides further functionality to foster the market for future internet applications and services in a specific domain.
- An SLA Support, which monitors and evaluates runtime data according to the agreements of service levels.
- A Revenue Sharing System (RSS Engine), which allows the calculation and distribution of revenues according to the agreed business models.
- A set of Service Composition enablers, which allow to compose existing services to value added composite services and applications, which can be monetized in the Business Framework.
- A set of Mediator enablers, which can be used to achieve interoperability between future internet services and applications and also allow to interface to existing enterprise systems.

This set of self-contained enablers representsonly an initial starting point for a future business framework. It is expected that supplemental enablers (e.g. for contracting, quotation ...) will be developed outside of the FI-Ware projects.

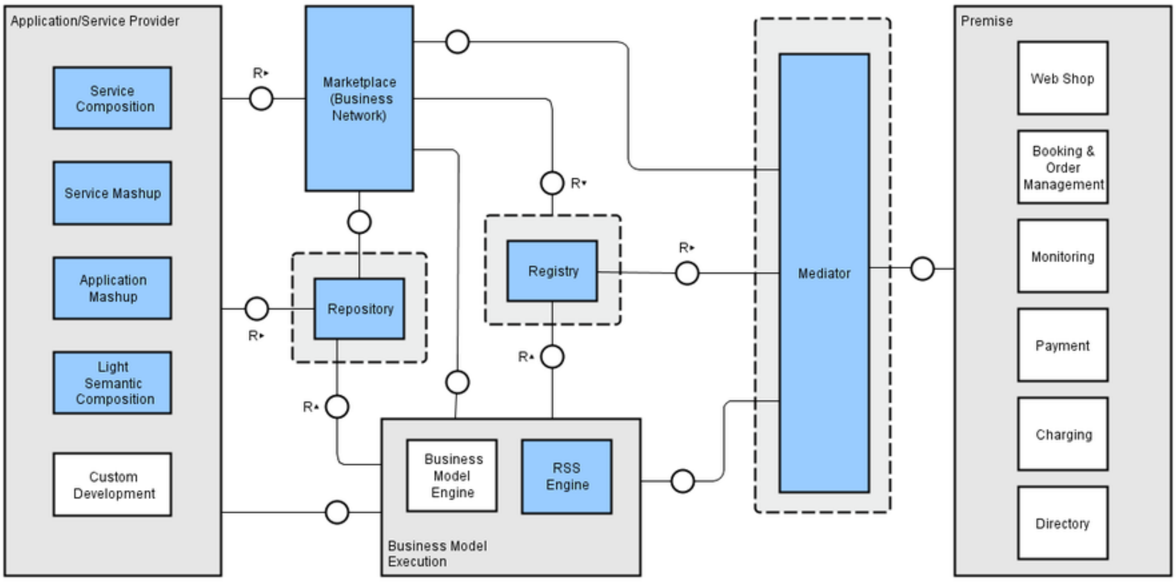
The Business Framework is heavily relying on USDL as common uniform description format for services which does not only focus on technical aspects of service, but also on business aspects as well as functional and non-functional service attributes.. USDL itself is not a Generic Enabler, since it is a data format and vocabulary specification. It will be introduced as an Open Specification, which is used by different enablers in their provided and consumed APIs.

The Applications and Services Generic Enablers are named according to their main functionality which is different from the role names introduced in the FI-Ware Vision. While the role names (Aggregator, Broker, Gateway ...) are used to describe the stakeholders of the service ecosystem in an abstract way, the enablers names are referring to software components.

```
Remark: Due to internal changes in the project consortium, the Store
component has suffered a delay and its description is still missing.

At present, UPM is working on it and a detailed description of the
Store component will be delivered shortly.
```

24.1.2 Architecture Overview

The following diagram gives an example of how the generic enablers can be combined to form a concrete architecture for a Service Business Framework. FI-WARE Generic Enablers are marked by blue boxes. The Business Model Engine is a a place holder for one or more GE to be defined by the Open Call partner who will address this specific problem space.



Apps and Services Ecosystem Architecture Blueprint

The service or application deployment into the cloud or the Internet of Things infrastructure is concern of the infrastructure provider. The monitoring of data for SLA and Revenue is provided by the GEs of other chapters.

It is important to understand that all implementations of Generic Enablers in the FI-WARE platform will only be used within the business framework as it is described in their service description (in USDL). It is the responsibility of the service provider to host the service on his own platform or to make use of FI-WARE Cloud services. The provider needs to provide interfaces to accept commercial transactions such as a customer/consumer buying a service on the store and has to do all necessary preparations in order to let the customer actually use the service as described in the contract. The provider also needs to deliver status information to the Revenue Sharing component. Other monitoring input sources for e.g. SLA tracking might be available on provider side.

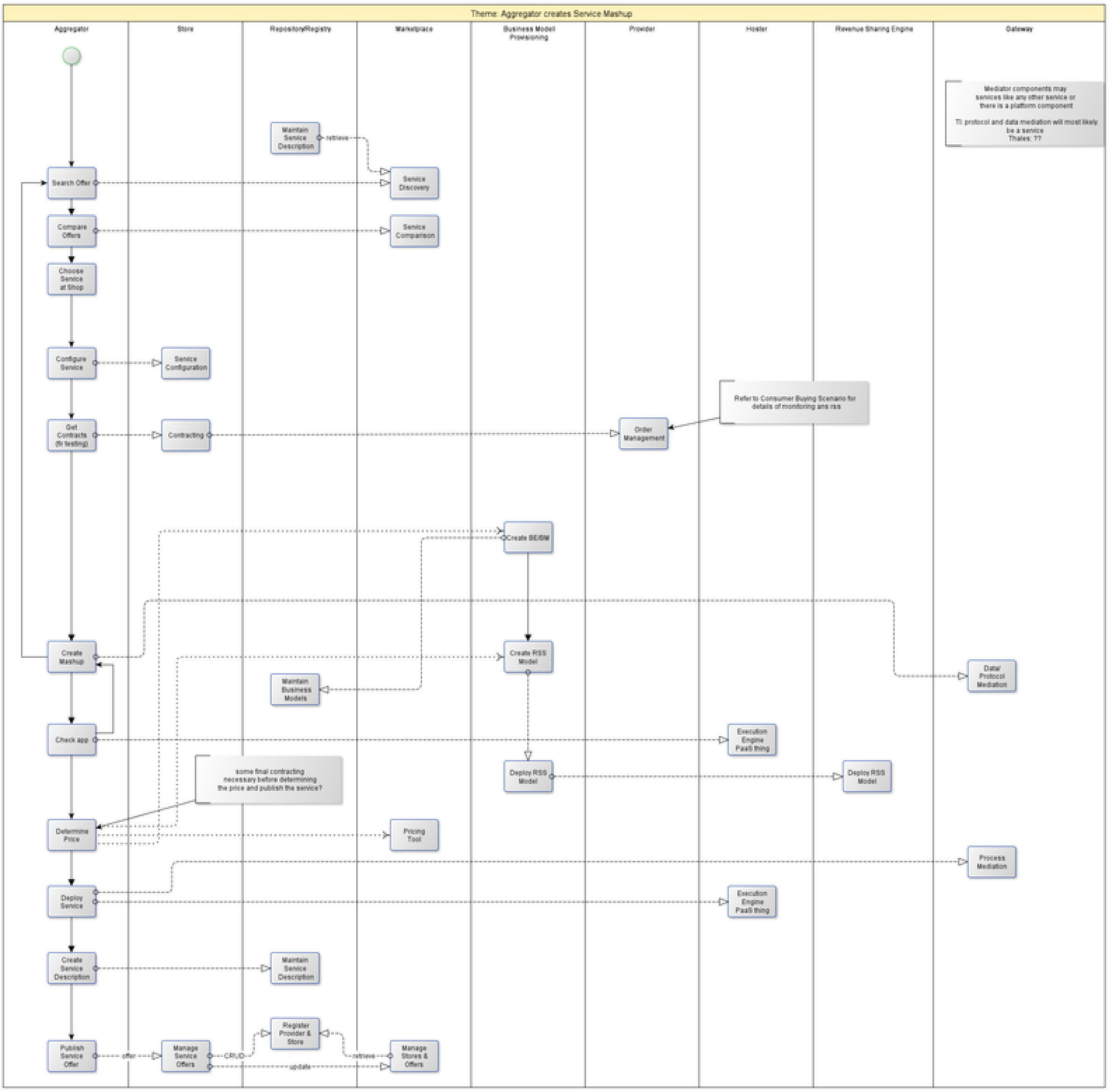
The architecture given here is only a blueprint showing an example of how the Apps&Services GE can be used to form an encompassing target platform. In a real use case domain the architecture might look different, according to the individual needs of the use case. However, it is important to recognize that the interplay among the GE and other application platform components is ensured by the FI-Ware Open Specifications which take care of their communication and interaction. The Linked USDL service description is responsible for sharing and referring to necessary information about the involved services, which occur in the domain specific use case.

24.1.3 Use cases

The FI-WARE generic platform consists of a number of Generic Enablers that can be used to instantiate a concrete platform for a certain FI-WARE Use Case domain. The individual Use Case platforms might differ from each other according to the concrete number of GE and their customization. Also additional domain-specific system components are added. The following generic scenario diagrams show how the GE of the Apps and Services Chapter are typically used in a sample Use Case. We provide scenario diagrams from the perspective of a Service Aggregator providing services and from the perspective of a Consumer buying and executing services.

The Service Aggregator is a provider that creates composite services or mashups from existing services in the Business Framework and offers these as services or applications again.

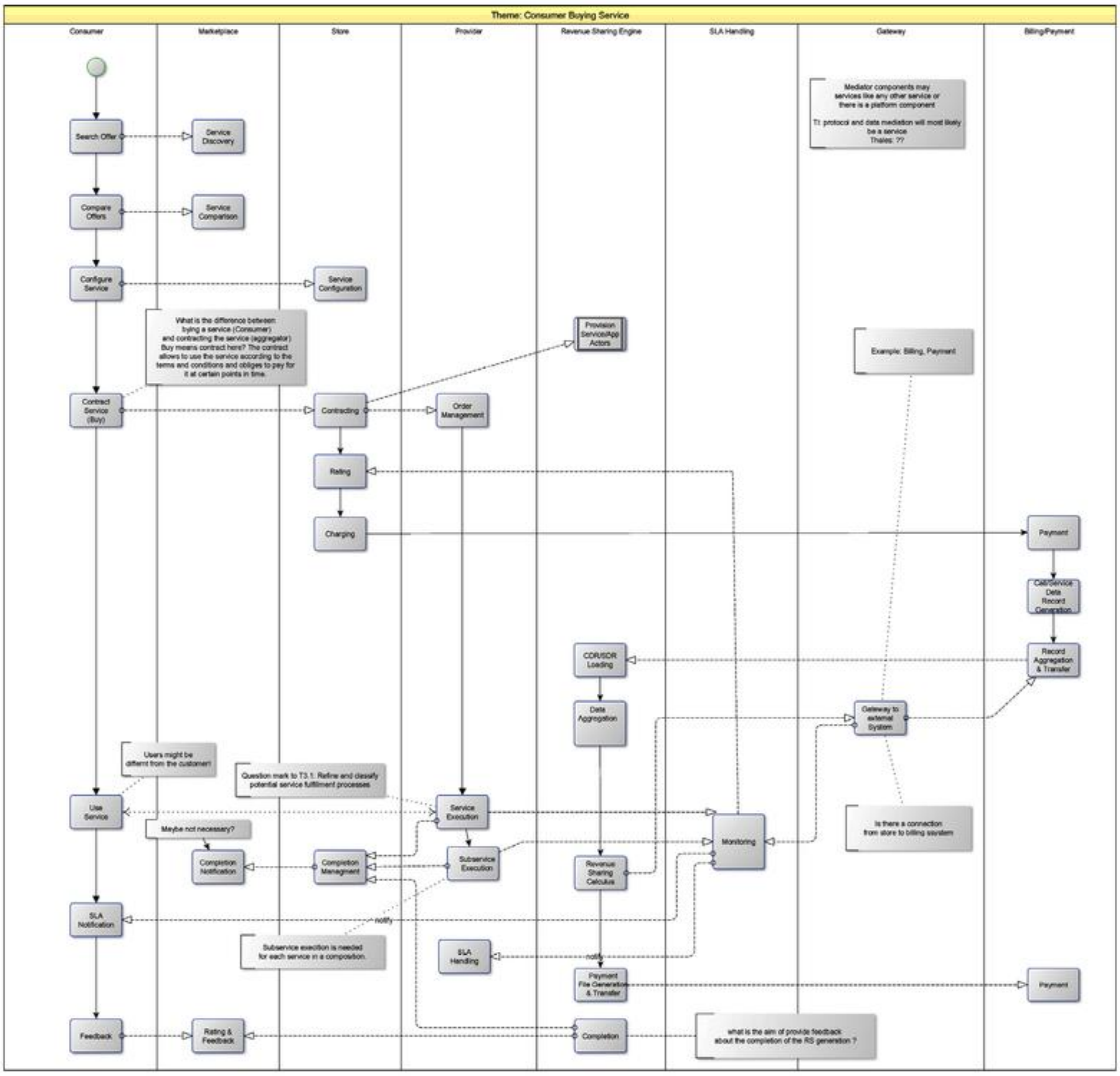
This Theme describes core composition and mashup capabilities on example of the Aggregator Creates Mashup end-to-end process and shows the interaction of a number of generic enablers from the Apps chapter. Other supported kinds of composition like web service composition or event-based composition are expected to be covered by the introduced end-to-end process as well, where necessary, by introducing slight deviations. The used functionality is described in more detail in the corresponding EPICS.



Aggregator creates mashup theme

The Consumer use the Business Framework in order to discover, examine, buy and execute services in a FI-Ware platform.

This Theme describes the Consumer Buying Service end-to-end process and shows the interactions of generic enablers from the Apps chapter. The used functionality is described in more detail in the corresponding EPICS.



Consumer bying service theme

24.1.4 Unified Service Description Language

USDL is the Unified Service Description Language, which is used to share inforamtion about services and apps between the different GE.

- FIWARE.ArchitectureDescription.Apps.USDL

24.1.5 Architecture description of GEs

- FIWARE.ArchitectureDescription.Apps.Repository
- FIWARE.ArchitectureDescription.Apps.Marketplace
- FIWARE.ArchitectureDescription.Apps.Registry
- FIWARE.ArchitectureDescription.Apps.RSS

- FIWARE.ArchitectureDescription.Apps.Mediator
- FIWARE.ArchitectureDescription.Apps.Store
- FIWARE.ArchitectureDescription.Apps.ServiceComposition
- FIWARE.ArchitectureDescription.Apps.ServiceMashup
- FIWARE.ArchitectureDescription.Apps.ApplicationMashup
- FIWARE.ArchitectureDescription.Apps.Light-weightedSemantic-enabledComposition

This list of enablers is not conclusive, and may be extended in future phases of the FI-WARE project. For project management reasons and limited resources, some of the mentioned roles in the FI-Ware Vision, such as the SLA monitoring, business elements and models, and multi-channel/multi-device support. Some of these topics will addressed in projects extending FI-WARE core consortium by Open Calls, depending on incoming feature requests of the FI-PPP use case projects.

24.1.6 Interfaces to other Chapters

It is intended that any Generic Enabler implementation can be offered within the Business Framework. We also encourage to implement the Business Framework using Generic Enabler implementations of other chapters. The following subsections give some indications of GE implementations that might be relevant.

24.1.6.1 *Data*

GE implementation for Big Data can be used to realize large distributed service description repositories and registries. The Complex Event Processing and Publish Subscribe enablers can be used for gathering monitoring data for SLA monitoring and Revenue Sharing.

24.1.6.2 *Security*

Within a Business Framework implementation there is the need for a common user identification mechanism across the different components. This could be achieved by the IdMaaS enabler of the security chapter.

The Security Monitoring enabler can be used to ensure compliance to the security requirements of Business Framework.

The Privacy enabler can be used to realize privacy according to privacy where consumer related private information is captured and stored.

24.1.6.3 *Cloud*

Cloud enabler implementations can be used to host parts of the Business Framework in the cloud in order to provide scalability and cost efficient operation.

The Cloud Object Storage could be used to store artefacts required by the Business Framework components.

24.1.6.4 *Internet of Things*

If a Business Framework implementation relies on Sensors and actuators in the Internet of Things the can use the IoT enablers implementations.

If a service that is commercialized via the Business Framework, the service description has to reflect this somehow. Therefore the service description need to be extended by parts describing the specific IoT requirements (needed resources, specific SLA, ...). This description is derived from the exposed capabilities of the IoT enablers.

25 FIWARE ArchitectureDescription Apps USDL

You can find the content of this chapter as well in the wiki of fi-ware.

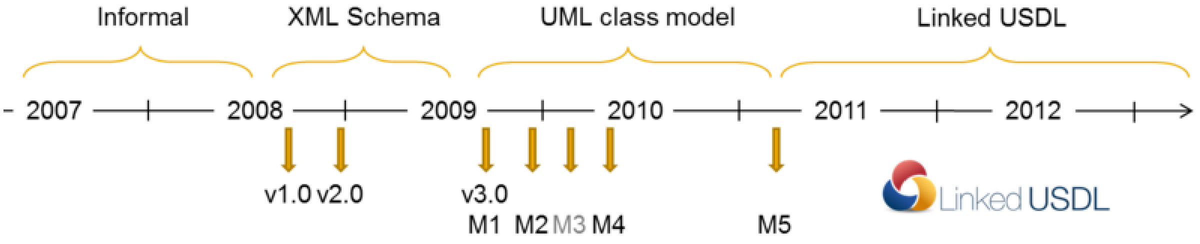
25.1 Overview

USDL (Unified Service Description Language) is a platform-neutral language for describing services. USDL is building a layer on top of technical service descriptions such as WSDL, which captures the information necessary to manage services in the business framework across the whole service lifecycle.

Since USDL is just a data format and a set of vocabulary definition, we introduce it into FI-WARE as an Open Specification. Because of the fundamental role for the business framework, we will outline the rationale and concepts of USDL in this page as part of the architecture description of the Application and Services Ecosystem chapter.

In the section USDL 3.0 Specifications M5 of Materializing Applications/Services Ecosystem and Delivery Framework in FI-WARE we outlined the necessity of a unified service description language, and described the power of USDL. Within FI-WARE we want to achieve adoption of USDL on a new level. In reflection of the work in the W3C USDL Incubator Group and the resulting recommendations, we will focus on easy of use and harmonization as well as integration with a broader range of standards.

Developing of USDL was always a collaborative and interdisciplinary approach. Initiated by SAP Research roughly 2 dozens of researchers from approximately 10 public funded projects and many research locations around the world contributed contributed their expertise from different research areas. Industrial partners such as Siemens, Attensity, HP, and others gave their feedback and recommendations in a W3C Incubator Group. As a next step of evolution the Linked USDL community was founded for broad and worldwide adoption and collaborative further development in an Open Source like community.



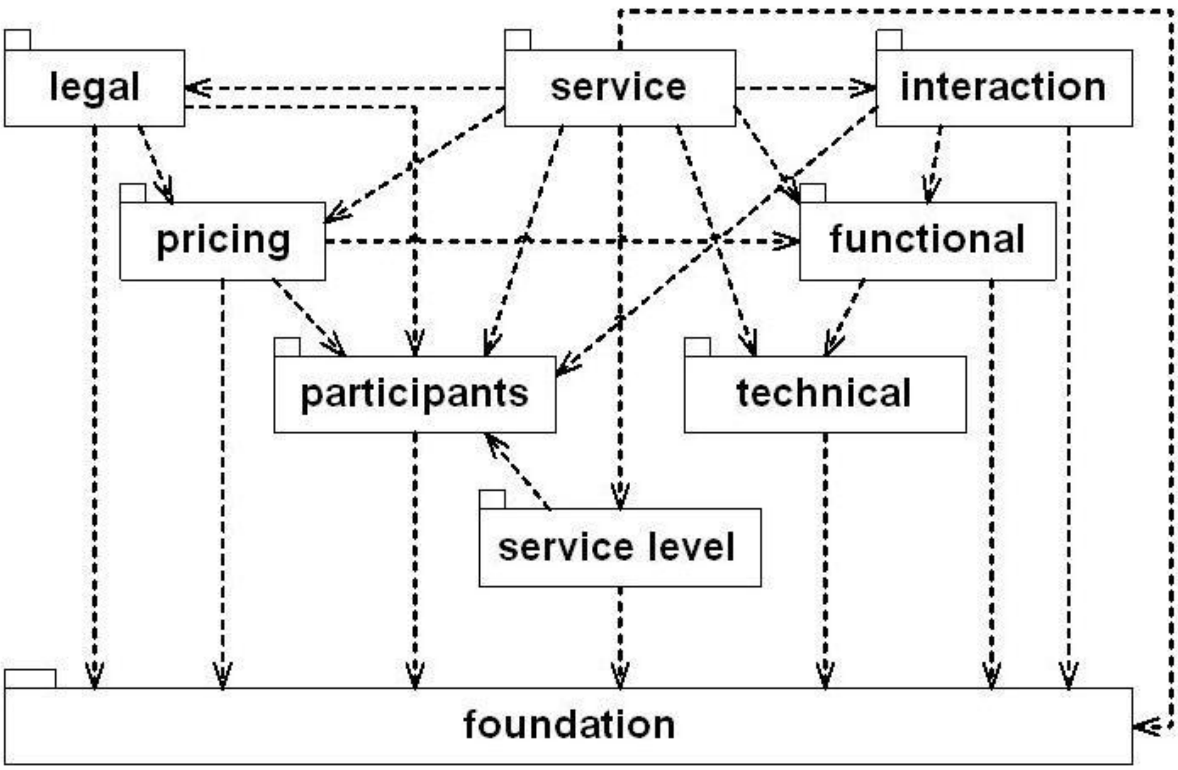
The FI-Ware approach will allow comprehensive service descriptions by employing the Unified Service Description Language (USDL) in its registry and repository. USDL builds on standards for the technical description of services, such as WSDL, but adds business and operational information on top. With its ability to describe both human and IT-supported services that not only implement business processes, but also tie in assets linked to contents and the Internet of Things, USDL is set apart from many of the related approaches mentioned above.

USDL is modularized to describe various aspects of services:

- Foundational Module: This module provides a common set of concepts and properties, such as time, location, organization, etc. that are used in all modules.
- Service Module: Describes the general information about the service type, nature, titles, taxonomy and descriptions.

- Participant Module: This module describes the participating organizations, contact persons and their role within the service fulfilment.
- Functional Module: This module contains information about the specific capabilities of a service, input/output parameters and constraints.
- Interaction Module: A module that describes the points of interaction and the responsible participants or participant roles in course of the service fulfilment.
- Technical Module: Describes how functions (capabilities) of a service are mapped to technical realizations of the service (e.g. WSDL operations, parameters, faults, etc.)
- Pricing Module: Contains information about price plans, price components, fences, etc. for a service.
- Service Level Module: The module which specifies service level agreements, such as time schedules, locations, and other constraints.

Legal Module: Contains information about the terms and conditions, IPR, licenses, and rights of use for the participating parties.



USDL modules and relationships

Necessary extensions to USDL for business, operational and technical perspective of supported front-end mashups and backend composite services will be identified and defined in the context of FI-WARE project. Furthermore, USDL extensions for artefacts from the chapters listed below will be considered:

- Internet of Things Service Enablement

Applications and services may have specific requirements of handling access to and management of sensor data and things within the “real” world. In many application scenarios, such as logistics, retail, or manufacturing, these capabilities will have an impaction even on business, as well as operational aspects. It will be necessary to investigate the extension of USDL according to aspects of the service lifecycle, which allows describing the relevant properties and capabilities of the Internet of Things in respect to accessing data as well as managing resources.

- Security, Privacy, Trust

The future internet platform will heavily depend on security, privacy and trust in any business relevant activity. Especially crowd-sourcing, flexible composition and mashups require adequate mechanisms to maintain security related characteristics. At the business framework level it is necessary to make security-specific information transparent. USDL needs to be extended by modules that allow describing the security related qualities and requirements of services and applications.

- Cloud Hosting (XaaS monetization)

Many services/apps will rely on specific hosting and cloud services, which will have technical as well as business relevant implications. The service description needs to contain the relevant information on hosting in order to cover these implications. Therefore XaaS description modules for describing various hosting modules need to be provided.

- Data/Context Management Services

For security monitoring and analytics of activities in the platform it will be necessary to emit and collect events and data for real-time or post-mortem analysis. For the various stakeholders it need to made transparent, when, how, and which data will be collected for which purpose and to whom. So the service description needs to be enriched by data and context information accordingly.

USDL will provide the means for integration of all these areas, FI-Ware generic enablers and the respective platform products. It allows tracing information and processes across the whole service lifecycle.

25.1.1.1 ***USDL Design Principles***

- rely on Web standards such as HTML, XML, RDF, …
- openness integrate into different contexts
- extensibility for addressing unforeseen or domain specific aspects
- flexibility to adapt to different domains via module variants
- harmonized with other information descriptions on the Web (existing schemas)
- ability to link-up with other information on the Web
- simplicity, easy to understand
- expressive power to specify at an appropriate level
- graceful degradation, keep the platform operational with partially incomplete and inconsistent descriptions

25.1.1.2 *Background*

There exists a plethora of existing service description efforts that can be grouped into different strands. Each of the strands has its own motivation and representation needs for capturing service information. The individual efforts can be attributed to the following criteria: (i) whether the scope of the effort lies in capturing IT or business aspects of services or the whole service system. (ii) the purpose of the corresponding effort, e.g., enabling of normative data exchange, facilitation of software engineering, or acting as reference model. (iii) Whether the effort is able to capture business network relationships between services. (iv) Whether the effort is standardized.

The first strand of service description efforts is the field of Service-oriented Architectures (SOA). SOA is a way of thinking about IT assets as service components, i.e., functions in a large application are factorized in stand-alone services that can be accessed separately. Because of their IT focus, most approaches limit their attention to the field of software architecture. Originally, several standards bodies specified roughly two dozens of different aspects which are collectively known as WS-* (incl. WSDL, WS-Policy, Ws-Security, etc.). Since one of the key components of a SOA is a service registry, the OASIS standards body introduced the concept of Universal Description, Discovery and Integration (UDDI), i.e., a specification for a platform-independent registry. UDDI services shall be discovered according to information such as address, contact, known identifiers, or industrial categorizations based on standard taxonomies. However, UDDI does hardly prescribe any schema for such information. As the concept of SOA matured, calls for support in software and service engineering increased. Hence, the OMG standards body dedicated its focus to software engineering for SOA, and, subsequently defined the Service-oriented architecture Modeling Language (SoaML). Finally, the multitude of description efforts and different definitions of SOA led to a Reference Model for Service Oriented Architecture (SOA-RM) from OASIS. Similarly, The Open Group drafts an alternative reference model in form of an ontology for Service-Oriented Architectures (SOA Ontology).

A second strand consists mainly of ontologies in the field of Semantic Web Services. The main goal of Semantic Web Services approaches is automation of discovery, composition, and invocation of services in a SOA by ontology reasoners and planning algorithms. The most prominent efforts are OWL-S and WSMO. Many similar efforts have surfaced in literature. With the many approaches around came the need to specify a reference model for semantic SOAs. Consequently, the OASIS is also about to specify a Reference Ontology for Semantic Service Oriented Architectures (RO-SOA).

The third strand is rooted in the rise of on-demand applications that led to the notion of software-as-a-service (SaaS), covering software applications (e.g., CRM on-demand) and business process outsourcing (e.g., gross-to-payroll processing, insurance claims processing) to cloud and platform services. The emphasis of service here implies that the consumer gets the designated functionality he/she requested together with hosting through a pay-per-use model. Thus, software-as-a-service is not synonymous with SOA. This difference triggered the Software-as-a-Service Description Language (SaaS-DL). SaaS-DL builds on WS-* to capture SaaS specificities in order to support model-driven engineering. The strand of SaaS also contains a standard, namely, the W3C recommendation called SML (Service Modelling Language). One anticipated use for SML is to define a consistent way to express how computer networks, applications, servers, and other IT resources are described or modelled so businesses can more easily manage the services that are built on these resources.

The fourth strand is driven by schools of business administration and focuses on capturing the purely economic aspects of services regardless of their nature (with less or no focus on IT services and software architectures). The German standard DIN PAS 1018 essentially prescribes a form for the description of services for tendering. The structure is specified in a non-machine-readable way by introducing mandatory and optional, non-functional attributes specified in natural language, such as, classification, resources, location, etc. The PhD thesis of (O'Sullivan) adopts a wider scope and contributes a domain independent taxonomy that is capable of representing the non-functional properties of conventional, electronic and web services. The work compiles the non-functional properties into a series of 80 conceptual models that are categorized according to availability (both temporal and locative), payment, price, discounts, obligations, rights, penalties, trust, security, and quality.

The fifth strand is also economic but draws attention mainly to describing Service Networks, i.e., the ecosystem and value chain relationships between services of economic value. The e3Service ontology models services from the perspective of the user needs. This offers constructs for service marketing, but in a computational way, such that automated reasoning support can be developed to match consumer needs with IT-services. The main focus of this work is to generate service bundles under the consideration of customer needs. The Service Network Notation (SNN) captures similar aspects to the e^3Service ontology. However, SNN is an UML model that can be analyzed for measurements of added value for each single participant as well as for the whole network optimization of value flows.

Finally, there are overarching efforts that concentrate on the bigger picture of service systems or service science also taking into account socio-economic aspects. Stephen Alter was one of the first to realize that the concept of a service system is not well articulated in the service literature. Therefore, he contributes three informal frameworks as a first attempt to define the fundamentals of service systems. The work of Ferrario and Guarino can be seen as a continuation and formalization of Alter's approach. Although differing in its main notions, they present a reference ontology for ontological foundations of service science which is founded on the basic principles of ontological analysis. In turn, this reference ontology forms the core part of the TEXO Service Ontology which extends it by ontology modules for pricing, legal, innovation, or rating information.

USDL is the only effort which covers IT and Business aspects, serves both a reference and exchange purpose, considers business network related information and is about to be standardized.

25.1.1.3 ***Relations to standards***

USDL is dedicated for business aspects of the services business framework and has no means of describing the technical aspects of services. Therefore USDL is meant to be used in conjunction with other Web standards such as WSDL etc. in order to address detailed technical information. In the case of WSDL, USDL refers to a WSDL description and can even map high-level interactions to elements of a technical service description in WSDL.

The standardization of USDL was discussed in the W3C USDL Incubator Group, which published the final report in September 2011. The common understanding and recommendation of the group members, was to keep the alignment with Web standards and go for a W3C standardization process. So the discussion with the W3C has started to establish a new standardization group in 2012. With respect to the W3C policies and processes, we estimate that the group could be in place in spring 2012.

25.2 Basic Concepts

25.2.1 Web of Data

"in recent years the Web has evolved from a global information space of linked documents to one where both documents and data are linked. Underpinning this evolution is a set of best practices for publishing and connecting structured data on the Web known as Linked Data." Bizer et al. Linked Data - The Story So Far. International Journal on Semantic Web and Information Systems (IJSWIS)(2009)

Linked USDL is a set of simple Linked Data vocabularies encompassing various modules of USDL for describing and linking all relevant aspects of doing business with services.

The core vocabulary contains fundamental aspects and core properties of services, service composition, interaction, service offerings, participants and legal constraints. Additional vocabularies are available for describing price model, service level profiles and security features. USDL can be complemented with existing vocabularies of the Linked Open Data world to describe various other aspects related to services.

Over the last decade, the service sector has become the biggest and fastest-growing business sector in the world. For the first time ever, it now employs most people worldwide. Various companies and research institutes have started to explore different aspects of the service sector to determine which services can be managed through IT and, being combined, lend themselves into value-added services and service networks. Linked-USDL provides the basis for sharing information about services in such envisioned service networks. Service networks need to be supported by an appropriate Web-based infrastructure consisting of distributed repositories, marketplaces, business model support, service communities and more.

Potential users and stakeholders for Linked USDL are

- Service providers
- Resellers, brokers
- Infrastructure providers
- Consumers

Various different stakeholders in the service business come together and work jointly on specifications and applications.

Linked Data is proven approach to put data in the Web and allow linking to other data much like linking Web pages. Data is expressed in a universal standard data format, which is easy to produce and consume.

25.2.2 Rationale

25.2.2.1 *Easy to use*

The entry barrier for applying Linked USDL should be as low as possible in order to enable also small companies describing their services with a reasonable amount of effort and resources and participate in the ecosystem. There is a balance between simplicity and expressive power to maintain. It must be possible to describe complex services without to sacrifice the simple cases.

25.2.2.2 *Simplify the process of further development*

In comparison to the Ecore object modelling approach, the Linked Data approach will make it easier to extend USDL with new aspects and domain specific variations.

25.2.2.3 *Reusing existing standards and existing tools*

By relying on existing and accepted standards, the USDL vocabularies can be concentrated on specific aspects and will be simpler and better to understand. Furthermore synergies with other communities can be leveraged.

25.2.2.4 *Linkable*

It should be possible to link aspects of a service description with various other information resources according to the overall life cycle of a service, which relate to each another.

25.2.2.5 *Platform neutrality*

USDL abstracts from platform (technology) specific aspects and covers communication of different components/actors on an abstract level of interactions.

25.2.2.6 *Domain independence*

USDL descriptions are generic and not per se tailored to a specific application domain. Nevertheless it contains elements (taxonomies, properties) to enable to express domain-specific information.

25.2.2.7 *Extensibility*

USDL is not complete in all aspects. It rather abstracts important aspect of business with services. A fundamental principle of USDL is the extensibility according to specific business and life-cycle aspects in order to allow more functionality in a certain context.

25.2.3 Data Model

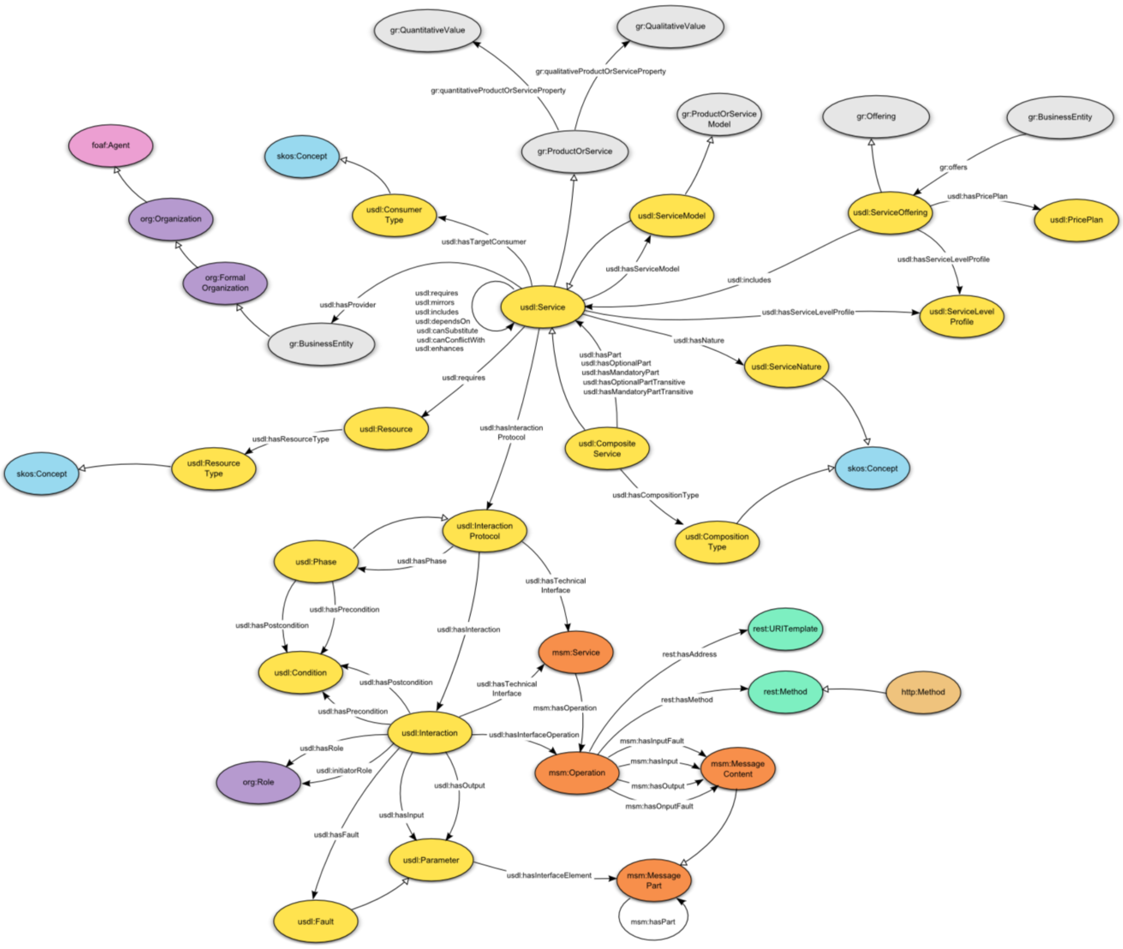
USDL is building on top of Linked Open Data principles. So the underlying data model is the graph model of RDF, which is standardized by W3C and supported by a wide range of tools (repositories, querying, programming language bindings, etc.).

USDL is reusing existing Linked Data vocabularies as much as possible in order to allow linking elements of USDL descriptions to other existing datasets. Examples of vocabularies currently used are FOAF, GoodRelations, DCTERMS, MSM, ORG, SKOS, Time, VCARD, sawsd, sa-rest, wl, and ctag. This allows the integration of data and processes on a global (Web) scale and allow applications (mashups) that integrate information across multiple sources.

USDL is not one monolitic RDF ontology. It rather comes as a collection of harmonized vocabularies, which express different aspects of the service business. The USDL Core module is the definition of basic concepts and properties for describing services, service models, composition, interaction, and classification. Further vocabularies complement the core vocabularies for describing other aspects, such as service level agreements, pricing, security, and platform requirements and resources (computing infrastructure, IoT enablement, network and devices). The supplemental vocabularies can be available in different flavors, depending on the needs of the application domain.

In the context of a specific technical realization of the platform, USDL can be serialized into a couple of serializations such as RDF/N3, RDF/XML, RDF/JSON for the developer convenience. The Generic Enablers using USDL are encouraged to deliver different serializations on request (content negotiation).

The following picture gives an overview of the USDL core vocabulary.



25.2.4 Use Cases Examples

25.2.4.1 *Telecom: Trading Services*

Problems

- No common format to describe the business side of technical services (e.g. WSDL)
- Marketplaces are 'islands' of services
- No solution to transfer services across marketplaces

The use of a common language such as Linked-USDL enables providers to transfer services across platforms

25.2.4.2 ***Cloud Services: IaaS***

Problems

- Many offerings in the wild
- No coherent description of services available
- No common marketplace

Comparison of offerings (price, SLA, capabilities, …) is very difficult for users Linked-USDL can help to put light into the dark and make Cloud offerings more transparent to the consumer!

25.2.4.3 ***Transport & Logistics***

Freight Forwarder's problems (FINEST use case)

- No common scheme for transportation services
- Planning of routes with many legs is cumbersome
- Main criteria are price, reliability and time
- Hard to find transport options if plan must be changed on the fly
- Vast amount of transport options is inaccessible
- Phone calls, paper work are still dominant collaboration means

A spot market and planning tool based on Linked-USDL logistic service descriptions would have an enormous effect.

25.2.5 Creating a community of practice

The definition of Linked USDL is not final. After stabilizing the core vocabulary, additional aspects as well as domain specific extensions and variants need to be supplemented. Therefore before going into a standardization process, Linked-USDL need to be adapted and proven useful by the stakeholders above within different use case domains. So it is an important goal of the linked-usdl.org to facilitate a community of users, providers, researchers, developers etc.

The following steps are identified for starting the community:

- Identification and invitation of stakeholders (research, development, service providers, consumers, ...)
- Providing more documentation (background and use case presentations, snippets, manifesto, etc.)
- Providing a basic infrastructure (repository, search) for testing and demonstration
- Apps showing value and attract people (such as a simple marketplace and additional mashups)
- Concept of managing/moderating the community with appropriate community tools
- Organizing events, challenges and the like

An alignment with other service description and standardization activities is also desirable.

25.3 Architecture

USDL itself is just a set of vocabulary specifications that is used to describe services in a business framework and platform relying on the concept of services. USDL descriptions are usually stored in distributed repositories to be used by the different components when necessary or transferred as part of the API of components of such a platform (e.g. between a marketplace and stores). It's purpose is that of a *lingua franca* within the framework.

USDL descriptions can be created manually by using dedicated editing tools. However, USDL descriptions or parts of them can also be generated automatically from information in existing catalogs and enterprise systems. For example a service store can provide an interface for marketplaces to deliver service descriptions in a USDL serialization from the specific native data structures of the store implementation (internal database schema). A specific USDL API for this generation will not be provided because existing RDF tools are sufficient for this purpose.

However, USDL implies a common infrastructure of basic services and tools such as editors, repositories, search engines and marketplaces. FI-WARE will provide reference implementations and open specifications for some of this base infrastructure components.

25.4 Design Principles

- **Unified Service Descriptions**

The central purpose of USDL is to describe services for stores and marketplaces as well as for the business relevant aspects in the runtime platform, e.g. monitoring service level aspects and revenue sharing.

- **Modular Design**

USDL follows a modular design approach. Not every concept or property needs to be specified in a USDL Service Description

- **Extensibility Principle**

USDL provides extensibility mechanisms to allow business and life-cycle specific aspects to allow more functionality in a certain context.

25.5 Open Specifications

25.5.1 Open API Specifications

USDL is a schema/vocabulary definition and does not provide any API

25.5.2 Other Open Specifications

USDL is used in many other projects outside the scope of the FI-PPP initiative. An open source like community was founded on linked-usdl.org in order to foster take up and collaboration. All Linked USDL vocabularies relevant for FI-WARE will be published and maintained by the Linked USDL community. For example currently there are four vocabularies complementing each other for describing different aspects of the service business.

- Linked USDL Core Vocabulary

- Linked USDL Pricing Vocabulary
- Linked USDL Service Leval Agreements Vocabulary
- Linked USDL Security Vocabulary

It is expected that these vocabularies will be complemented by additional topic specific or domain specific vocabularies coming from the Use Case projects.

25.6 Further References

BARROS, A. ; OBERLE, D.: Handbook of Service Description: USDL and Its Methods. Springer, 2012. – ISBN 9781461418634

26 FIWARE ArchitectureDescription Apps Repository

You can find the content of this chapter as well in the wiki of fi-ware.

26.1 Copyright

26.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

26.3 Overview

Together with the Registry and the Marketplace, the Repository is a core enabler of the FI-Ware Business Framework. The repository provides a consistent uniform API to USDL service descriptions and associated media files for applications of the business framework. A service provider can use the Repository to publish the description of various aspects of the service according to a unified description language. Wheras the Repository is used to publish service descriptions (service models), the Registry is used for storing runtime information about concrete instances and their configuration settings.

USDL is used in its Linked Data version "Linked USDL". Documentation can be found at <http://linked-usdl.org/> . Information about the FI-Ware Platform is available at https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/Main_Page . USDL describes services on a metadata level and can refer to supplemental resources of any media type. Therefore the repository must be able to store resources in arbitrary formats. The RDF datamodel of USDL allows to refer to entities of the service description via the resource URL. Therefore Linked-USDL is already well prepared to allow the distribution of service descriptions all over the Internet.

26.3.1 Target usage

The Repository is a place to store service models, especially USDL descriptions but also other models required by components of the overall delivery framework (e.g. technical models for service composition and mashup). The repository provides a common location for storage (centrally or distributed and replicated), reference and/or safety.

The use of a repository is required in order to appear at the marketplace or other tools referring to a number of central repositories for information relevant for interoperation of the enablers and roles within the FI-Ware platform. The repository contains published descriptions which can be utilized by any component in respect to privacy and authorization constraints imposed by the business models. Usually a repository is under control of an authority and usually is keeping track of versions, authenticity and publication dates.

26.3.1.1 *User roles*

- The Provider creates services and has an original description describing basic service information as well as technical information. He needs to upload and publish service descriptions on the repository in order to make them available to other components of the platform, such as the Shops/Stores, Aggregators, etc.

- The Aggregator will use for example technical and service-level information of existing services in the repository in order to do a compositon of a value added service or application. So for example, in order to give a valid statement about the availability of the new composed service the availability of each contained service needs to be regarded. The Aggregator also needs information about the technical interfaces of a service in order to develop code to call them correctly . Service descriptions for the newly created composite service can be uploaded and published to the repository again.
- The Broker needs all kind of business relevant descriptions of services, such as general descriptions, business partners, service-levels, and pricing, to be presented in the shop/store. Also technical information can be required, on a level to be able to do comparisons between services for the consumer.
- The Channel Maker needs detailed information about the channel to ensure the proper channel creation or selection. Further a channel may require embedding or wrapping the service so it can be accessed by the user through the specific channel. Various channels and devices such as Web (browser), Android, iOS but also global as well as local social networking and community platforms such as Facebook, LinkedIn, MySpace, Xing, KWICK! might be supported.
- The Hoster requires information on service-level descriptions, deployment and hosting platform requirements to provide the necessary infrastructure in a reliable and scalable way.
- The Gateway will use information about technical interfaces to provide data, protocol and process mediation services. The gateway also provides services for mediation towards premise systems outside of the FI-Ware platform.

26.4 BasicConcepts

26.4.1 Web Citizen

The repository is relying on Web principles:

- URI to identify resources
- consistent URI structure based on REST style protocol
- HTTP content negotiation to allow the client to choose the appropriate data format supporting HTML, RDF, XML, RSS, JSON, Turtle, ...
- Human readable output format using HTML rendering ('text/html' accept header) including hyperlinked representation
- Use of HTTP response codes including ETags (proper caching)
- Linked Data enablement supporting RDF input and output types

26.4.2 Open Distributed Architecture

The Repository Open Specification has to be seen as a specification of the repository abstract functionality. There can be many technologies used to implement the functionality. Often the repository protocol is implemented on top of a Web content management system.

Also we envision a large number of repositories containing service descriptions, which also might to refer to descriptions in other repositories. Repositories can be hosted by the provider or a provider may use repository services of platform providers. The latter might be an alternative for small sized providers, which don't want to provide an own infrastructure.

The service descriptions in a repository are typically used by different other components of the platform, such as service stores or marketplaces, by extracting information needed for the specific functionality.

26.4.3 Data Model

The repository is structured into core objects, which are *resources* and *collections*. These objects constitute also the granularity of access control. Collections are containers for storing resources, which are typically used to maintain all resources belonging to a certain service description in one place.

26.4.3.1 ***Resources***

The resources are mainly the USDL service descriptions themselves as well as complementary media files that are used within the service descriptions.

26.4.3.2 ***Collections***

A collection is a container for collecting resources. Multiple collections can be used on the repository for various purposes. Collections can be nested and may provide versioning of the resources. Collactions are used to keep all content that is locally referred from the service descriptions together in one place. For example a service description often has additional documentation, depictions and other collateral information, which can be bundled together in one collection.

26.4.3.3 ***Recipes***

Recipes are virtual containers selecting resources from different collections.

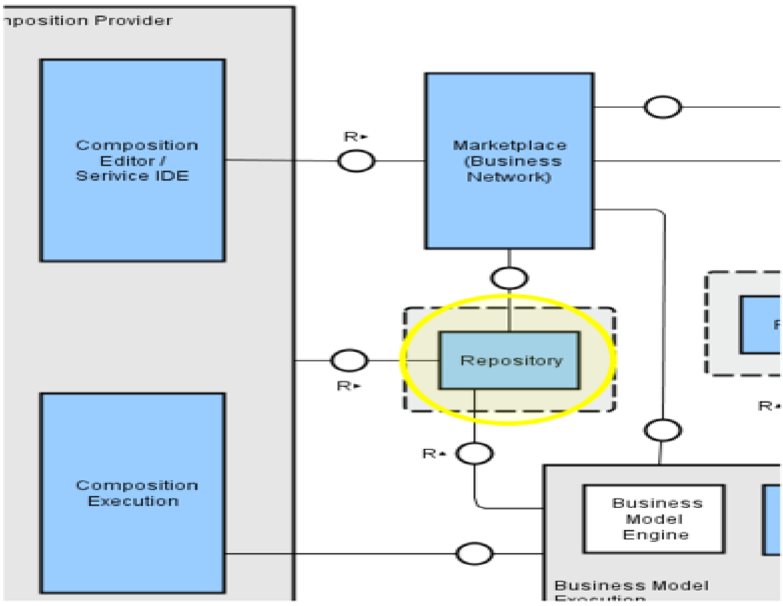
26.4.4 Content Negotiation

For optimal interoperability and flexible use, the repository should be able to deliver the results of an operation in multiple formats. HTTP content negotiations should be used to let the client choose an appropriate content type. Basic content types (mime-type) are

- HTML - to deliver the results in hyper-linked HTML that can be rendered directly in a Web Browser
- RDF - various RDF serializations for processing in applications
- JSON - Javascript Object Notation for easy processing in a mashup environment.

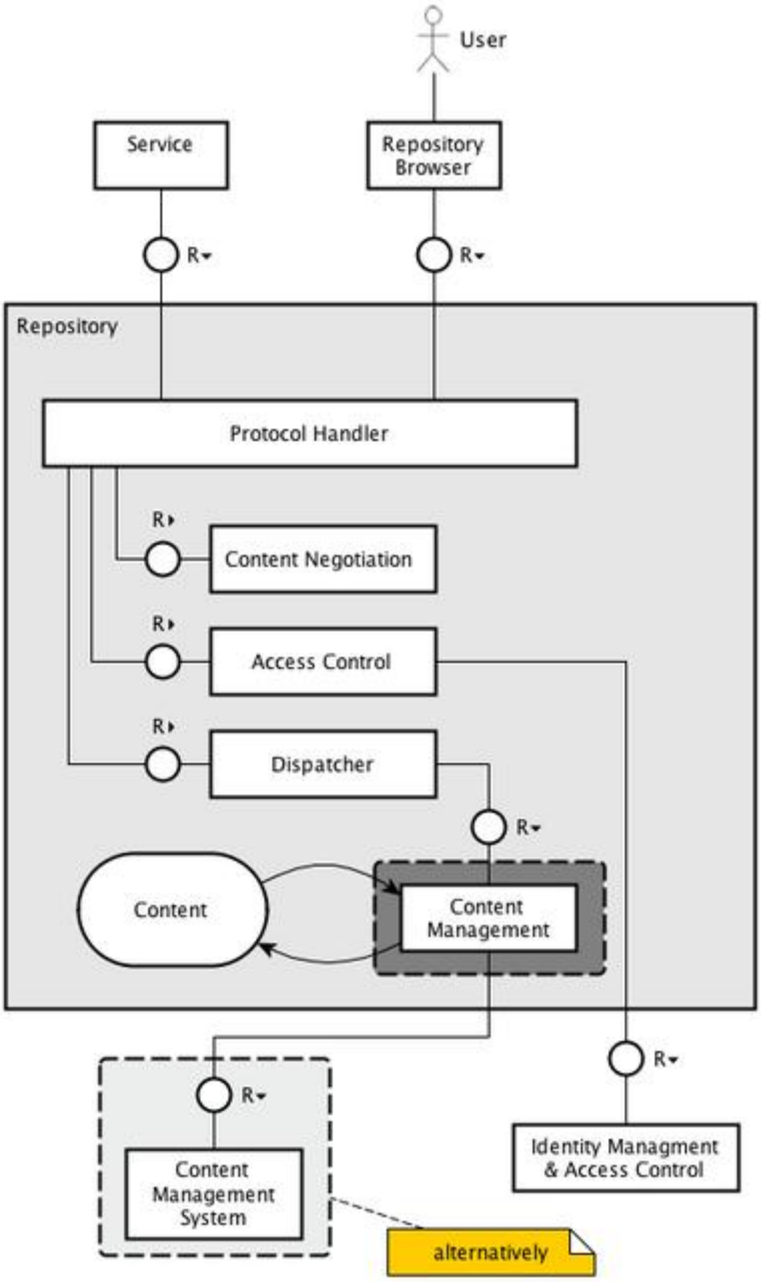
26.5 Repository Architecture

The repository GE is used by various other GE within the FI-Ware platform. Namely Marketplace, Store, Composition Environment as well as SLA monitoring and Revenue Sharing can access repositories to retrieve detailed information about a service or application. The composition environment for example can retrieve available service offerings for composition from the marketplace. In order to get detailed information about the respective services, the repository API is used. Finished composite services or applications in turn can be described in Linked USDL and published in a repository. New offerings for the service can be posted to the marketplace. Similarly the mediation GE can get details about a service to be mediated from the repository and push back mediator proxy services for a complex mediation type, to be reused by many applications. The repository is also used to store business models according to composite services and applications, which will be used by the business model execution environment and revenue sharing system.



Repository in the context of the Business Framework

Besides the FI-Ware platform also Future Internet applications or composite services on top of the FI-Ware platform can use the repository as a service for their own purpose. An example of the inner architecture of the Repository is shown in the following diagram.



Example high-level architecture of a repository implementation

The architecture shown here is only a blueprint for possible implementations of the repository and show the functional components, necessary to realize this functionality. There are many technology options for a concrete implementation, depending on the context and application domain and its nonfunctional requirements. Since the requirements according to repository size, workload and other parameters can be quite different, there is no obvious all-encompassing implementation solution. The implementations can span very simple ones, which provide only few extensions to a standards Web service to very sophisticated ones that utilizes enterprise content management systems (e.g. based on the "CMIS - Content Management Interoperability Services" standard).

The repository only stores and provides access to serice descriptions. Since there is no common standard for versioning, and the requirement according to versioning may vary depending on the use case scenario. We do not require version control from a repository implementation, although a real implementation can provide versioning models and mechanism (e.g. using the capabilities of the underlying CMS system).

Also there is no requirement regarding consistency checking of the service descriptions in the repository. The applications themselves have to ensure that the descriptions are consistent. All clients of a repository have cope with incomplete and inconsistent information by default. This reflects the architecture of the Web, where also no consistency commitment of the pages on different Web servers can be made. To ensure integrity additional measures have to be taken.

26.5.1 Technical Interfaces

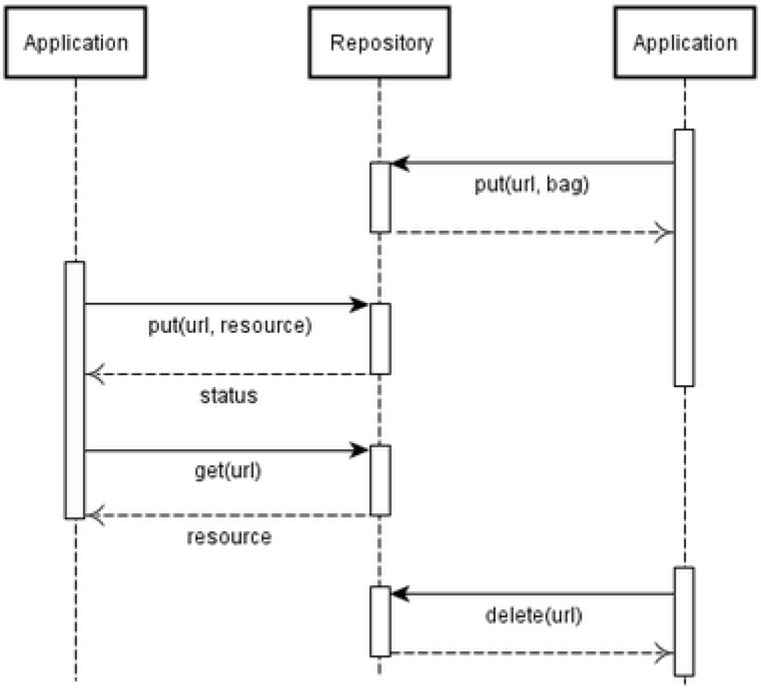
- FIWARE.Interface.Apps.USDLRepositoryRest - A very simple REST based protocol based on plain HTTP.

26.6 Main Operations

The Repository operation protocol is kept very simple. It basically provides operations to get and put resources, such as service descriptions and media content. Additional operations are used to structure the repository into collections of resources.

26.6.1 Managing Resources

The core functionality of a repository is to store resources and retrieve them when necessary. Further resources sometimes need to be updated and eventually deleted. The following diagram shows an example sequence of resource management operations of a repository.



Example sequence of resource management operations

The ***Get Resource*** operation can be used to retrieve a resource from the repository. This operation delivers the actual content of the resource and/or metadata about the resource, such as the media type, creator, or modification date, depending on the used technical interface. The following parameters need to be exchanged:

- *resource identifier* - Resource identifier of the resource to be returned.

If only information about collections is requested, the collection identifier is used instead of the resource identifier.

- *collection identifier* - Identifier for the collection, which contains the resource.
- *resource* - Resource which will be returned
- *media type* - Media type of the resource which will be returned.

The ***Put Resource*** operation is used to store a new resource into the repository or update an existing resource with the same resource identifier. The repository should take precautions to provide inconsistent changes due to concurrent access. The following parameters need to be exchanged:

- *resource identifier* - Identifier which contains the resource.
- *collection identifier* - Identifier which denotes the collection into which the resource will be put. The collection can be a part of the resource identifier, if for example URL paths are used to identify a resource.
- *resource* - the content of the resource to be stored into the repository

In order to delete a resource irrevocably from the repository the ***Delete Resource*** operation is used. The following parameters are exchanged:

- *resource identifier* - Resource identifier of the resource to be deleted.

26.6.2 Managing Collections

Collections are used to put a structure into the repository. In order to easily access parts of the repository, it allows clients to get information about the contents of individual collections.

The ***Create Collection*** operation creates a new collection in the repository, containing the necessary details such as owner, policies, and other metadata attributes. It requires the parameter:

- *description* - Description of the collection to be created, which contains the location path within the repository and administrative data such as creator and access policies.

To get the details and contents of a collection the ***Get Collection*** operation is used. The collection information contains information such as owner, policies, textual descriptions, dates, versions, number of resources, and more. The level of detail of the description may depend on the authorization level of the requester. The following parameters can be involved in the operation:

- *collection identifier* - Collection identifier for which a description is to be returned.
- *filter* - Optional filter expression to select the properties to be filtered.
- *description* - Returned collection description containing information according to the filter expression.

A collection in the repository can be deleted with the ***Delete Collection*** operation. This operation can only be successful for a requester that has the appropriate authorization. The delete operation requires the identifier of the collection as input. After this operation the collection is no longer accessible for clients. Only one parameter is necessary:

- *collection* - Collection identifier for the collection to be deleted

26.6.3 Listing Content

The ***List*** operation lists collections and/or resources contained in the repository, which are accessible by the user. This operation usually is needed for a repository browser and maintenance tool as well as an editor tool in order to select the resource to be maintained.

The operations using the following parameters: No input parameters are required. However, for practical reasons it might be useful to restrict the list of collections and resources by specifying the number of results and the starting offset.

- *collection* - Collection for which the list is restricted to.
- *index* - Index of the first element of the result set to be returned.
- *limit* - Maximal number of results to be returned.
- *filter* - A repository implementation might also offer the possibility to filter the list of collections according to specific criteria. An optional filter expression can be used to

reduce the number of delivered results. A repository may support different criteria to filter the output

- *result list* - The operation results in a list of collections/resources that is returned to the client and contains resource descriptions according to the collection, filter, index, and limit expressions.

26.6.4 List the additional services

Besides the operations described above, a repository might provide additional services, such as search, backup, etc. A repository should list and describe the additional to the clients when the ***List Services*** operation is invoked. If additional services are offered only for specific collections of the Repository, the collection identifier can be used to list the actual available services for this collection. The required parameters for this operation are:

- *collection* - Collection identifier for which the services will be listed.
- *result list* - List of service descriptions for available services returned to the client.

26.6.5 Searching the Repository

A Repository might provide searching service, to search service descriptions according to the occurrence search terms in properties of the description and media content. It is desireable that in compliance to OpenSearch the repository provides an OpenSearch description to the search API.

26.6.6 Querying the Repository

A repository might also provide a more complex querying service in a specifc query language. As an example a query service based on SPARQL [REP2]would allow to execute complex queries on the Linked Data RDF model [REP1].

26.7 Basic Design Principles

26.7.1 Rationale

There are many proprietary solutions implementing repository functionality and also many standards for various types of repositories. Within FI-Ware we try to abstract this functionality into a Generic Enabler.

26.7.2 Implementation agnostic

The API abstracts from the concrete implementation technology. Implementations using various kinds of databases should be possible. Although the main goal is to store services descriptions in a distributed environment, any implementation of a repository can be used as long as the technical interfaces comply with the GE operation protocol and can be mapped (mediated) to the FI-Ware preferred REST-based reference implementation.

26.8 Detailed Specifications

26.8.1 Open API Specifications

- Repository Open RESTful API Specification (PRELIMINARY)

26.8.2 Other Open Specifications

The data formats for the API rely on the Linked USDL specifications:

- Linked USDL Core Vocabulary
- Linked USDL Pricing Vocabulary
- Linked USDL Service Level Agreements Vocabulary
- Linked USDL Security Vocabulary

26.9 References

REP1 RDF Primer W3C Recommendation 10 February 2004 (http://www.w3.org/TR/2004/REC-rdf-primer-20040210/)

REP2 SPARQL 1.1 Overview, W3C Working Draft 01 May 2012 (http://www.w3.org/TR/2012/WD-sparql11-overview-20120501/9

27 FIWARE ArchitectureDescription Apps Marketplace

You can find the content of this chapter as well in the wiki of fi-ware.

27.1 Copyright

27.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

SAP strives to make the specifications of this Generic Enabler available under IPR rules that allow for a exploitation and sustainable usage both in Open Source as well as proprietary, closed source products to maximize adoption.

27.3 Overview

In general a marketplace is an instrument to facilitate commerce by bringing together vendors and buyers, or offers and demand, or producers and consumers. A marketplace can support a variety of mechanisms to achieve this. This specification describes the FI-WARE Marketplace Generic Enabler, which is part of the Applications/Services Ecosystem. Any offering in context of the application and service business can be supported by marketplace functionality. A marketplace implementation can offer many different kinds of services to the participants.

The core functionality of the Marketplace is to provide a uniform service interface to discover and match application and service offerings from providers and sources (e.g. published by different stores) with demand of consumers. This core functionality provides a basis for extended services depending on the domain and nature of the target markets.

27.3.1 Target usage

Internet based business networks require at least one marketplace and stores, where people can offer and deal with services like goods and finally combine them to value added services. On the marketplace one can quickly find and compare services, which enable you to attend an industry-ecosystem better than before. Services become tradable goods, which can be offered and acquired on internet based marketplaces. Beside automated internet services this also applies for services that are provided by individuals. Partner companies can combine existing services to new services whereby new business models will be incurred and the value added chain is extended.

Given the multitude of apps and services that will be available on the Future Internet, providing efficient and seamless capabilities to locate those services and their providers will become crucial to establish service and app stores. Besides well-known existing commercial application stores like Apple App Store, Google Android Market, and Nokia Ovi, there are first efforts to establish open service and app marketplaces, e.g. in the Amazon Web Service Marketplace, the U.S. Government‘s Apps.Gov repository and Computer Associates‘ Cloud Commons Marketplace. While these marketplaces already contain a considerable number of services, they are currently, at a premature stage, offering little more than a directory service.

FI-WARE will fill this gap by defining generic enablers for marketplaces and providing reference implementations for them.

27.3.1.1 ***User roles***

- Service provider will place offers on the marketplace or in a service/app store.
- Consumer can search, browse and compare offers
- Repository will be used to get services descriptions
- Registry acts as a universal directory of information used for the maintenance, administration, deployment and retrieval of services. It will be used to store information necessary for service run-time execution.
- Service store will participate on a marketplace and publishes offerings.
- Channel Maker will consumers give access to the marketplace

27.4 Basic Concepts

The marketplace is structured into five core components. These components are *Registry & Directory*, *Offering & Demand*, *Discovery & Matching*, *Recommendation*, and the *Review and Rating* component.

27.4.1 Registry and Directory

The Registry and Directory component holds information of registered stores, participants and their role(vendors, buyers, resellers, ...) and takes care of registering, updating, and deleting information about market relevant entities.

27.4.2 Offering & Demand

A service offering consists of a link to a concrete USDL description, a pricing model and the classification of the service. The Offering component is responsible for exchanging service offerings with stores and version handling/archiving of out-dated offerings.

Symmetrically to offerings also the demand side of the market need to be represented. A service demand according to expected functionality, pricing and service levels might be expressed, classified and published to the marketplace.

27.4.3 Discovery & Matching

This component is about discovering and matching offering and demand, either explicitly expressed by concrete offerings or implicitly contained in the search criteria for the inquiry process.

27.4.4 Recommendation

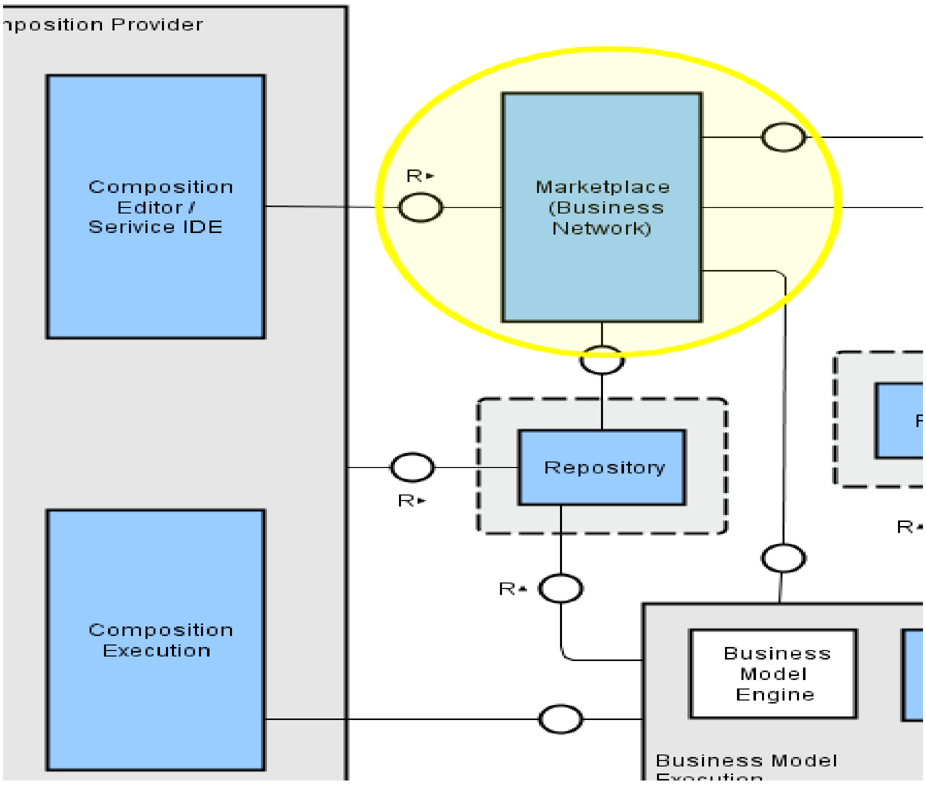
This component provides the user service recommendations based on the users' profile and context in comparison to explicit semantics of available offerings as well as previous activities and experiences of the marketplace participants (Wisdom of the crowd and social networks).

27.4.5 Review & Rating

The Review and Rating component allows users of the marketplace to give textual and star-rating feedback for services and stores along predefined categories. Reviews of users and their overall rating about applications and services can be used to improve the quality of the recommendation.

27.5 Marketplace Architecture

The Marketplace GE is used by other GE within the FI-WARE platform. Namely the Composition Editor and the Service IDE. The Marketplace itself relies on functionality provided by the Identity Management Service GE and the Repository.

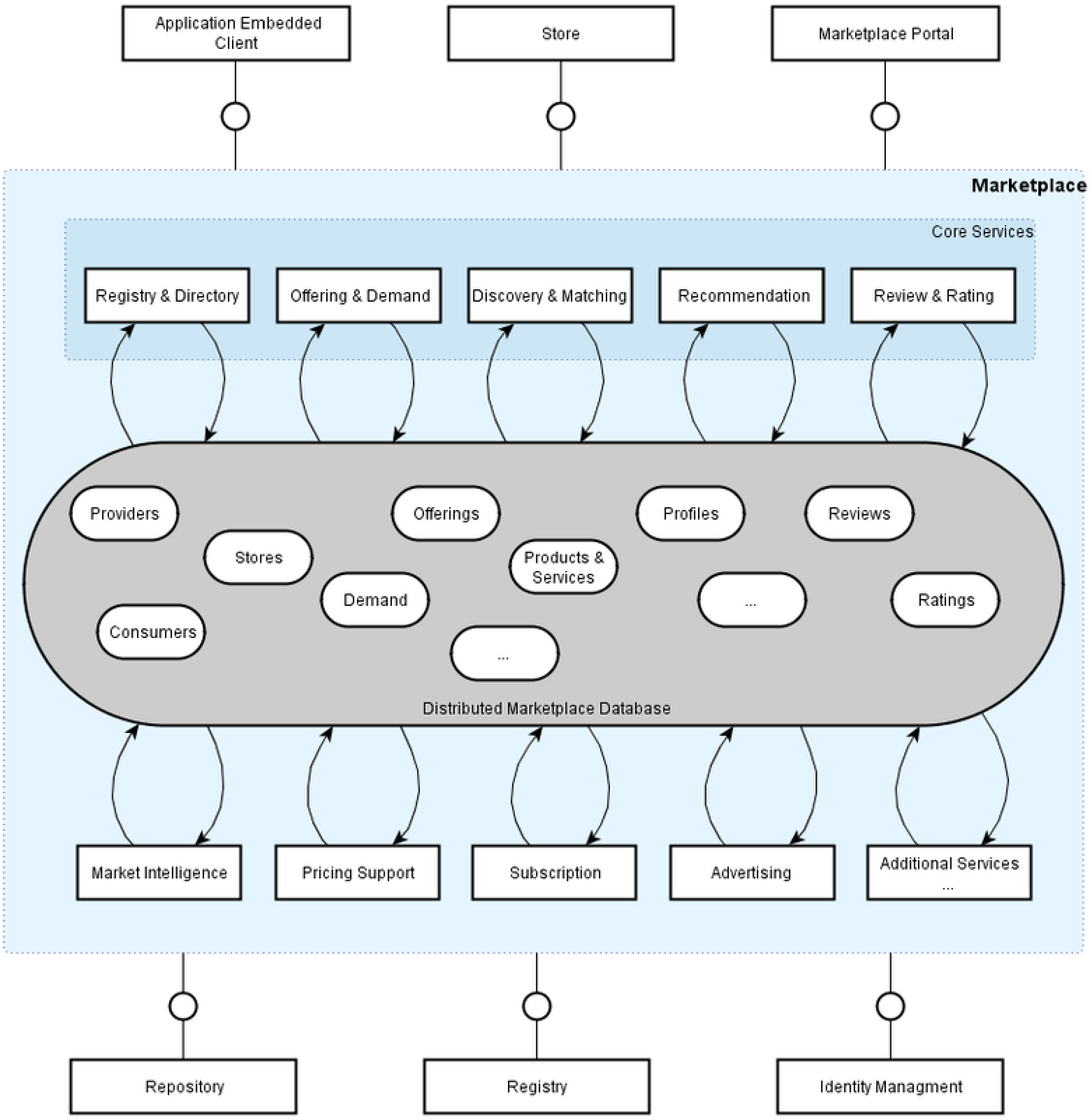


Marketplace in the context of the Business Framework

The Marketplace provides functionality necessary for bringing together offering and demand for making business. These functions include basic services for registering business entities, publishing and retrieving offerings and demands, search and discover offerings according to specific consumer requirements as well as lateral functions like review, rating and recommendation. FI-Ware will focus on the core functions but aims to provide a framework to offer also additional services. Besides the core functions, a marketplace may offer value because of its "knowledge" about the market in terms of market intelligence services, pricing support, advertising, information subscription and more.

Furthermore, the marketplace can be accessible by a HTML-based user interface for end users (Marketplace Portal), namely service consumers and service providers (stores) or a programmatic API, which allows to embed market place functionality into existing applications and environments. So for example can the marketplace API be used in order to provide marketplace access directly embedded into the composition environment (in-app shopping). A service/application store can register at the marketplace in order to get access to an API which allows creating new service offerings on the marketplace. The actual buying process always takes place at a store. So for ordering a specific service, the buyer gets returned to the stores ordering management service.

The functional components of the marketplace implementation are outlined the following diagram. This functional architecture picture is to be seen as a blueprint for implementers of the marketplace GE. It does not presume the use of a certain technology. The general idea is that all functions will operate on a huge (virtual) database of entities relevant for the business framework, such as people, organizations, products, services, offerings, ratings, etc. In practice this data might be in one uniform distributed database or a set of databases, which only virtually build the marketplace database. In fact for scaling reasons, in huge marketplaces the database is rather realized by many systems, distributed globally. Linked Data can be a good foundation for a marketplace database abstraction layer (the web is the database). In this case, which FI-WARE is focussing on, the marketplace database is more like a (semantic) index of all marketplace information bits and pieces stored elsewhere. As a query and update mechanism, we are relying on Semantic Web Repositories, Query Systems and a Query Language.



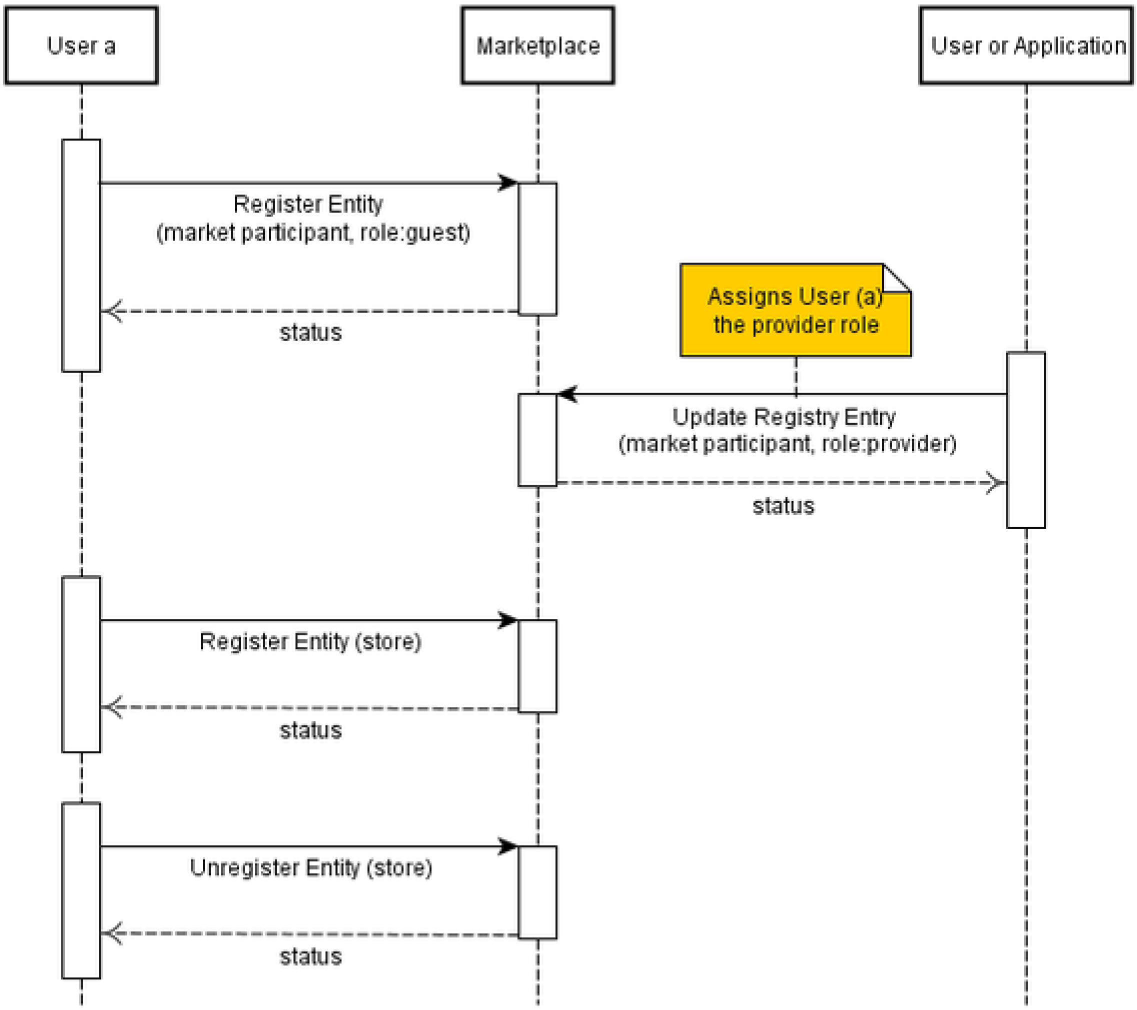
Example architecture of a marketplace implementation

27.6 Main Operations

Each functional block of the Marketplace can be considered as a GE as well. Consequently it has an own abstract operation protocol specification attached.

27.6.1 Registration and Directory

It should be possible to register a number of marketplace relevant entity types such as stores, market participants and their roles in certain business aspects (provider, consumer, reseller ...). The diagram below depicts an exemplary sequence of operations with the Registration and Directory component. A user application refers to an application which has marketplace functionally embedded.



Example sequence of registration operations

The Marketplace provides a ***Register Entity*** operation for registering market participants and related business elements. Register a new Store on the marketplace. Parameters:

- *entity* - Description of the entity to be created.

The following core entity types are supported:

- *Store* - The stores which are covered by the marketplace. There will be further operations of other marketplace components, which will need to communicate with the stores, in order to get the stores offerings and invoke operations on the stores.
- *Market Participant* - The participants (users) of the marketplace. A market participant must have at least one of the following roles:
    - **Guest** - Guest users are only allowed to the discovery capabilities of the marketplace.

- **Consumer** - The actual buyers. The organizations or persons who can buy services and applications on the marketplace.
- **Provider** - Providers of services and applications available on the marketplace. Information about providers is needed in different components on the marketplace.
- **Reseller** - Reseller of services and applications. The reseller has no own service portfolio and sells services from other providers.

Existing registration information on the marketplace can be modified or updated with the ***Update Registry Entry*** operation. The parameters are similar to the Register Entity operation:

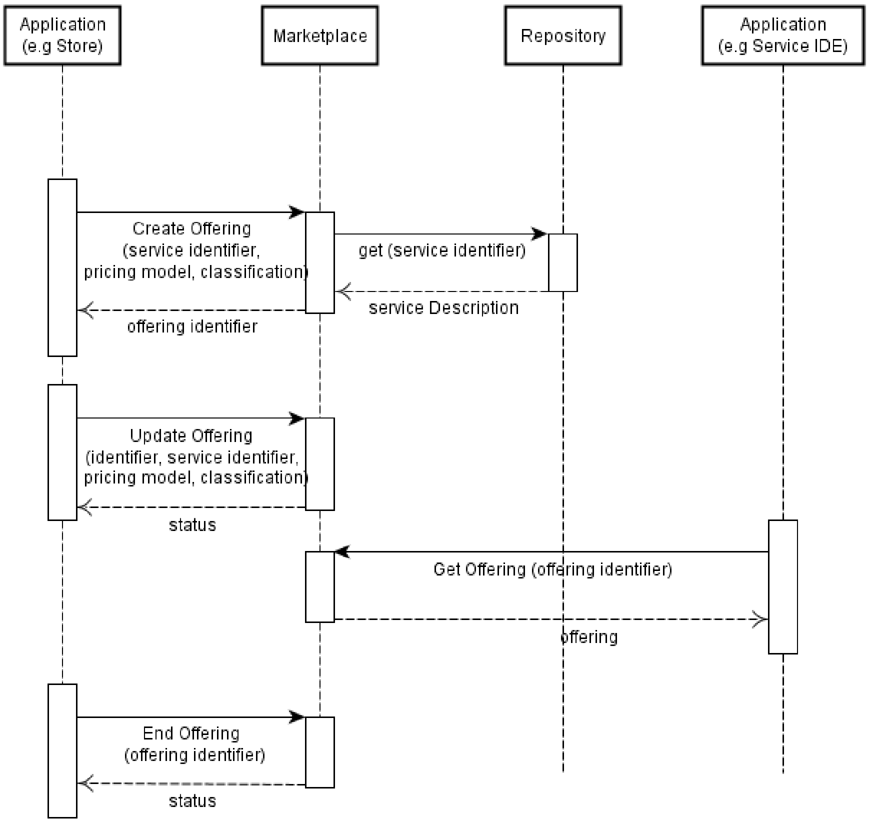
- *entity* - Description of the entity to be updated including the entity identifier.

For unregistering entities such as stores or market participants from the marketplace, the ***Unregister Entry*** operation will be used. Unregistering an entity from the Marketplace means that this entity gets flagged as unregistered and not further regarded for the marketplace operation. So, complete deletion is not possible due to consistency and history reasons. The following parameter is required:

- *entity* - Entity to be unregistered from the marketplace registry.

27.6.2 Offering & Demand

Offerings are retrieved from various sources (actually mainly stores, but other sources would be possible). So the operation is rather to ask a registered store for actual offerings. The following diagram shows an example sequence of offering management on a marketplace.



Example sequence of offering management operations

The ***Create Offering*** operation can be used to actively push an offering from a registered store into a marketplace. This operation returns the identifier of the published offering. The following parameters need to be exchanged:

- *service identifier* - Identifier of the service to be offered / link to a USDL service description.
- *pricing model* - Pricing model for the service/app.
- *classification* - Optional - Classification of the offering, e.g. name of a category the offering belongs to

The ***Update Offering*** operation can be used to update an already published offering on the marketplace. The old offering information gets versioned on the markeplace. This operation returns an OK status result on success or a parameter allowing to identify the reason for fault for further exception handling. The following parameters need to be exchanged:

- *offering identifier* - Identifier of the offering.
- *service identifier* - Identifier of the service to be offered / link to a USDL service description.
- *pricing model* - New or updated pricing model for the service/app.
- *classification* - Optional - Classification of the offering.

In order to withdraw or end a concrete offering from the marketplace the ***End Offering*** operation is used. A complete deletion is not possible due to dependency and history reasons. The following parameters are exchanged:

- *offering identifier* - Identifier of the offering.

This operation returns an OK status result on success or a parameter allowing to identify the reason for fault for further exception handling.

The ***Get Offering*** operation can be used to retrieve a offering from the marketplace. This operation delivers the actual offering data, such as pricing information, service description, associated store, and the offering classification. The following parameters need to be exchanged:

- *offering identifier* - Identifier of the offering.
- *version* - Optional parameter. If not set, the latest version of the requested offering is returned

The ***Get Offering History*** operation can be used to retrieve the history of an offering from the marketplace. This operation delivers a list of all versions of an offering. The following parameters need to be exchanged:

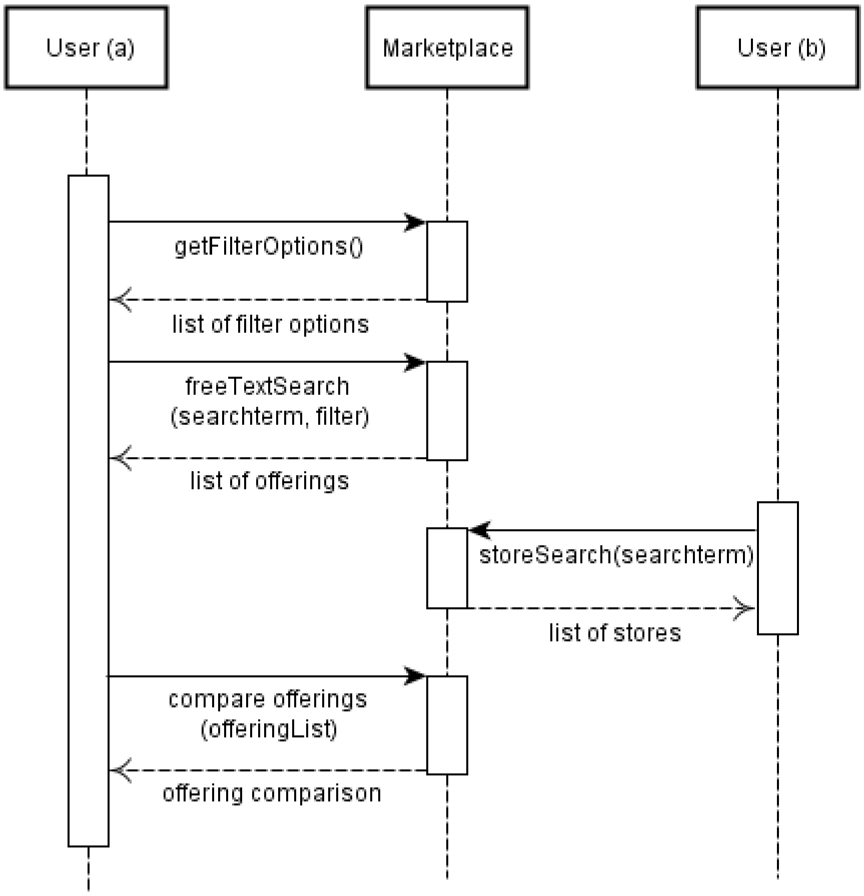
- *offering identifier* - Identifier of the offering.

The ***List Offerings for a Store*** operation can be used to retrieve all offerings from the marketplace from a specific store. For practical reasons it might be useful to restrict the list of offerings by specifying the number of results and the starting offset. The following parameters need to be exchanged:

- *store Identifier* - Identifier of the store.
- *filter* - Optional filter expression to reduce the number of delivered results.
- *index* - Index of the first offering to be returned.
- *limit* - Maximal number of results to be returned.

27.6.3 Discovery & Matching

The Discovery and Matching component supports primarily customers finding offerings and stores matching their needs. The following diagram shows an example sequence of a marketplace user (a) searching and comparing offerings as well as a second user (b) searching for stores.



Example sequence of discovery and matching operations

The ***Free Text Search*** operation can be used to search the marketplace for offerings using a search string with wildcards and filters. This operation delivers a list of all offerings which matches the specified search term and filter constraints. The following parameters can be involved in the operation:

- *search string* - Search string, potentially with wildcard operators.
- *filter* - Optional - Filter expression to reduce the number of delivered results.
- *index* - Optional - Index of the first offering to be returned.
- *limit* - Optional - Maximal number of results to be returned.
- *page size* - Optional - Number of results per page.
- *sort by* - Optional - List of sortoptions, sorted by application order.

The ***Get Filter Options*** operation can be used to get the possible filter options for a free text search. There might exist classification category specific filter options. If the classification input parameters is set, a list of these specific filter parameters gets returned.

- *classification* - Optional parameter to get filter options for offerings associated with the specified classification.

Sort options are used to define the order of an offering result list. ***The Get Sort Options*** operation returns a list of possible sort options for a list of offerings. If a classification category is specified in the request, category specific sort options get returned. The optional parameter for category specific sort options is:

- *classification category* - Optional parameter to get filter options for offerings associated with the specified classification.

To get a comparison of pricing models and USDL service descriptions between offerings the ***Compare Offerings*** operation is used. An optional filter expression can be used to reduce the number of delivered results. The parameters for this operation are:

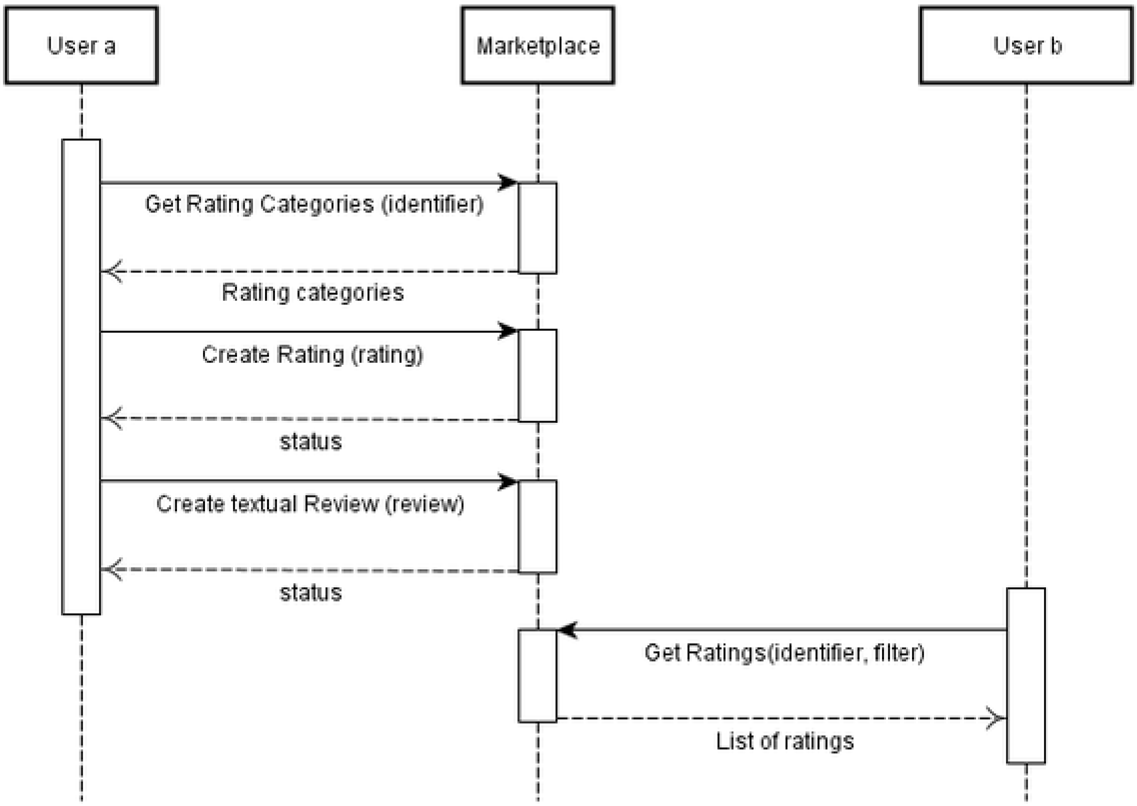
- *offering list* - List of offerings
- *filter* - Optional - Filter expression to reduce the number of delivered results.
- *index* - Optional - Index of the first offering to be returned.
- *limit* - Optional - number of results to be returned.

The ***Store Search*** operation enables a client to search a marketplace for registered stores using a search string with wildcards and filters. A list of stores matching the search string as well as the specified filter criteria is returned.

- *search string* - Search string, potentially with wildcard operators
- *filter* - Optional - Filter expression to reduce the number of delivered results.
- *index* - Optional - Index of the first store to be returned.
- *limit* - Optional - Maximal number of results to be returned.

27.6.4 Review and Rating

The Review and Rating component allows users of the marketplace to retrieve and create textual and star-rating feedback for rateable entities such as stores and services. An example sequence of marketplace users creating and retrieving ratings is pictured below.



Review and Rating Example sequence

To get the average rating for a store, a service or any other rateable entity on the marketplace the ***Get Rating*** operation is used. This operation delivers the average ratings for a rateable entity by means of different rating categories as well as an overall average rating. The following parameters need to be exchanged:

- *identifier* - Identifier of a store or a user or any other rateable entity

The ***Get Ratings*** operation delivers get the details of ratings for a rateable entity. Filter expressions are supported to reduce the number of results. If no filter expression is given then all ratings for the specified entity instance are returned.

- *identifier* - Identifier of a store or a service or any other rateable entity
- *filter* - Optional filter expression to reduce the number of delivered results.
- *index* - Index of the first rating to be returned.
- *limit* - Maximal number of results to be returned.

In order to retrieve a list of textual reviews for a rateable entity the ***Get textual Reviews*** operation is used. It also supports filter expressions as well as limits and offsets. The following parameters are available for this operation:

- *identifier* - Identifier of a store or a user.

- *filter* - Optional filter expression to reduce the number of delivered results.
- *index* - Index of the first reviews to be returned.
- *limit* - Maximal number of results to be returned.

The ***Create Rating*** operation is used to persist a new rating for a entity instance in the marketplace. This operation returns the identifier of the new rating as result. To get the available rating categories for a rateable entity use the *Get Rating Categories* operation beforehand. The following parameter needs to be exchanged:

- *rating* - Rating entry which includes a rating value for each mandatory rating category as well as a link to the rateable entity instance

To get the available rating categories for a rateable entity the ***Get Rating Categories*** is used. It returns a list of available rating categories for the specified service, store, or any other rateable entity instance. The following parameter has to be provided:

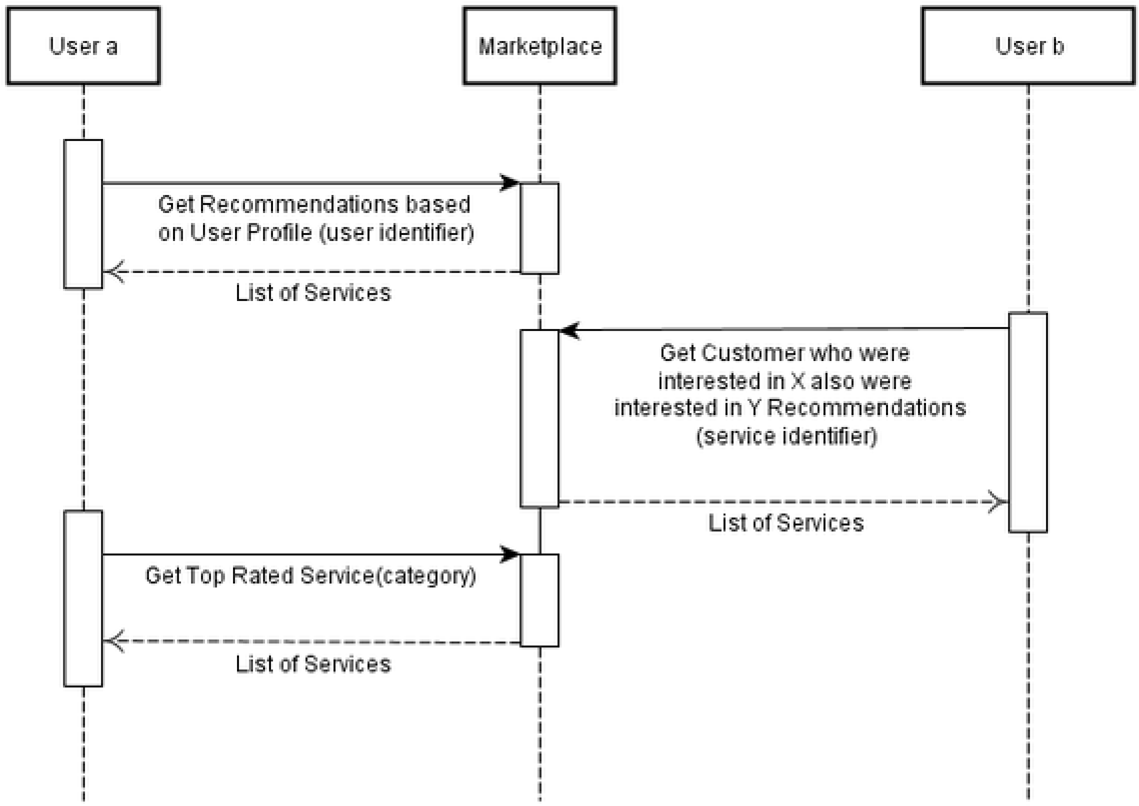
- *identifier* - Identifier of a rateable entity instance.

The ***Create textual Review*** creates a textual review for a entity instance that is flagged as rateable. It returns the identifier of the created review as result. As input parameter, the textual review is sufficient:

- *review* - Review entry which includes the textual review and an the identifier of the rateable entity instance.

27.6.5 Recommendation

The Recommendation component supports users in finding services matching their needs based on user specific data. The following diagram outlines an example sequence of a user retrieving recommendations.



Example sequence of recommendation operations

To retrieve a list of recommendations for a user the ***Get Recommendations based on User Profile*** operation is used. Recommendations might be based on the users' profile, browsing behaviour, order history and other user specific data. This operation returns a list of recommended services till the specified limit. The following parameters need to be provided:

- *user identifier* - Identifier of a user
- *limit* - Optional - Maximal number of results to be returned.

The ***Get Customer who were interested in X also were interested in Y Recommendations*** returns a list of services that were often bought together with the a given service. The supported parameters for this operation are:

- *user identifier* - Identifier of a service
- *limit* - Optional - Maximal number of results to be returned.

To get the top rated services on the marketplace as a whole or the top rated services for a certain classification category the ***Get Top Rated Services*** is used. The following parameters are supported:

- *classification category* - Optional parameter to get the top rated services for a certain classification category.
- *limit* - Optional - Maximal number of results to be returned.

27.7 Design Principles

- **API Technology Independence**

The Marketplace API is independent from implementation technology.

- **Interoperability and flexible use through HTTP content negotiation**

As described from the architecture, the Marketplace offers different interfaces in order to satisfy users' need to discover, compare, create, and update offerings. The Marketplace determines the right representation from information provided in the users' header data, so a client can receive the best representation for its abilities.

- **Mutiple store support**

The Marketplace GE is not limited to interact with one specific store. It supports multiple decoupled stores as long as they implement the marketplace interfaces for Registry and Offering & Demand.

27.8 Detailed Specifications

27.8.1 Open API Specifications

- FIWARE.OpenSpecification.Apps.MarketplaceRegistrationREST
- FIWARE.OpenSpecification.Apps.MarketplaceOfferingsREST
- FIWARE.OpenSpecification.Apps.MarketplaceSearchREST

The Marketplace GE will access the Repository its REST API:

- FIWARE.OpenSpecification.Apps.RepositoryREST

27.8.2 Other Open Specifications

The data formats for the API rely on the Linked USDL specifications:

- Linked USDL Core Vocabulary
- Linked USDL Pricing Vocabulary
- Linked USDL Service Level Agreements Vocabulary
- Linked USDL Security Vocabulary

28 FIWARE ArchitectureDescription Apps Registry

You can find the content of this chapter as well in the wiki of fi-ware.

28.1 Copyright

28.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

SAP strives to make the specifications of this Generic Enabler available under IPR rules that allow for a exploitation and sustainable usage both in Open Source as well as proprietary, closed source products to maximize adoption.

28.3 Overview

While the Repository Enabler is used to store complete service descriptions, which are more or less static and change only rarely, the Registry Enabler is used to store information on service instances necessary for run-time execution. Discovering entities and their description in an open distributed system often is achieved via registries, which have a well-known address. The registry serves as a kind of directory and for example can store detailed settings for concrete infrastructure components as well as information about human or computing agents. The information can range from stable to extremely volatile and is needed to make specific settings for and adjustments to other components in the platform. For example, the Registry can be used by the Marketplace in order to register stores, providers, persons, infrastructure components and more. The functionality and purpose of the Registry GE is comparable to the Microsoft Windows Registry [REG3] or the Light-weight Directory Access Protocol LDAP [REG1] or the UDDI Business Registry [REG4]. The main difference is that the Registry GE is fully relying on standard Web protocols and has a simpler data model.

28.3.1 Target usage

The Registry acts as a universal directory of information used for the maintenance, administration, deployment and retrieval of services. Existing (running) service endpoints as well as information to create an actual service instance and endpoint are registered. This GE will be used by potentially all GE in the Apps Chapter in order to build a common database of run-time configuration options and properties. It can also be used by GE of other chapters, such as the Cloud, Security, Data or IoT to announce their instance specific information to the rest of the platform components. There could be multiple instances of the Registry for different purposes and usage domains.

28.3.2 Rationale

The Registry has different requirements according to scalability and performance in comparison to the Repository. Highly volatile information can cause a lot of workload for a repository and needs to be dealt with quickly. On the other hand, the number of clients for a repository is usually much smaller.

28.3.3 Background

Registries are quite a common pattern in software architectures. With in the internet protocol stack defined by the IETF RFC, LDAP (Light-weight Directory Access Protocol) [REG1] is a common technical realization of the registry functionality. Other examples of registry implementations are UDDI [REG4]and the Windows Registry database [REG3].

28.4 Basic Concepts

28.4.1 Register and Deregister Entries

This component is mainly used by resource providers to register and deregister their resources as entries in the Registry.

28.4.2 Retrieving Registry Entries

The Retrieving Registry Entries component is responsible for retrieving registry entries. A client service can ask for specific settings and options of the operating environment.

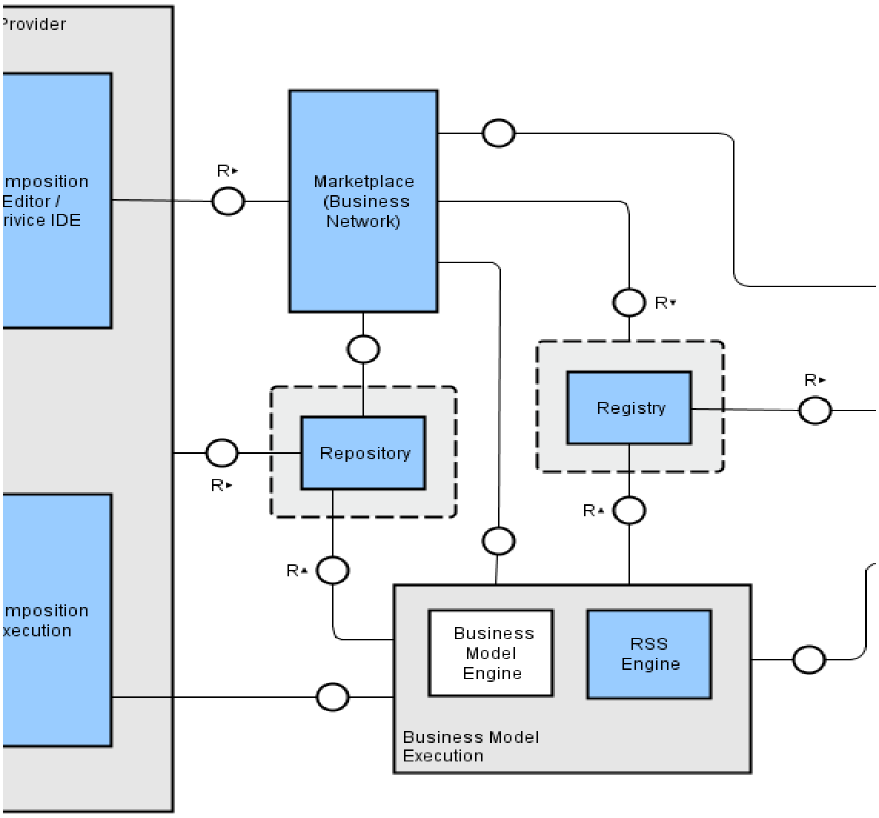
28.4.3 Data Model

The basic data elements of the registry are the ***Registry Entry*** containing the actual information and the ***Registry Key*** to access the data entries in the Registry. A registry entry can be a single atomic piece of data or a structured data such as a record of properties. The exact data model and its encoding will be defined in the interface specification. The schema (the exact names of properties and their value encoding) is the matter of the application developer or the community of developers in a respective application domain.

The Registry Key is used for accessing individual entries or a collection of entries is often organized as path into an underlying registry internal organization such as a tree.

28.5 Architecture

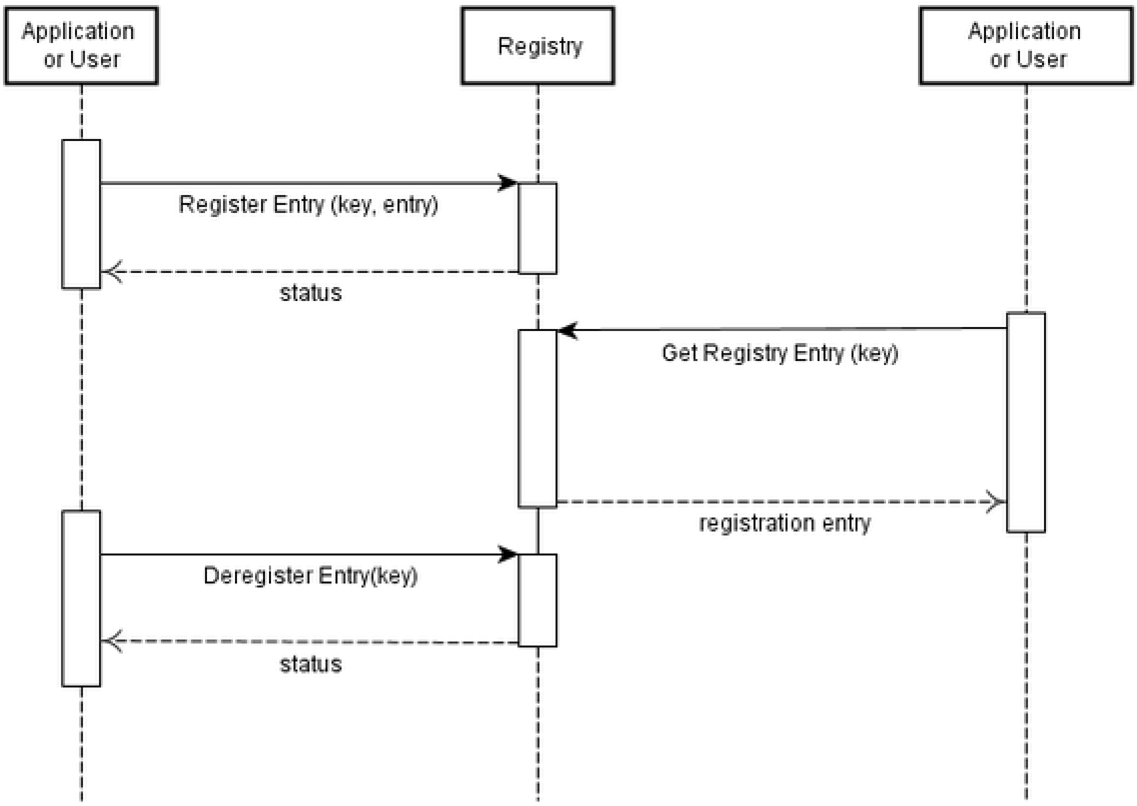
The Registry is a searchable index of the Repository GE.



Registry / Repository in the context of the Business Framework

28.6 Main Operations

The following diagram shows an example sequence how a user or other GEs can register, retrieve, and deregister a registry entry.



Example sequence of Registry operations

28.6.1 Register and Deregister Entries

The ***Register Entry*** operation is used to write or update a register information entry into the Registry. Two parameters are essentially needed:

- *key* - The Registry Key of the entry to be registered. A key can exhibit an organizational structure such as a tree.
- *entry* - The Register Entry to be registered. The entry is usually a list of name/value pairs.

For the ***Deregister Entry*** operation only the Registry Key of the Registry Entry is needed:

- *key* - Registry Key of the entry to be de-registered

28.6.2 Retrieving Registry Entries

It must be possible to directly retrieve a Registry Entry using a unique Key using the ***Get Registry Entry*** operation. In this case one parameter is sufficient:

- *key* - Registry Key of the entry to be retrieved

The ***Query Registry Entries*** operation allows the retrieval of Registry Entries matching a filter expression:

- *filter* - A filter expression to select entries to be retrieved.

28.6.3 Basic Design Principles

An implementation for the Registry might take different design decisions and technological approaches, depending on the non-functional requirements of the repository. A registry implementation might be highly distributed and scalable, if it is used by many parties on a global scale. Also the database schema might be of different complexity depending on the data requirements of the use case. Prominent example technologies which have a distributed nature are LDAP, UDDI, or distributed key/value stores for large amounts of records such as MongoDB, CouchDB, or Cassandra. The Registry specification follows the separation of concerns principle. So it is possible to supplement it with an authentication and authorization system such as OAuth [REG2].

28.7 References

REG1	LDAP protocol specifications(RFCs 4510,4512,4514,4516,4517)
REG2	OAuth2 protocol (http://tools.ietf.org/html/draft-ietf-oauth-v2-30)
REG3	Microsoft Windows Registry (http://msdn.microsoft.com/msdnmag/issues/1100/Registry/)
REG4	UDDI Business Registry (http://en.wikipedia.org/wiki/Universal_Description_Discovery_and_Integration)

29 FIWARE ArchitectureDescription Apps RSS

You can find the content of this chapter as well in the wiki of fi-ware.

29.1 Overview

The RSS GE is in charge of distributing the revenues originated by the usage of a given application among the several stakeholders involved. In particular, it focuses on sharing part of the revenue generated by an application between the marketplace provider and the service provider responsible for the application.

Revenue sharing is based on a set of business models that dictate how to distribute revenues. The RSS GE must be fed with these models and additional information regarding users. Users are assigned to different groups depending on their role. Each Group of Users has a Revenue Share model assigned per each possible revenue source. All these assignments are done externally to the RSS GE and must be fed via available APIs. In addition, service usage information (CDRs) must be periodically fed to the RSS GE to enable the revenue sharing process.

There are different events that may generate revenue sharing, the most important being:

- Application download
- Advertising
- Application usage (use of API, subscription).

Depending on the business model defined for a particular revenue source, not all concepts will be treated in the same way. For instance, for the “usage of application” source, there may be a sharing of the incomes generated by subscriptions, but not of those generated by the use of API, both concepts included in this revenue source.

It will also be possible to assign a RS model to individual users for each revenue source. In this case, the model assigned to the user will be used instead of the RS model assigned to his/her Group. The assignment of RS models to users is also externally managed.

If a user belongs to several Groups, the revenue he gets depends on the kind of activity he performs. For example, if the user has both developer and tester roles, the use of the applications he develops will give him a RS as defined by the RS model in Developer Group, while the testing of applications will give him a RS as defined by the RS model in the Tester Group.

If a Group has no RS model assigned, users belonging to that Group will not get any RS from the user role related to that Group.

29.2 Basic Concepts

29.2.1 Data Model

The main data artifacts used by the RSS GE are as follows:

29.2.1.1 *Revenue Sharing models*

RS models describe the way to distribute revenues between the different stakeholders involved in a given service. A Revenue Share model is defined by:

- *Identifier*. A unique model identifier.
- *Description*. A textual description of the model.
- *An algorithm*. It defines how the share is calculated.
    - *Textual description*
    - *Algorithm type* (fixed percentage, functions with an increasing slope, intervals with fixed share per interval, etc.). At least the fixed percentage and the exponential RSS algorithms will be supported in the first release of FI-WARE.
    - *Parameters*. A set of values for the parameters associated to the algorithm (e.g. for the fixed percentage algorithm, the specific percentage).

29.2.1.2 *CDRs*

CDRs provide accounting information related to the usage of services available in FI-WARE. Therefore they should provide information about the actual service being used, the application which uses the service, the service provider, the costs incurred, etc. Thus it should contain, at least, the following parameters:

- *CDR source*: the system that is providing CDRs (marketplace, e-store, charging application ...).
- *Service id*: service whose usage is being charged.
- *Operator ID:* Mobile Network Code (ITU MNC)
- *Correlation Id*: CDR sequence number - This element is a register index and must be unique for each source.
- *Time Stamp*: time/date
- *Country Id*: Mobile Country Code (ITU MCC)
- *Application Id*: application which is using the service being charged.
- *Event Id*: event that is being used for charging.
- *Purchase Code:* Identifier of the actual purchase operation.
- *Description:* Additional textual description.
- *Cost: amount of a given currency charged in this CDR*
    - *Currency*
    - *Units*
- *Taxes: taxes charged in this CDR in a given currency*
    - Currency
    - Units
- *User id*: User or user group subject to revenue sharing (MSISDN)

29.2.1.3 ***Payment info***

The payment file contains the following information for each user:

- *Amount*: amount to be paid.
- *Currency:*: currency in which the amount must be paid.
- *Specific information*: specific payment information corresponding to the selected payment method:
    - *Paypal*: Name, surname, email
    - *Bank account*: Account number, account name, description.
    - *Credit card*: Card number, card name, security code, description, expiration date
- *Tax information": textual description of tax information that must be included in the invoice*

29.2.1.4 ***User Management***

RSS GE needs the following information about users:

- *Identifier*
- *Name and surname*
- *Profile*. A profile is composed of one or more Groups of Users. Only one Group per user role is allowed. Therefore, if the user belongs to several Groups, each of them must have a different user role (that is, the profile can't have two developer groups).
- *Operator* (an operator is related to a country).
- *Language*
- *Payment currency*: currency in which the user will receive the payments.
- *Revenue share models* (not mandatory, in general the RS model is assigned to the Groups). The user may have a different RS model for each combination of revenue source, role, event and service.
- *Payment information*:
- *Payment method chosen to receive the revenues*. Different payment methods should be supported, like:
    - *Paypal*.
    - *Bank account*.
    - *Credit card*.
- *Payment information corresponding to the selected payment method*:
    - *Credit card*: Card number, card name, security code, description, expiration date, credit card provider (VISA, MasterCard …). A user may have several credit cards.
    - *Paypal*: email

- *Bank account*: Account number, account name, description, Swift code (US accounts), IBAN (UE accounts), bank office address (UE accounts). A user may have several accounts.

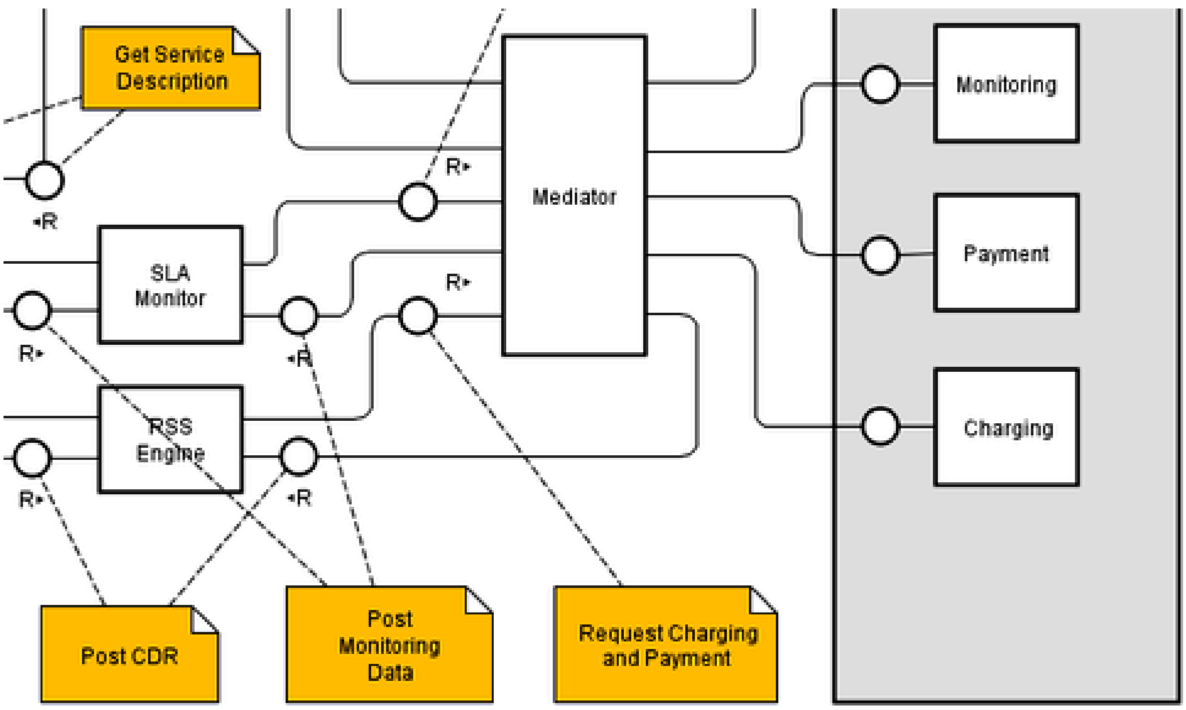
29.2.1.5 *User Groups Management*

RSS GE needs the following information about User Groups:

- *Identifier*. MSISDN based. An identifier can never be reused.
- *Name*. Group name.
- *Description*. The description will be used only for administrative purposes.
- *User role* (developer, tester, trusted partner …)
- *Category* (VIP …).
- *Revenue share models*: The Group may have a different RS model per for each combination of revenue source, event and service.

29.3 Architecture

The following diagram gives an overview of how the RSS interfaces with the rest of the GE that make up the FI-WARE marketplace.



The main interactions between RSS and other enablers are as follows:

- Charging
  - Reception of CDRs

- BE&BM: Business models would be read by the RSS GE from the Service Repository. At least the following information should be available:
    - RSS Models provisioning
    - Assignment of RS model to users and User Groups
- Information about users. This information is handled by the IDMaaS GE:
    - User and User Groups Management
- Payment Broker (external component)
    - Sending payment information

29.4 Main Interactions

29.4.1 Receiving CDRs

In order to calculate revenue shares, RSS must be fed with CDRs from the charging application. Several CDRs should be aggregated in a file and sent to RSS using the following operation:

- *ReceiveCDRs*
- *Parameters:*
    - CDRList: list of CDRs

29.4.2 User and User Groups Management

RSS GE will get information about users and user groups from the IDMaaS GE. This information should include a user identification, role and other relevant parameters.

29.4.3 RSS Models management

The RSS GE will get information about RSS models from the service repository GE.

29.4.4 Assignment of RS model to users and User Groups

The RSS GE will get information about the assignment of Revenue Share models to users and user groups from the service repository GE.

29.4.5 Payment Management

In order to distribute the revenue shares to all stakeholders, the RSS GE generates a file with the information needed by an external payment broker. The RSS GE sends the file to the external broker which then conducts the actual payment process.

29.5 Basic Design Principles

The RSS GE exposes all its functionalities as Web Services to facilitate integration with other GE and external components. Besides, it provides a web based GUI for operation and management. The type of algorithms the RSS can use for calculating revenue shares can be easily extended by adding new algorithms.

30 FIWARE ArchitectureDescription Apps Mediator

You can find the content of this chapter as well in the wiki of fi-ware.

30.1 Copyright

30.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

30.3 Overview

Providing interoperability solutions is the main functionality of the Mediator. The heterogeneity that exists among the ways to represent data (i.e. to represent syntactically and semantically the information items that are requested or provided by an application or a service), and to represent the communication pattern and the protocol or the public process needed to request a functionality (executing a composition in a different execution environment or implementing dynamic run-time changes might require a process mediation function), are problems that arise in FIWARE. Acknowledging the necessity to deal with these heterogeneities, mediation solutions are required in FIWARE.

The mediator is basically a middleware application responsible for providing interoperability among different communication protocols and among different data models. For example it can convert ASCII delimited message payloads from older protocols such as FTP into an XML message payload submitted to a web service (both soap over http or rest over http). Thus the main capabilities of the mediator are protocol and data transformations. An other example of data transformation is the transformation of an XML payload into an other XML payload through XSLT or XQuery.

The Mediator provides data mediation and protocol mediation capabilities to enable clients that play the role of Mediation Service Creators to compose different kinds of target service, and enable clients that play the role of Mediation Service Clients to invoke the mediated services (see picture).
The Composition Engine GEs plays both roles of Mediation Service Creator and Mediation Service Client.

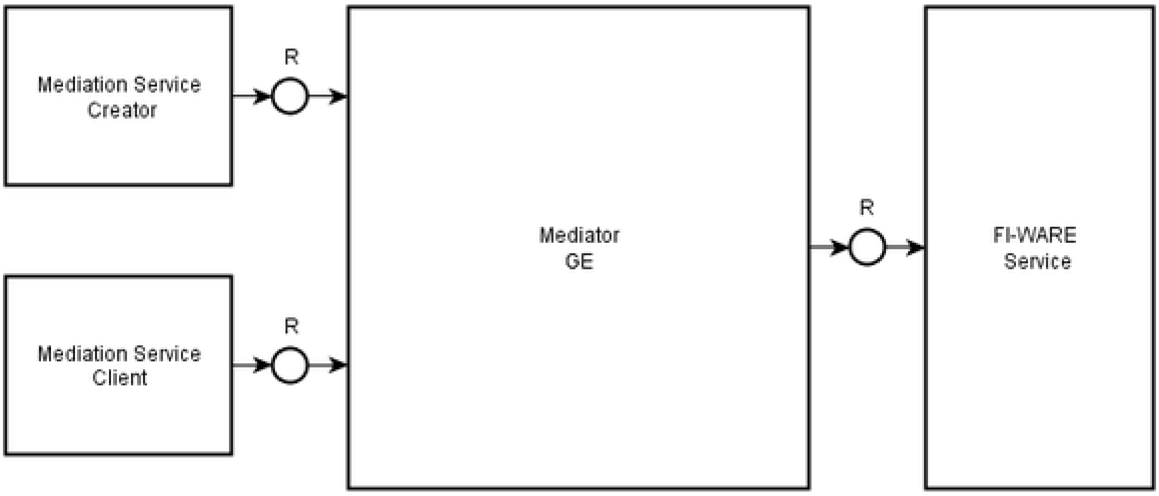


Figure 1.1 Mediator Role in FI-WARE

The mediator provides an Administrator GUI and APIs in order to allow mediation services to be constructed given a target service to be used in a service composition.

30.4 Basic Concepts

FI-WARE platform should be able to support services exposed through different protocols and technologies and enable the creation of new composed services. The mediator shall provide the "glue" between the service layer and the composition layer in order to enhance the composition capabilities of the composing GEs.

Within FI-WARE we abstract this functionality into a Generic Enabler called Mediator. The API abstracts the concrete implementation technology. Implementations using various kinds of platforms and frameworks should be possible. The main goal of the mediator is to provide a virtual proxy of the target service to be used by the Compositon Engine GE instead of the target service. The Virtual Proxy is configured with *Mediation Tasks* and *Dynamic Mediation Tasks* that provide data mediation and protocol mediation capabilities in order to make the target service suitable for composition. The first release of this GE will provide an Administration GUI for the configuration of such Virtual Proxies. The final release will provide remote generic APIs to allow the configuration of the Mediator directly by the other GEs

30.4.1 Data Model

The mediator offers a set of available *mediation tasks* and *dynamic mediation tasks*: the set of mediation capabilities that can be used via the mediator. The mediator allows users to create and manage their *mediation services*: a *mediation service* is a virtual proxy towards a web service that executes a chain of *mediation tasks* and/or *dynamic mediation tasks* between the caller and the target service. The *mediation tasks* and *dynamic mediation tasks* must be choosen from the set of available task types and the concrete implementation of the mediation tasks to be chained are potentially provided by different mediator implementations.

Each mediator implementation (asset) will provide its own set of addressable *mediation tasks* and/or *dynamic mediation tasks*. How to build a concrete *mediation task* or *dynamic mediation task* depends on the specific mediator implementation.

30.4.1.1 ***Mediation Task***

Mediation tasks are the mediation capabilities that can be used via the mediator. The mediator maintains a set of the available mediation tasks.

The concrete implementation of a mediation task is provided by a specific mediator implementation (asset).

Examples of mediation tasks provided by the TI Mediation asset include:

- SOAP2REST: allows a REST Service to be called from the SOAP protocol
- SOAP2POX: allows a service that is expecting a POX Payload (Plain Old XML) to be called from the SOAP protocol
- TCP2HTTP: allows a service exposed via HTTP to be called using TCP transport

The mediation tasks exposed by Mediation GE that can be implemented using the TI Mediation asset, are a chain of the built-in low level mediation capabilities provided by WSO2 ESB and Apache Camel

A short list of these **mediation capabilites**:

Name	Description
Send	Send a message out
Log	Logs a message
Property	Set or remove properties associated with the message
Sequence	Refer a sequence
Event	Send event notifications to an event source
Drop	Drops a message
Enrich	Enriches a message
Enqueue	Create an enqueue mediator
Filter	Filter a message using Xpath (if else logic)
Out	Inbuilt filter for choosing messages in ESB out path
In	Inbuilt filter for choosing messages in ESB path
Switch	Filter a message using Xpath (switch logic)

Router	Route messages based on XPath filtering
Conditional Router	Route messages based on 'Condition'
Validate	Schema validation for messages
XSLT	XSLT Transformations
XQuery	XQuery Transformations
Header	Set or remove SOAP Headers
Fault	Create or remove SOAP Faults
Rewrite	Create a rewrite mediator

We provide some example of the configuration of these mediation tasks using the TI Mediation asset.

Example 1: WS-Security
Virtual proxy configuration that adds WS-Security to the unsecured target service "ServiceExample"

```
<proxy xmlns="http://ws.apache.org/ns/synapse"
name="SecuredServiceExampleProxy" transports="https"
startOnLoad="true">
 <target>
    <inSequence>
       <header xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
name="wsse:Security" action="remove" />
    </inSequence>
    <endpoint>
       <address uri="http://<ServiceExample URL>" />
    </endpoint>
    <outSequence>
       <send />
    </outSequence>
 </target>
 <enableSec />
 <policy key="conf:/repository/axis2/service-
groups/SecuredServiceExampleProxy/services/SecuredServiceExampleProx
y/policies/UTOverTransport" />
</proxy>
```

Example 2: Protocol Transformation TCP2HTTP

```
<proxy                        xmlns="http://ws.apache.org/ns/synapse"
name="TCPServiceExample" transports="tcp" startOnLoad="true">
  <target>
     <endpoint>
        <address uri="http://<ServiceExample URL>" />
     </endpoint>
     <outSequence>
        <send />
     </outSequence>
  </target>
</proxy>
```

Additional Mediation tasks will be added to the Mediator GE on the basis of FI-WARE needs.

30.4.1.2 *Dynamic Mediation Task*

In the current vision of the mediator GE, this enabler allows users to create and manage mediation services “offline”, at design time. A mediation service is a chain of *mediation tasks* and *dynamic mediation tasks* between a service producer and consumer that can be accessed through a Web service interface. This chain deals with all the mediation problems that may arise between these two protagonists.

To cope with some limitations in this current pragmatic vision of how mediation is made – limitations related to potential missing information at design time – the mediator GE offers specific *mediation tasks* called *“Dynamic mediation tasks”*. These tasks may be needed because, at design time (when the chain of tasks is defined), all the needed data and/or information is not necessarily available to be able to solve mediation issues between the caller request (from “service consumer”) and the target service (“service provider”).

We postulate that this data and information will became available at runtime and that the *dynamic mediation tasks* will then dynamically solve the remaining mediation issues. This approach is the first step toward a fully dynamic mediation.

The concrete implementation of a *dynamic mediation task* is provided through features provided by an existing THALES asset called *SETHA2* that deals with data, protocol and process mediation in a SOA context.

We identify multiple *dynamic mediation tasks* types that are detailed hereafter:

- Data *dynamic mediation tasks*

The data dynamic mediation tasks can be used to solve the following issues:

- Consumer knows the target service and the operation to invoke but not its parameters (e.g. order of parameters, exact type of parameter to use, …)

- Consumer knows the target service but not exactly the operation to invoke (e.g. the precise name of the target operation is not known)

- Protocol *dynamic mediation tasks*

If the target service protocol is not known at design time, the protocol dynamic mediation task will be used to bridge the protocol gap between service consumer and producer (e.g. SOAP/HTTP service consumer and DDS service).

- Process *dynamic mediation tasks*

To be used if there is a potential process mismatch at runtime between the consumer and the producer; in the case where one of the processes (either from consumer or producer) is not known at design time.

A second step toward fully dynamic mediation in FI-WARE would be to rely directly on the THALES asset to deal with extreme cases where only consumer's side requirements are known at design time and all mediation must be automatically defined and invoked at runtime.

30.4.1.3 ***Mediation Service***

The mediation service represent the final mediation capability exposed by Mediator GE to the external word. A mediation service can be composed by a single mediation task (the simplest case) or by a chain of mediation tasks that can be provided by different Mediator implementations The mediation service is configured with a chain of *mediation tasks* and/or *dynamic mediation tasks*: they are the mediation tasks that the mediation service will execute between the caller and the service.

The *Mediation service URL* is the URL that allows the invocation of the target service with the mediation logic included in the chain of mediation tasks/dynamic mediation tasks configured.

To configure the mediation service the *Target service endpoint* must be specified: it is the URL of the target web service that will be invoked via the mediation service.

30.5 Mediator Architecture

The Mediator GE is used mainly by the Compositon Engine GEs within the FI-WARE platform. It provides a layer of virtual proxies to be used by the composer instead of the target services in order to allow the composer support various kinds of target services. Besides the FI-WARE platform, Future Internet applications or composed services on top of the FI-WARE platform can use the mediator as a service for their own purpose.

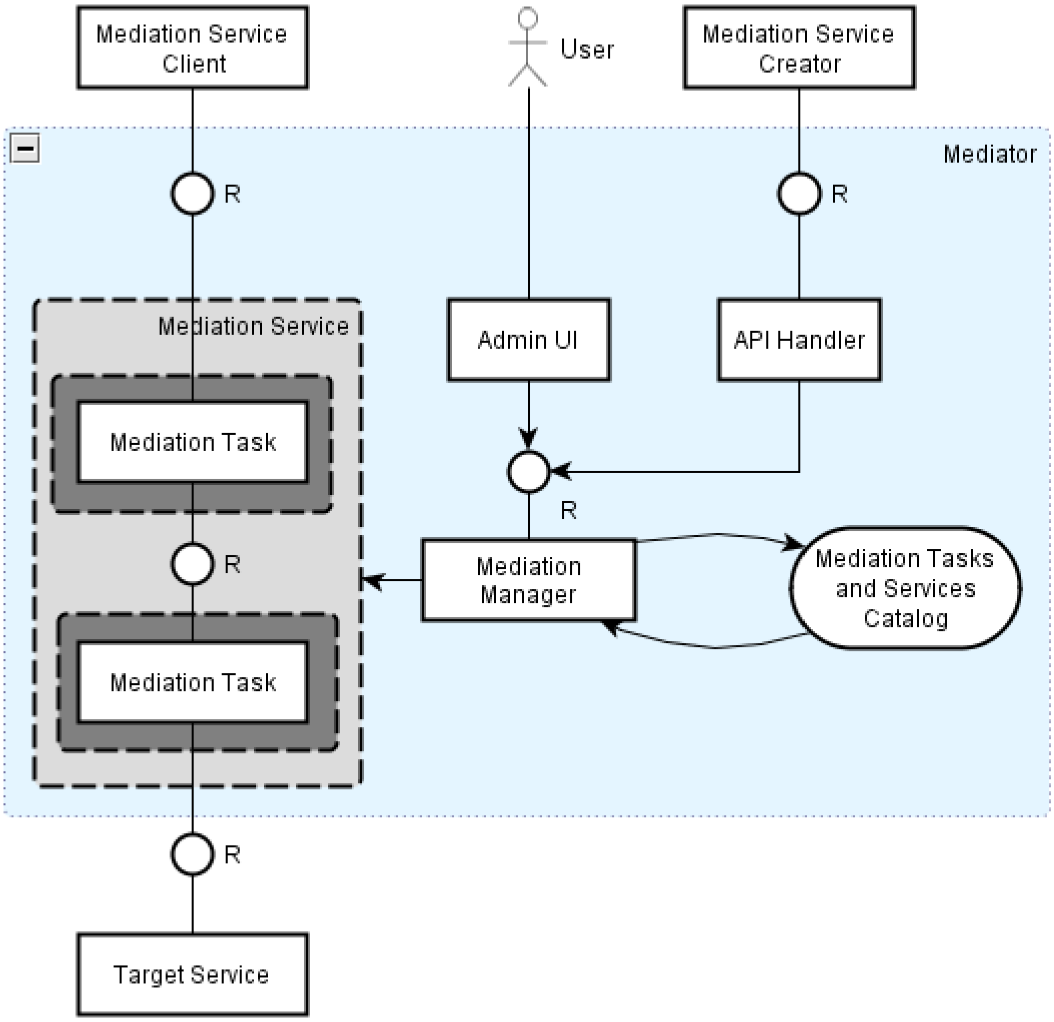


Figure 3.1: Mediator Architecture

30.6 Main Interactions

There are two main interactions provided by the mediator:

- at design time there are interactions in order to create and handle virtual proxies
- at execution time the mediator provides the virtual proxy, whoose URL has to be invoked in order to mediate the target service

The design time interaction occurs between a client that plays the role of the Mediation Service Creator and the Mediator.

The execution time interaction occurs beetween the Mediation Service Clients and the Mediation Services exposed by the Mediator. The mediated services are invoked by the Mediation Service Clients just like any other service.

As regards the current release, all design time interaction needed to manage mediation tasks and services is performed through the Web GUI of the various Mediator Implementations (assets). Refer to the User guide of the specific asset.

The main interaction at execution time will be:

- invocation of the mediation service, exposed by the mediator

The main interactions at design time will be (preliminary description of the remote APIs that will be provided by the final release of the mediator):

- create a mediation service
- delete a mediation service
- get a specific mediation service configuration
- get available mediation tasks
- get available dynamic mediation tasks

30.6.1 Invocation of the Mediation Service

At execution time the mediation capabilities are provided through services that can be invoked by the client GE using the mediation service URL. The mediation services can be exposed using various technologies, for example through soap web services and rest web services.

30.6.2 Mediation Service Management

The design time API will be designed for future releases of the FIWARE platform taking into account the available implementations (assets) of the Mediator GE

30.7 Basic Design Principles

- **API Technology independence**

  The API abstracts the concrete implementation technology. Implementations using various kinds of platforms and frameworks are possible.

- **Modularity**

  Mediation Tasks can be composed, creating Mediation Task chains that realize complex mediation logic.

30.8 Concrete Implementation Documentation

30.8.1 TI Mediation Asset

In order to learn how to create mediation tasks with the TI implementation of the FIWARE GE enabler refers to the documentation of the Architect week in Zurich TI-Mediator-doc and to understand in deep detail the concept of Virtual proxy and how they are implemented in the TI Mediation asset see apache-synapse project mediation catalog (http://synapse.apache.org/userguide/mediators.htm).

In order to have an understanding of the Administration GUI of the TE-Mediator see the User guide of Wso2 ESB Wso2-ESB-UserGuide

30.8.2 THALES Mediation Asset

Thales brings the *setha2* asset into FI-WARE in order to offer a set of dynamic mediation tasks. A brief description of *setha2* is available at SETHA2_DESCRIPTION.

Examples of mediation tasks provided by the THALES Mediation asset ("SETHA2"):

- Data *dynamic mediation tasks*
- Protocol *dynamic mediation tasks*
- Process *dynamic mediation tasks*

31 FIWARE ArchitectureDescription Apps ServiceComposition

You can find the content of this chapter as well in the wiki of fi-ware.

Disclaimer: The sustainability of this Architecture Description cannot be guaranteed due to internal changes in the project consortium.

31.1 Copyright

31.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

31.3 Overview

The Service Composition is a core enabler of the FI-WARE Platform. It allows users to create, manage and execute composed services. It consists of to main parts, the editor and the execution environment. The editor provides users with a graphical environment that allows them to create composed services in a more convenient way, providing graphical constructs for flow control and component service templates (and hiding away some of the service communication and data representation details). These composed service representations (i.e. skeletons) specify the main parts of the business logic of the composed services. During the run-time the composition engine dynamically decides about what services to invoke or which data source to use based on constraints evaluated at that particular time. Essentially the composition engine is creating the workflow step-by-step during runtime, and different composition decisions can be taken depending on external constraints or on the return values of previously executed services.

As an intermediary step, the composed service specification can be stored/fetched from the Repository GE. Moreover the Service Composition GE can search (based on different criteria) the Marketplace GE for services to either be used in a new composition or to be executed by the execution environment.

31.3.1 Target usage

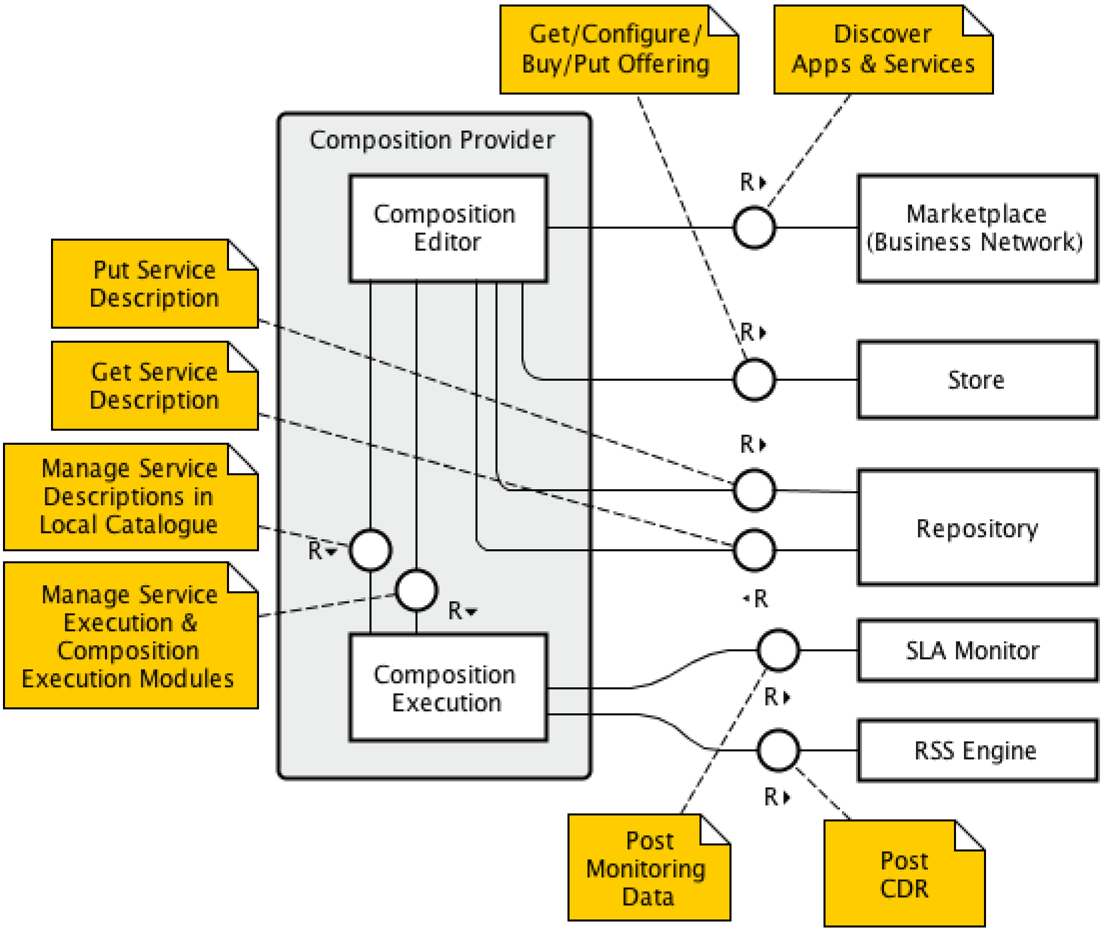
The Service Composition GE helps the service provider to create composed services. Editors should provide an environment to combine and configure applications and services in graphical way. The editor could cater for different user expertise (from technical experts to domain experts without technical expertise or even simple end-users with no programming or technical skills) and roles (from composed service creators, to resellers and finally to prosumers) by hiding complexity behind different types of construction blocs, trading off flexibility for simplicity. The Service Composition GE should allow testing, debugging, installing, executing, controlling and post execution analysis of the composed applications.

Composition descriptions and technical service descriptions should be stored/fetched to/from the Repository GE. The Service Composition could be connected to a user and identity management service for controlling access to the applications.

When creating compositions/mashups editors might connect o the business infrastructure:

- Marketplace to search for services
- Shops to purchase component services them for testing /deployment, and to expose composed services for purchase
- USDL Registry to browse business information related to services

31.4 Composition Provider Architecture



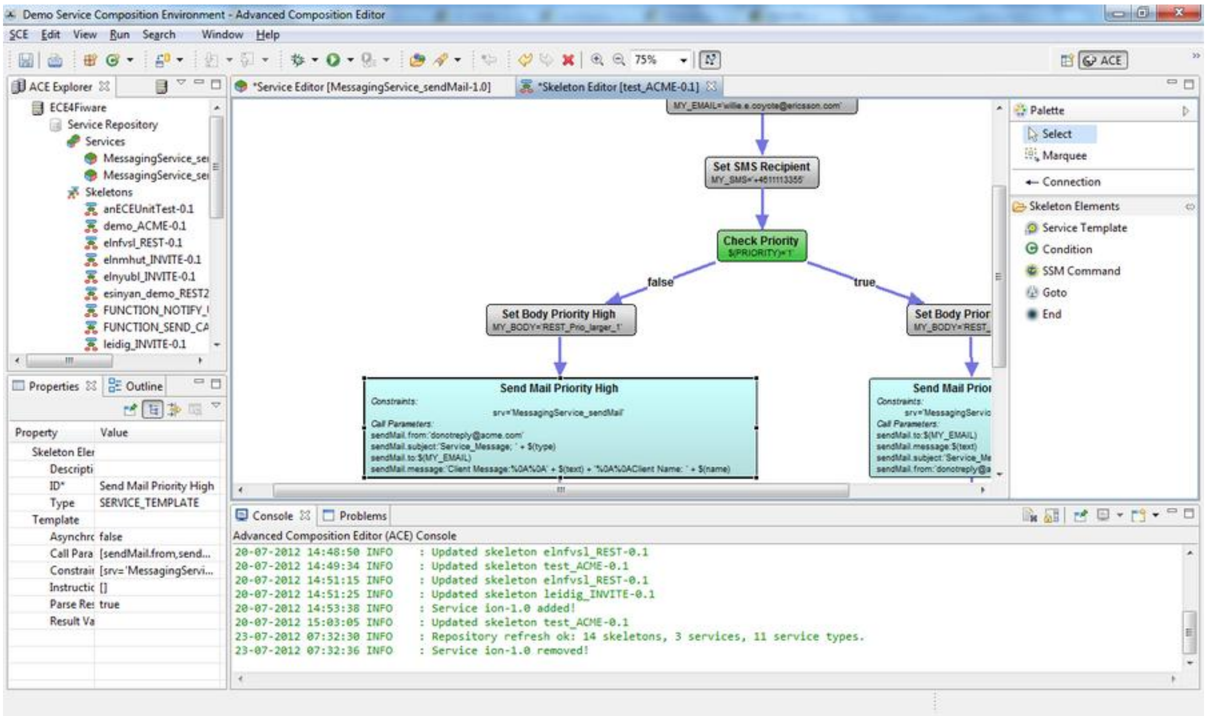
Composition Provider Architecture

31.5 Basic Concepts

The Service Composition GE offers two main functions: composition creation and composition execution.

31.5.1 Composition Creation

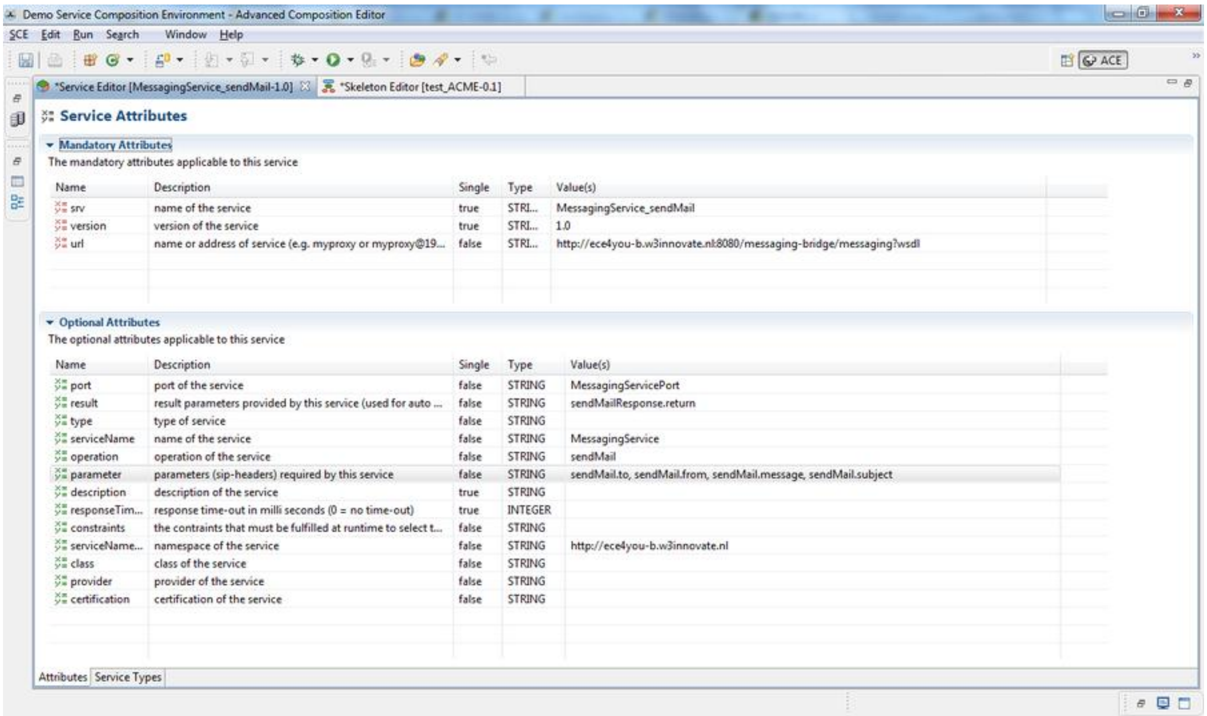
The editor allows the creation of composed services (also denoted as skeletons). The following figure shows the main graphical user interface exposed by the editor. There the user can define service descriptions (to describe how the component services can be invoked) and create compositions by adding skeleton elements and interconnecting them.



Composition Editor GUI

In order to create compositions, we need to specify in the editor how to access the services that represent the components that are invoked when executing the composition. The execution engine needs to know what are the API and the protocol used by these services, and how to set the parameters at invocation time. Thus we need to create a service description for all the services used in a composition.

When creating a service description first we need to specify what service type(s) this service is (e.g. SOAP, REST, SIP, etc). Then we need to specify values for the attributes that will be used when invoking the service (e.g. in case of a SOAP service the namespace, the port, the operation, the parameters passed to the operation, etc). The attributes to be specified could be either fixed or variable.



Composition Editor - Service Description Example

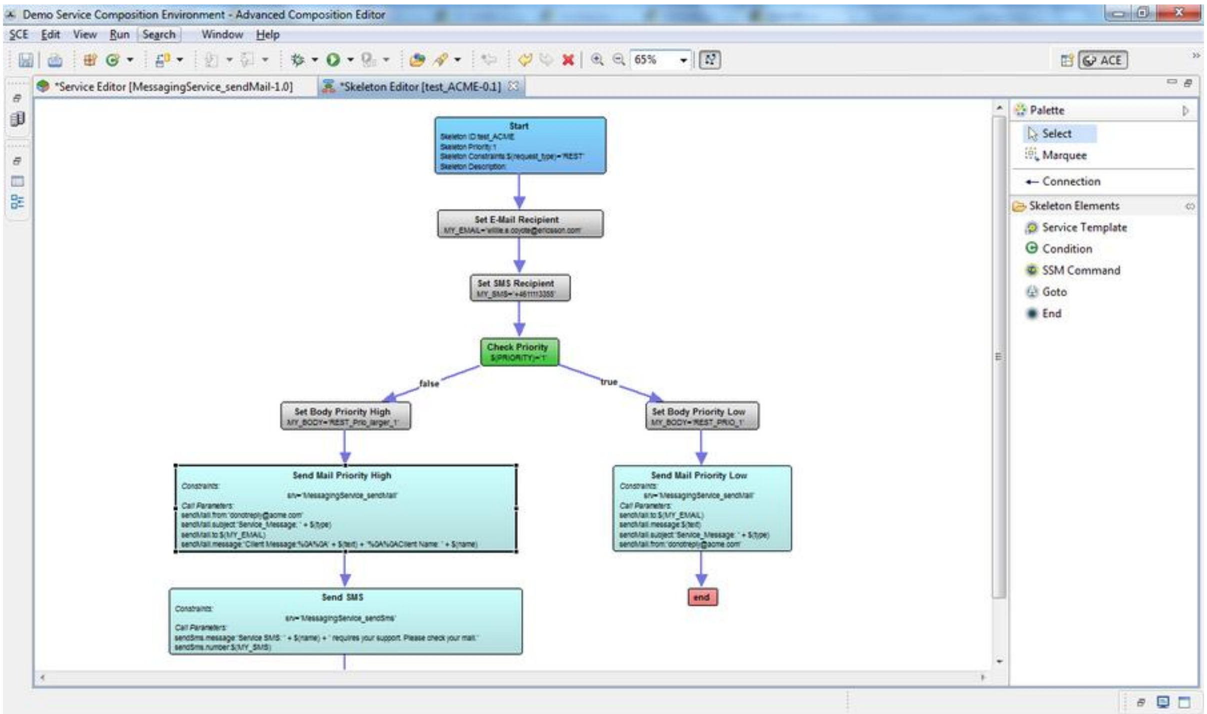
The skeletons provide the business logic, the data and control flow, and service templates (placeholders that are bound to specific service implementation during each invocation). Several services might be suitable to implement a service template. These services can have different input and output parameters and the editor needs to offer the possibility to correctly map the different dataflow types, while providing an easy way to use the unified interface. These elements are placed via drag&drop in the composition editing area and connected by linking their input and output ports. The elements can then be configured using static data or by using the relevant variables. Each skeleton consists of several interconnected elements of the following types:

- Service Template. The service template element is used during execution of the skeleton in a service composition to decide what service shall be invoked. The service description available in the local descriptions storage of the Composition Editor that fulfills the specified constraints at runtime will be selected. Only one service can be specified per service template element. Call parameters, service selection constraints, invocation semantics (synchronous or asynchronous) and the result variable (the name of the variable in which to save the response of the service) can be specified. Service selection constraints will be evaluated when the execution step arrives at this service template, thus providing late-binding for services.
- Condition. The condition element provides the possibility to branch within the skeleton upon certain conditions evaluated during runtime of the service composition execution. Different outgoing branches are supported where one outgoing branch can be connected to an unspecified default condition value which is chosen in case none of the other branches condition matches.
- SSM Command. The SSM Command element provides the option to set or remove variables in the memory space used during service composition execution. The

expression to be assigned in the setVariable clause can be of a static value or a condition to be evaluated at runtime.

- Goto. The goto element provides the option to perform a jump to another skeleton element during skeleton execution. The specified jump target can either be a skeleton element from the same skeleton and thereby offers the possibility to implement a loop construct or can be any skeleton element from a different skeleton present in the advanced composition repository.
- End. Each branch of a skeleton must close with an end element.

A skeleton is created iteratively using the GUI by using a selection tool for selecting skeleton elements and interconnecting them using a connection tool. When making a connection between a Condition Element and another skeleton element, branching can be realized.

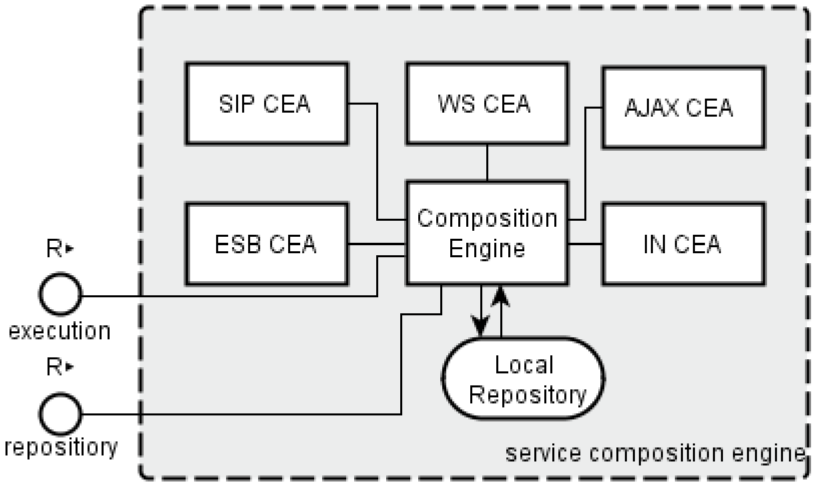


Composition Editor - Service Composition (skeleton) Example

Specification of global and local constraints are used to decide runtime service selection and event filtering. A global constraint can be specified in the skeleton start element, and is valid in the context of the composite service, for example all services used in that particular skeleton must be SIP services (syntax example: $(srv.type)='SIP'). A local constraint is to be specified within a service template. A constraint matches a service attribute against any SSM variable or literal value. Local constraints are evaluated every time the control flow of the skeleton arrives at the evaluation of a service template and invocation of a component service (syntax example: srv='ServiceName'+$(variable_name)). Note that global constraints are defining restrictions applicable to all components of the composite service, while local constraints are applied only for choosing the particular service specified by that particular template element.

Many communication-type services depend heavily on events, and first class support for events needs to be provided. External and internal events may start actions, events can be filtered, events can be triggered. Basically it is assumed that at execution all the services within a session have access to a shared state via a shared state manager (SSM). Any change in the shared state produces a high-level event related to it, e.g. change of the variable's value can generate a state change event depending on variable name, old value, new value. These events are the only way of communication between components using different technologies. The SSM employs a subscribe/notify model.

31.5.2 Execution



Execution Environment Architecture

Composite applications descriptions - the skeletons - are retrieved end executed by the engine. Protocol-level details related to the interaction with modules are left to the Composition Execution Agents (CEAs), which are responsible for enforcing composition decisions in the corresponding platform in a technology and protocol specific way. A shared state is used as means of mediating information between the application skeleton and the CEAs, thus coordinating the service execution. A variety of CEAs has been developed. The process is triggered by a composition execution agent (CEA) that receives a triggering event and requests the next step from the composition engine. Based on what the triggering events was, the composition engine selects the matching skeleton and creates a new session. Then, at each step it selects a suitable service that matches all the global and local constraints and serves it to the agent to execute. Execution results from the previous steps together with potential external events can influence the constraint-based decision process selecting the service for the new step. If several services are suitable to implement a certain step one of them is chosen. If a component service fails during execution, the next compatible one might be executed instead. An essential feature is the use of formal technical service descriptions for all constituent services. This service description is important for runtime service discovery, selection, and invocation. It is comprised of information about the service API and additional information used in service binding.

The local descriptions storage keeps skeletons and service descriptions used by the engine (previously careated in the editor or obtained from the Repository GE). Further the user can enable/disable the service and skeletons and access logging and tracing information.

The execution environment of the Service Composition exposes a basic life-cycle functionality for service and skeleton descriptions including import/export and enabling/disabling them to be triggered for execution.

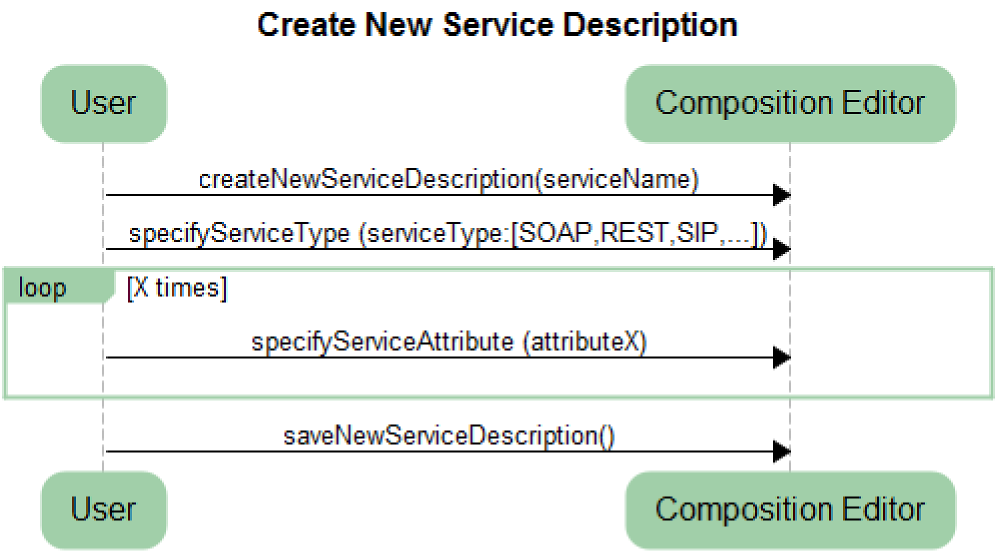
31.6 Main Interactions

31.6.1.1 *Create Composite Services*

This functionality allows the end user to create a new composition skeleton using graphical representation for data and control flow and for service placeholders. This is the main function and will be detailed further. Note that this functionality is not available as an API to be used from other architecture components, but functionality exposed to the end user via a GUI.

- **Create/Edit Service Description**

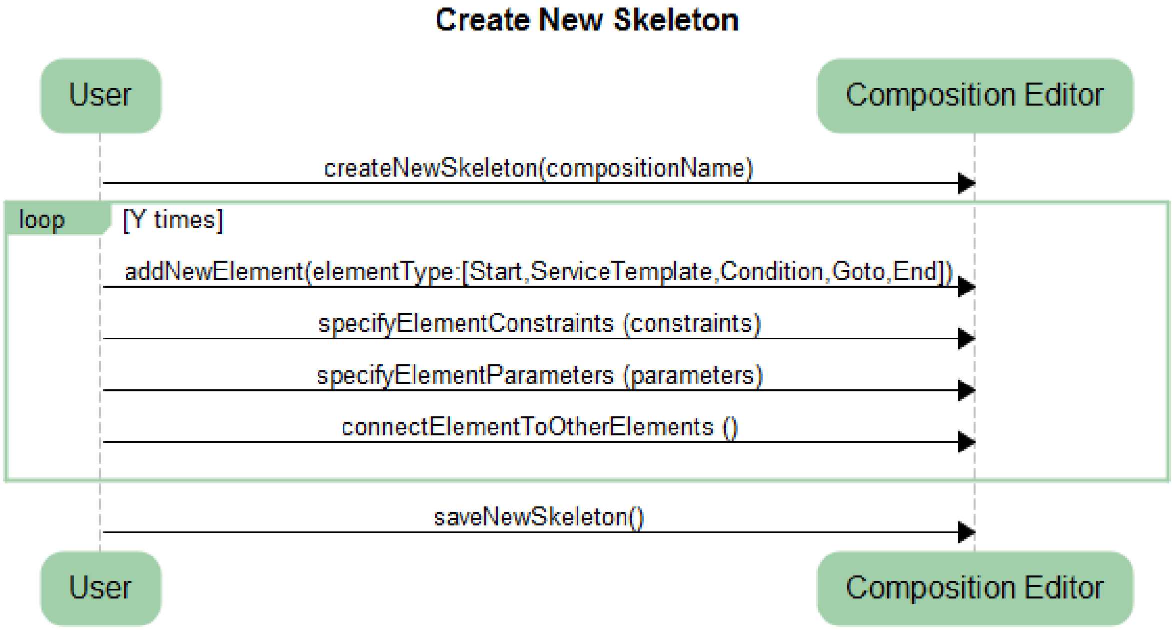
  This function offers users the possibility to create and edit component services. When creating a service description the user specifies what service type(s) this service is (e.g. SOAP, REST, SIP, etc). Then the user specifies values for the attributes that will be used when invoking the service (e.g. in case of a SOAP service the namespace, the port, the operation, the parameters passed to the operation, etc). The attributes to be specified could be either fixed or variable. Once a service is created it is stored in the local descriptions storage. Subsequently the user can browse and edit the description of component services that can be used in the composition.



Create New Service Description

- **Create/Edit Skeletons**

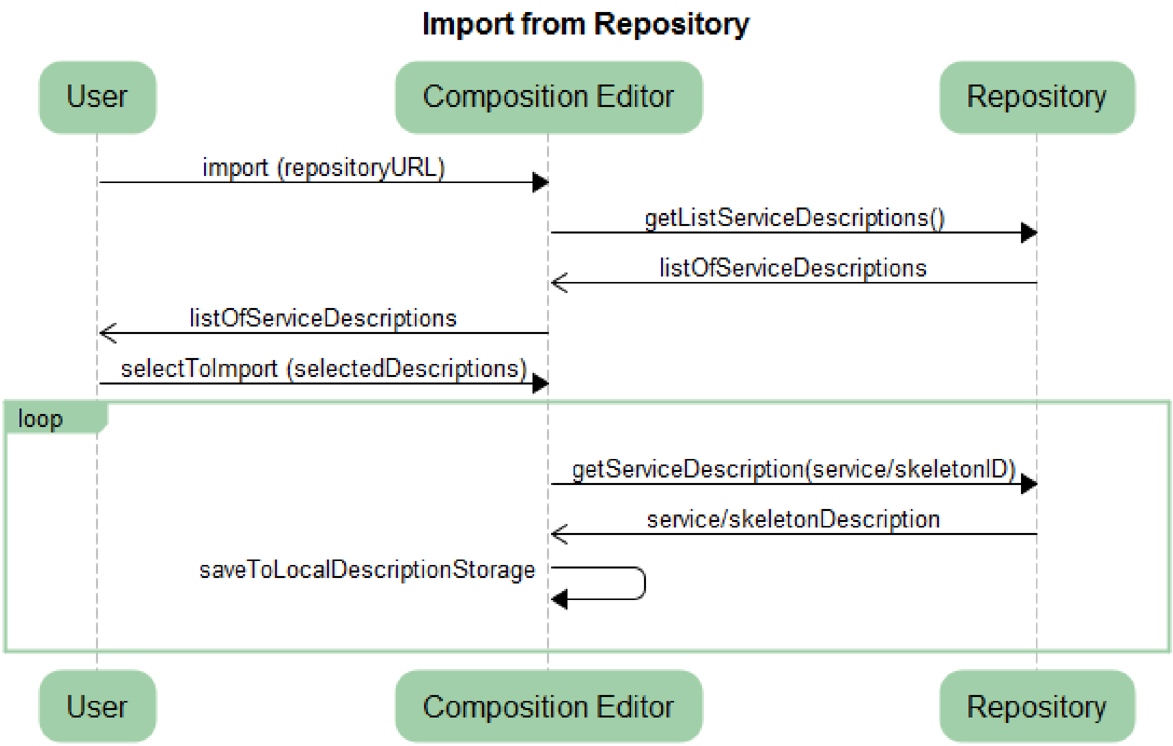
  In building the composition several building blocks are added iteratively, using the GUI. The building blocks expose data flow, control, and service invocation functionality (e.g. StartElement ServiceTemplate, Condition, StateManager Command, Goto, End). Connections between skeleton elements denoting result scope and partial order may be also edited. Two types of constraints are present in the context of skeletons, the skeleton constraint and the service constraints. The service constraints are being used for selecting the appropriate service during skeleton execution upon runtime. The service constraints are mandatory in the service template element and the skeleton constraint is optional in the skeleton start element. Once a skeleton is created it is stored in the local descriptions storage. Subsequently the user can browse and edit available skeletons.



Create New Skeleton

31.6.1.2 *Import Service Descriptions and Skeletons*

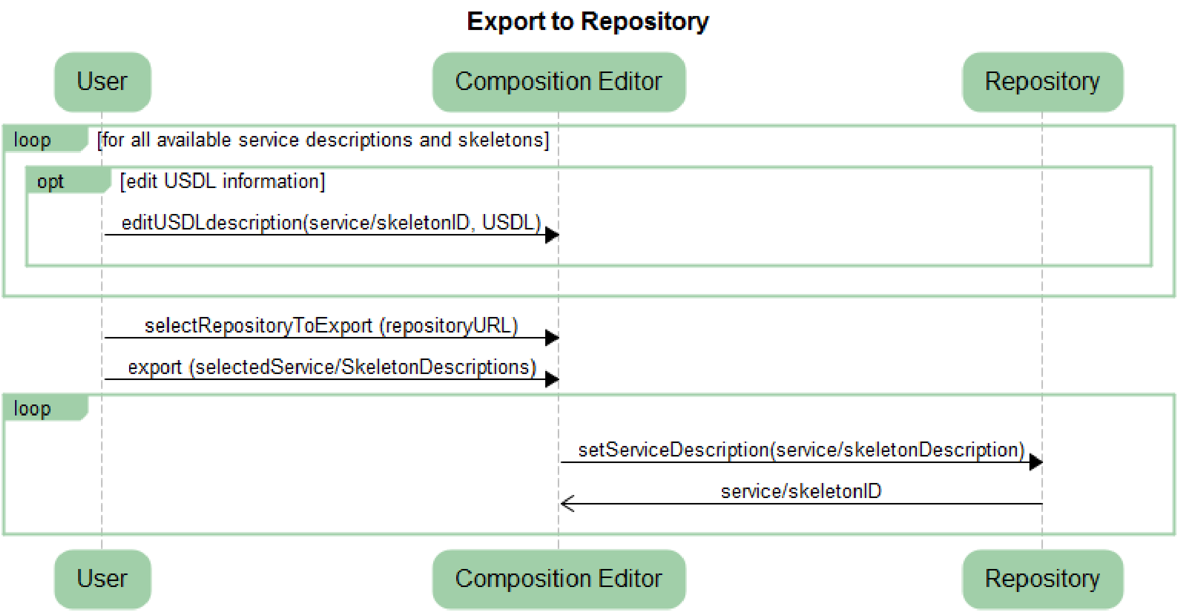
This operation imports (composed) service descriptions from a Repository GE to the local descriptions storage. Out of the list of available service descriptions and compositions only a subset may be selected. Note that from the perspective of the Repository and the USDL description part there is no differentiation between compositions (i.e. skeletons) and the other service descriptions. The Composition Editor however will make the difference and can edit and deploy the composed services in an execution engine to be run, while simple service descriptions can be used by the editor only as a component service in a skeleton and needs to be already up and running when the composition is triggered.



Import from Repository

31.6.1.3 *Export Service Descriptions and Skeletons*

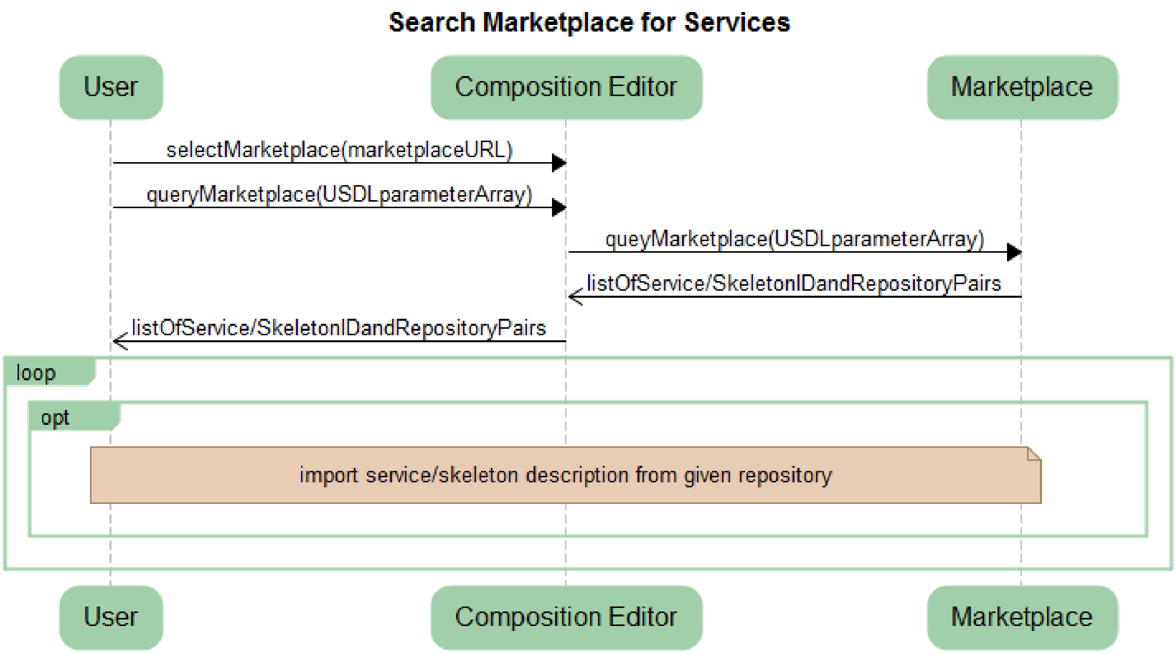
This operation exports (composed) service descriptions from the local descriptions storage to a Repository GE. Out of the list of available service descriptions and compositions only a subset may be selected. The description of a (composed) service may contain a USDL description providing a high level business description in addition to the technical description for the use of an execution engine. The latter may provide both the description of the API technology used for exposing/using this service and the composition skeleton that describes the runtime execution of the service in a formal composition language (if applicable).



Export To Repository

31.6.1.4 *Search Marketplace for Services*

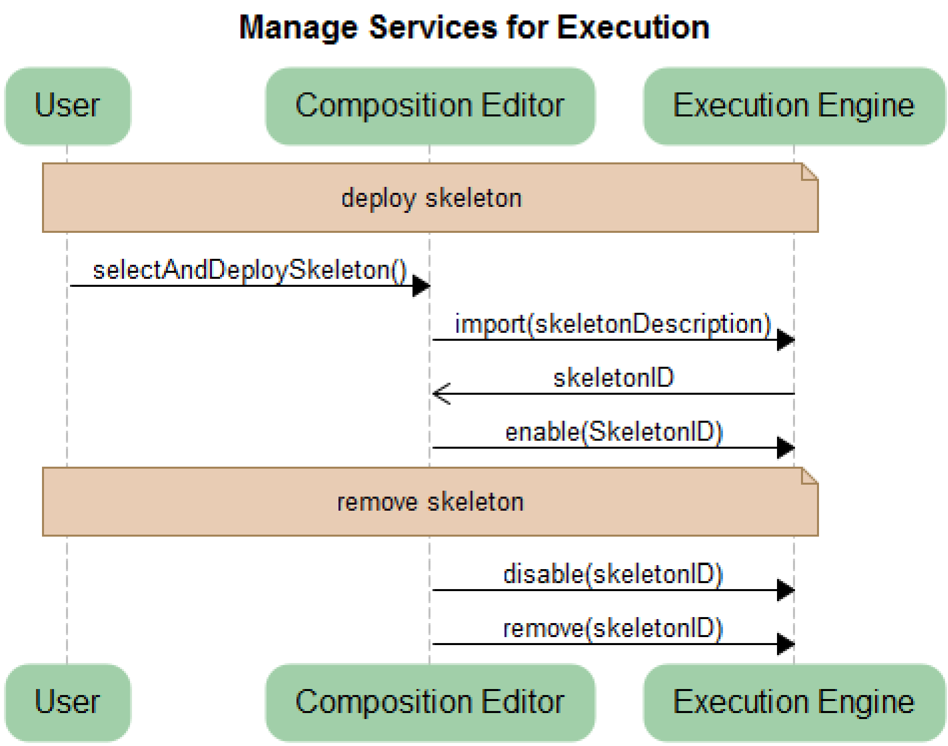
The Composition Editor may allow end users to search for the service they need. To do so, Marketplaces need to be queried, and the editor may allow detailed query construction based on constraints on USDL and other technical service description parameters.



Search in Marketplace

31.6.1.5 *Manage Services for Execution*

Through this UI the user may control how the service is deployed on the execution engine and specify execution parameters such as logging and tracing. Moreover the UI may be used to visualize results and additional information associated with previous runs.



Manage Execution Example

The interactions described below are basic functionality provided by the execution environment to an external controller. The editor is using these functions and exposes them to the end user for controlling the execution of the composed services.

- **Import**

This operation imports a service or a skeleton description (e.g. from the Repository GE) into the local descriptions storage of the execution environment.

- **Export**

This operation exports a service or a skeleton description from the local descriptions storage (e.g. to the Repository GE).

- **Remove**

This operation removes a service or a skeleton description from the local descriptions storage. Only disabled skeletons and services can be removed.

- **Enable**

This operation prepares a service or a skeleton description from the local descriptions storage for execution. In case of a skeleton it instantiates the composed application skeleton and makes it ready to be triggered for execution. Only enabled services will be considered during the skeleton execution.

- **Disable**

This operation removes a service or a skeleton description from the executable list.

31.7 Basic Design Principles

- **API Technology Independence**

  The API abstracts from the concrete implementation technology. Implementations using various kinds of platforms and frameworks should be possible.

- **Web Browsers do not have to limit the functionality of the editor**

  Modern web browsers as alternative to other GUI frameworks can and should be used fully implement the editor's capabilities.

- **User-matched interaction abstraction level**

  Editors could cater for different user expertise (from technical experts with skilled in the composition language to domain experts without technical expertise or even simple end-users with no programming or technical skills) and roles (from composed service creators, to resellers and finally to prosumers) by hiding complexity behind different types of construction blocs, trading off flexibility for simplicity.

- **The specification of the Service Composition GE is not tied to a particular technology for storing (composite) service data**

  The specific technology used for storing the inventory of services and their respective associations is not tied to any type of storage solutions, being opened to final implementations through SQL (MySQL, Oracle, ...) or No-SQL systems (MondoDB, Casandra, ...).

- **Service execution isolation**

  Service execution does not have to interfere in the execution of other widgets.

- **Composite service exposure via different API technology**

  Depending on what CEAs are available, a composed application can be exposed to the outside world via different API technologies

- **Execution engine deployment**

  An execution engine implementation might be highly distributed and scalable, if intended to be used by many parties on a global scale.

32 FIWARE ArchitectureDescription Apps ServiceMashup

You can find the content of this chapter as well in the wiki of fi-ware.

32.1 Copyright

32.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

32.3 Overview

Mashups are web applications which combine content and services from various sources in a value-adding manner. This composition of services and content can become a crucial business enabler in combination with the Internet of Services.

Mashups often focus on a very specific situational need and typically use web technologies (e.g. SOAP or RESTful services, or RSS feeds). For the mashup development, expert programming know-how and development environments have been required until now.

Web, telco, media services and content often are accessable via APIs (Application Programming Interface). APIs cover a set of functions that one computer program makes available to other programs so they can talk to the APIs directly. www.programmableweb.de lists more than 4200 APIs. Very popular APIs are google maps, twitter microblogging service, flickr photo sharing service, youtube video sharing and Google search.

The Service Mashup GE is implemented as a tool called Mashup Factory. Mashup Factory is an experimental toolset of DT, which allows end users without programming know-how to compose their own services for their immediate needs in communication, organisation and information.

The Mashup Factory toolset supports a graphical composition style, which allows the combination of services and content from several areas (e.g. communication, multimedia, geolocation). It supports integrated APIs from developergarden.com (e.g. SMS, Conference Call) by Deutsche Telekom and external sources (e.g. Google geo data, translator, weather forecast).

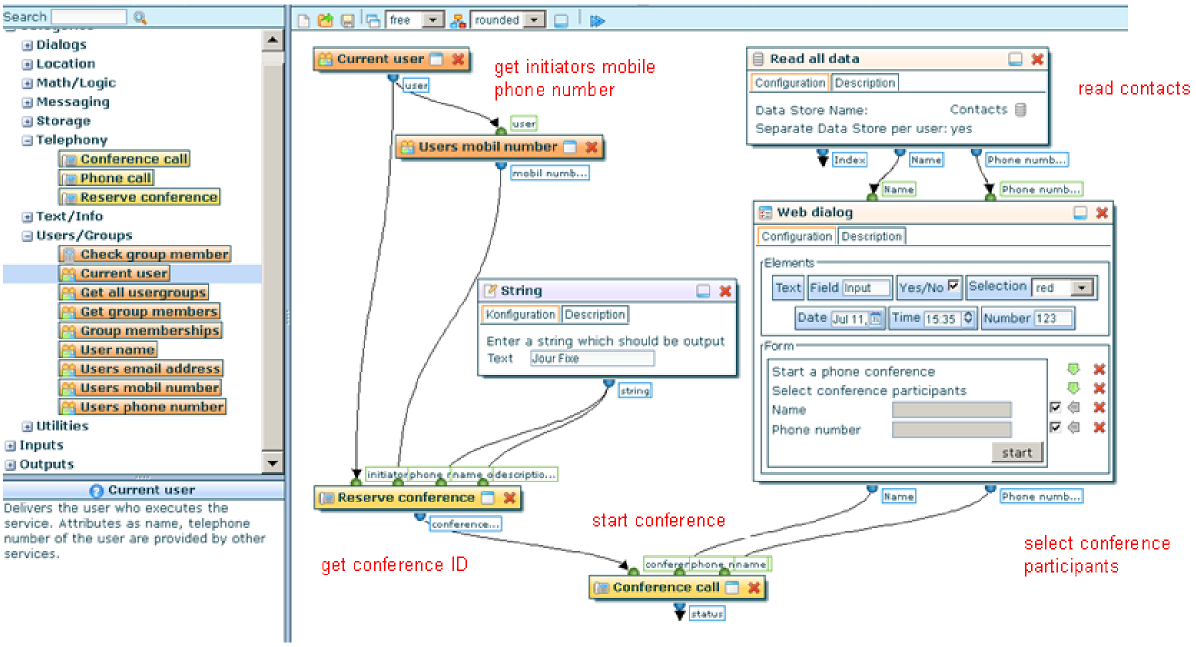
32.4 Basic Concepts

Mashup Factory is an experimental web-based application which supports a user to compose and execute mashups in an intuitive workflow. During the composition of the mashup, user activities like design, creation, configuration and simulation are enabled. After the mashup is designed, it can be activated, executed and monitored by the user. The mashup can be exposed to different consumer environments (e.g. portal, social network) where authentication and invocation can be performed. Technically speaking, the communication between the separated tasks are realised by dedicated service descriptions and service lists.

32.4.1 Build-in Service Repository

During the composition and design process the user can discover and use services from a repository.

Also we envision a large number of repositories containing service descriptions, which also might refer to descriptions in other repositories. Repositories can be hosted by the provider or a provider may use repository services of platform providers. The latter might be an alternative for small sized providers, which don't want to provide an own infrastructure. The figure below shows a screenshot of an example mashup which allows a user to initiate a phone conference with friends, whose phone numbers are provided from a data store. The service composition consists of basic services for conference call, data retrieval and a web dialog for the selection of conference participants from a user list.



Composition editor of Mashup Factory

The services can be catagorized into three categories: application logic (e.g. sending SMS), data (e.g. storing data) and user interface (e.g. creating web dialog). Each service has a dedicated interface, i.e. input and output parameters of particular types. The repository can be extended by other services.

For experimentation purposes we have included services of different areas in the repository:

- a click-to-call (1-1) and a click-to-conference call for telephony
- email and SMS for messaging
- store, retrieve, modify and remove data for structured data storage
- a design-time based editor to configure user interaction structures with input and output parameters
- content services: weather, bible verse, and translation services
- geo position, geo distance, google places and displaying geo positions within Google maps

- several services to support supplementary functions (e.g. text concatenation), logical functions (e.g. filter), user data functions (e.g. phone number).

32.4.2 Composition Editor

The services are graphically representated as boxes; input and output parameters are represented as ports. The user designs connections between services by linking equivalent input and output ports, represented by links. This allows the service creator to develop the mashup in a dataflow oriented composition style by combining services.

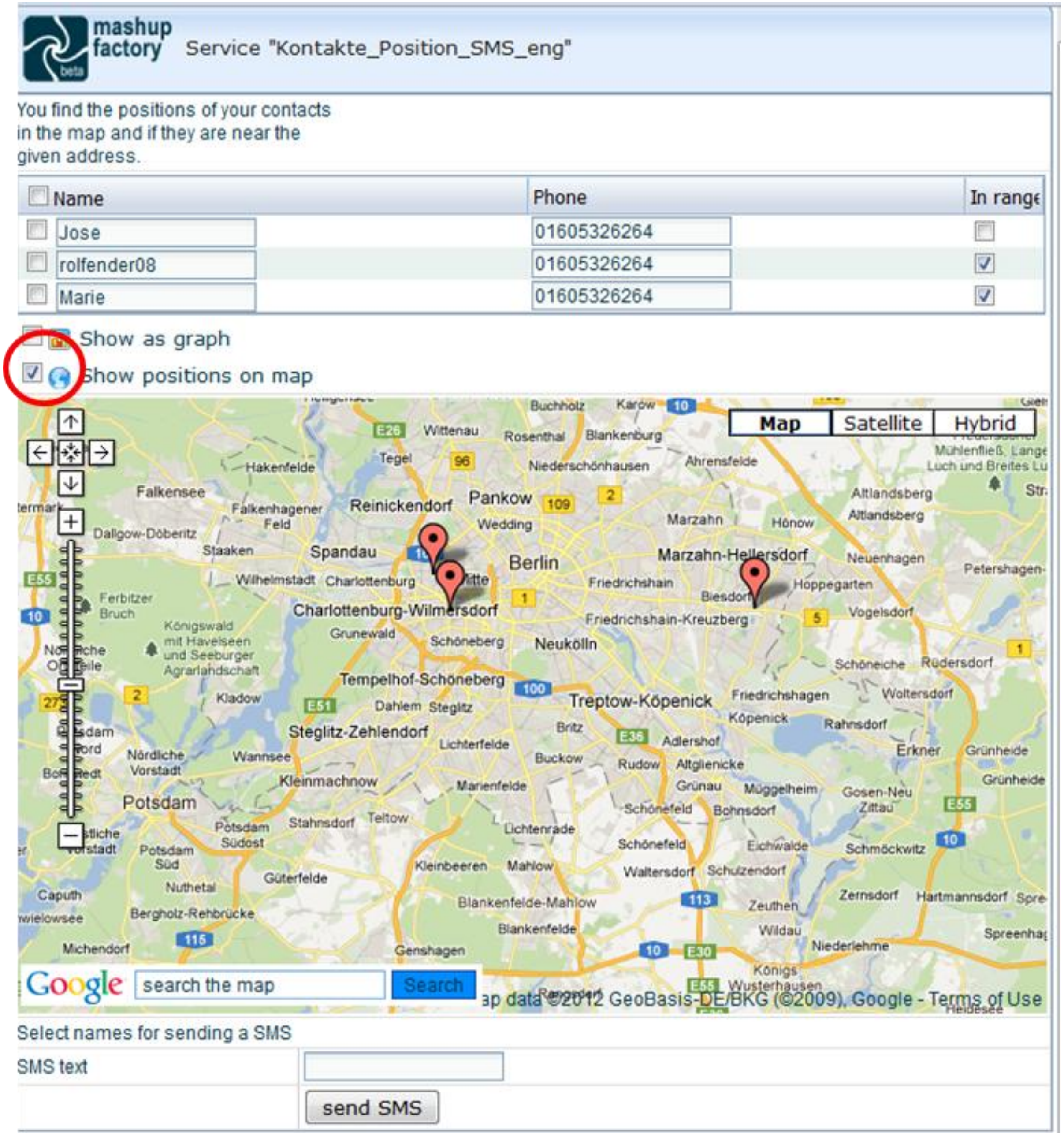
The user interface combines drag-and-drop features for services and simple textual editing functions for configurational purposes. Simple control constructs (e.g. filters, logical operations) are also available. Services are added via drag & drop from the repository to the work surface. A short service description is given in the left corner. Services have input and output ports, services are connected via linking input and output ports. There is a type checker which allows to connect only correct data types, e.g. date with date. Certain services are parameterized with configuration data, e.g. the web dialog. This service adds support for communication with a end-user in a web browser and transfer the user data to other services.

The composition determines which services are integrated in the mashup, and which output is produced. The output (see figure below) of a mashup composition can be

1. a web user interface (e.g. a digital diary)
2. an executed application (e.g. a telephone conference) or
3. a mixture of both (e.g. a web-based poll distributed by SMS).

When a service is composed (i.e. all input ports are connected with an output port), it can be saved as a composition by giving it a name. Service can be debugged by going step by step through the service watching the parameters transferred between the services.

The figure belows shows a screenshot of a service output with the geoposition of users displayed in google maps, with the option to send an SMS.



Output of a mashup

32.4.3 Service Execution

During the execution of a mashup the services are orchestrated in a data flow style, i.e. a service is executed when all input parameters are available.

The lifecycle of service execution can be managed by activating, stopping, renaming or removing services. The status of a service can be monitored and the period of activation defined. Dedicated users and user groups can be given permission to consume the services.

The execution is performed by a (standard) BPEL (Business Process Execution Language) engine. The services are called via SOAP requests and executed under the responsibility of the service provider. Thus, QoS of the services and the mashups cannot be guaranteed by Mashup Factory.

32.4.4 Users and Groups

Mashup Factory allows the management of users and groups. Some attributes (e.g. email address, phone number, login name) of the users can be edited and they can be organised to groups. User attributes can be retrieved by a service composition at run time. Every service composition can be given permission which users and groups (and their respective users) are allowed to execute it.

Caution: The users of Mashup Factory are responsible for the data of test users, keep legal constraints concerning data security.

32.4.5 Example Scenarios

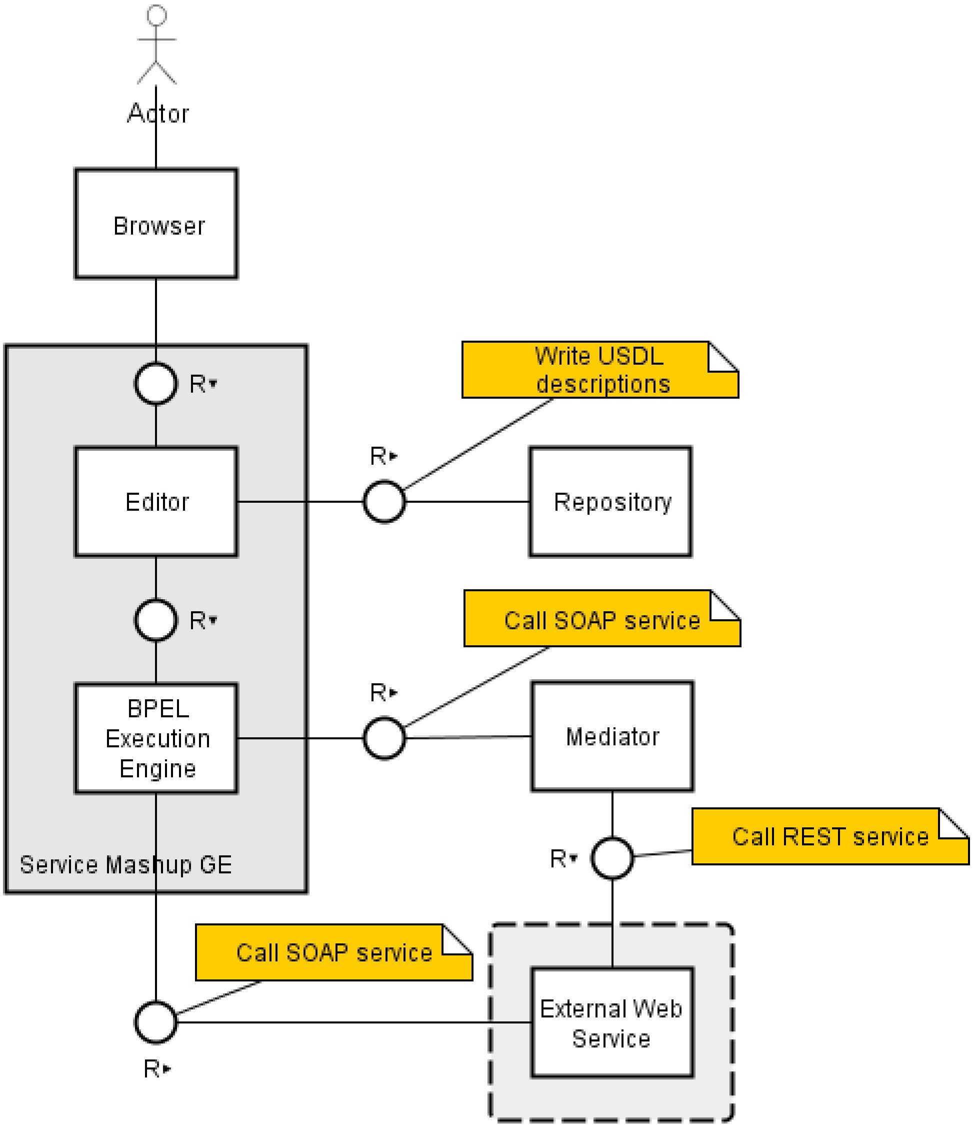
In the following section the composition process and some illustrative application scenarios will be described. Imagine a person who frequently organises (trekking) tours of a group of friends. He wants to reduce his effort to contact his friends for finding adequate dates and notify the group with relevant information. He builds a mashup by Mashup Factory which uses a weather service, SMS and a web based survey.

Other possible scenarios for supporting immmediate needs in communication, organisation and information are

- providing a digital diary for monitoring health data with weight, blood pressure data where an SMS will be sent to a supervising person in case of exceeding a threshold
- conducting a web based survey for a (sports) club
- sending SMS to customers if they are close (related to a geo position) to a defined place (e.g. special offers in shop)
- inviting club members via voice mail to a day of an open door
- distributing particular content (e.g. bible verses, pollen information) to an interested audience
- performing video transmissions to selected persons, e.g. for health training, education, team meetings

For realising the scenarios, appropriate services must be integrated and perform their services with an adequate level of quality.

32.5 ServiceMashup Architecture



Service Mashup Architecture

The Mashup Factory follows the architecture of a Rich Internet Application. All user interaction takes place via Web browser as user agent, all application logic is implemented on a Web server. There are no software components to be installed on the client side except for the Web browser.

32.5.1 Technical Interfaces

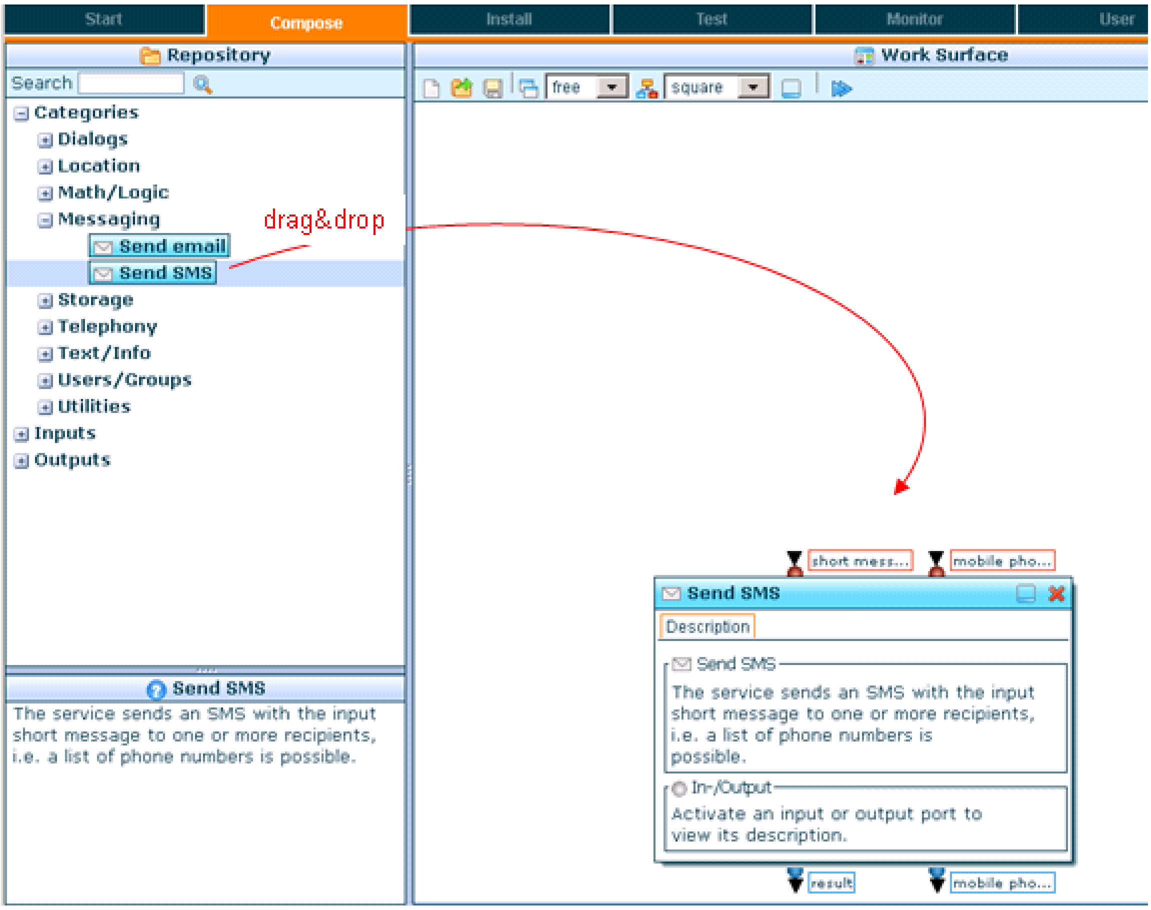
The Web browser communicates with the Mashup Factory server using HTML and Ajax technologies. The Web server needs to comply to the Java EE servlet container specification. The Editor communicates to the BPEL Execution Engine and the Repository GE via REST interfaces, the Mediator GE and all other external Web services are called from the BPEL Execution Engine via SOAP protocol. For the purpose of storing the BPEL service descriptions and other internal data an XML database is used that is also hosted in the servlet container.

32.6 Main Operations

To give an idea how to use and interact with the Mashup Factory to create Service Mashups this section provides a step-by-step introduction to compose examplary services.

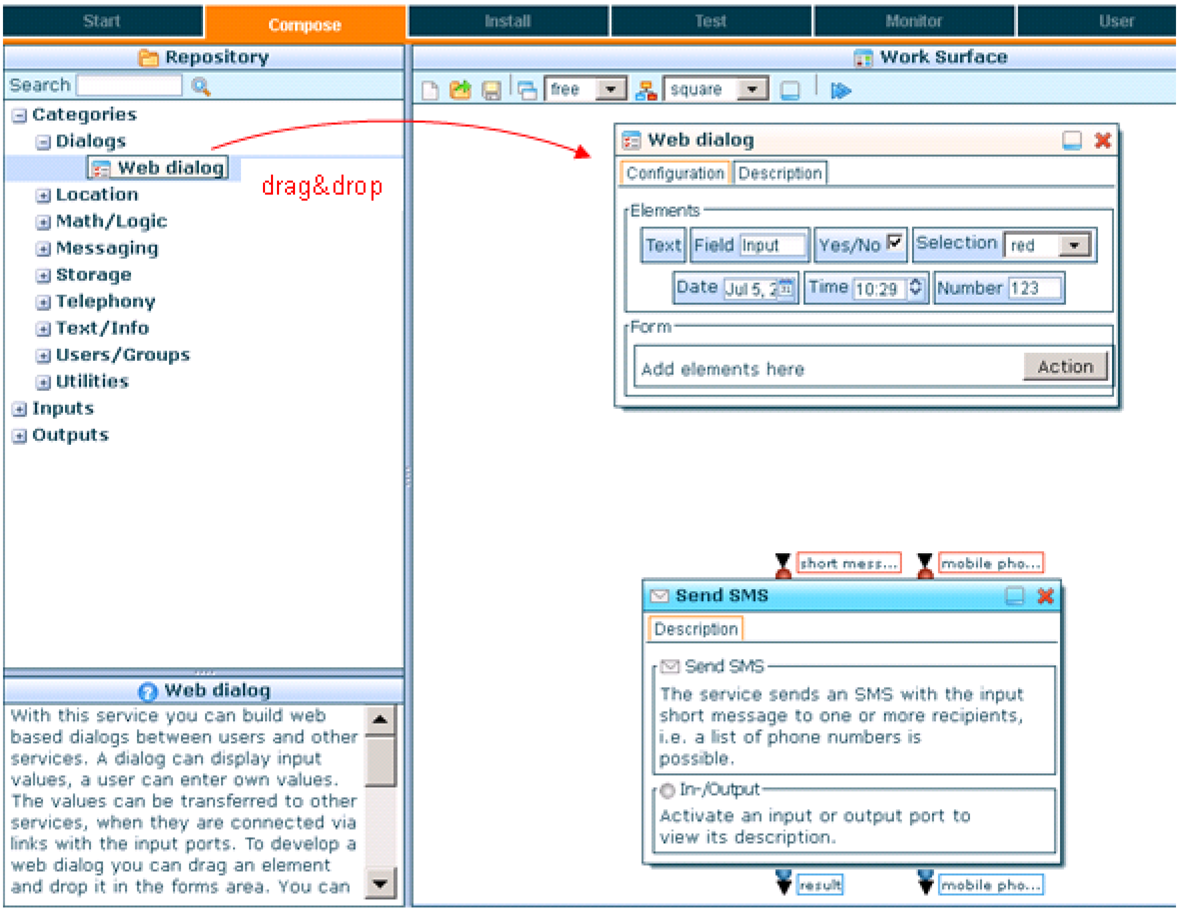
32.6.1 Send SMS Service

A "send SMS" service composition depicts the easy composition process with Mashup Factory and illustrates the functions and components. Purpose of the service is a web dialog to send a SMS, which can be used by an authorized user by entering the message text and the phone number. In the Compose section you browse the repository for “Send SMS”. Drag this box with the mouse and drop it on the work surface.



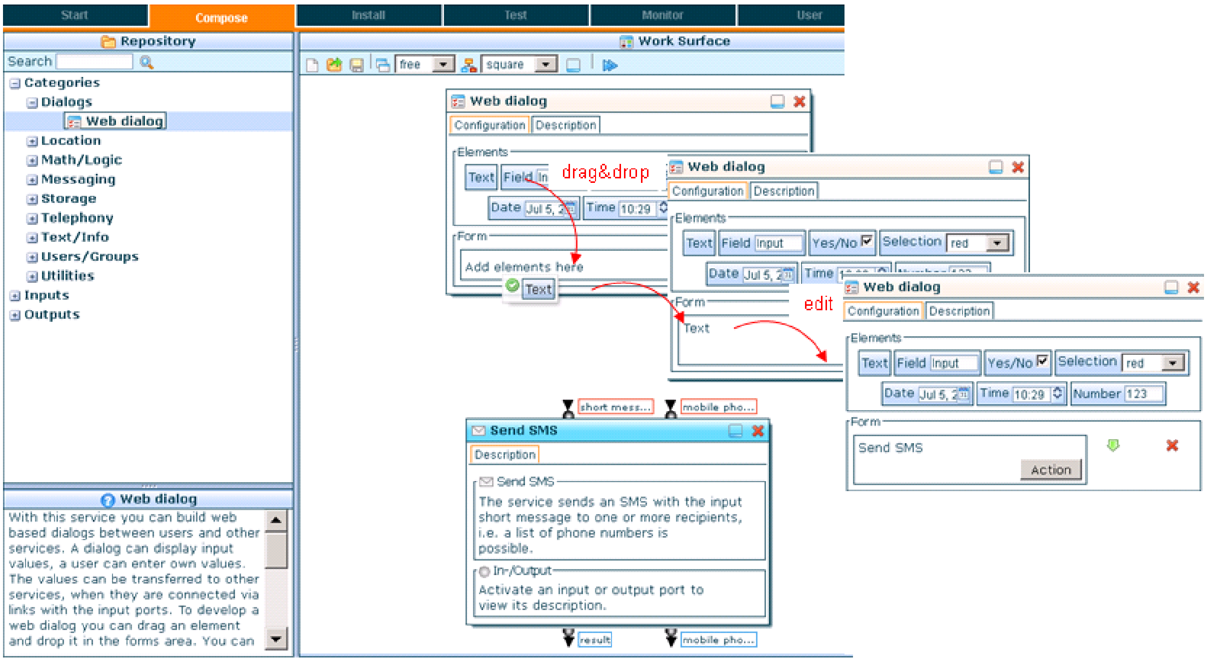
Send SMS Service - Step 1

The “Send SMS” service has two input ports which have to be connected with other services. Thus, fetch the Web Dialog from the repository and drag it on the work surface.



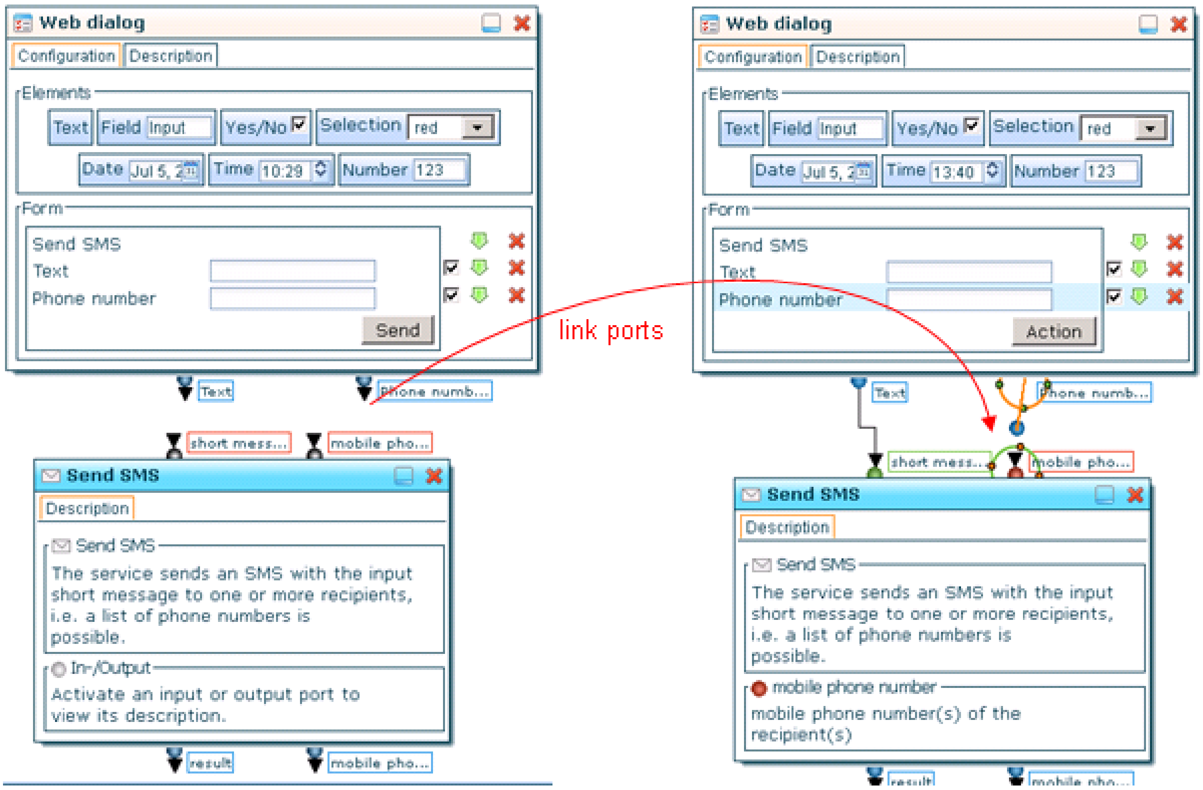
Send SMS Service - Step 2

In the Web dialog you can edit the user interface which later is displayed in the web browser. Relevant data fields of the user who consumes the service, are added herein. To create the appropriate dialog you can drag an element from the Elements area of the Web dialog and drop it in the Form space. Start with dragging a text element, position it, and edit it by double clicking on the Text field. Enter “Send SMS” and press return.



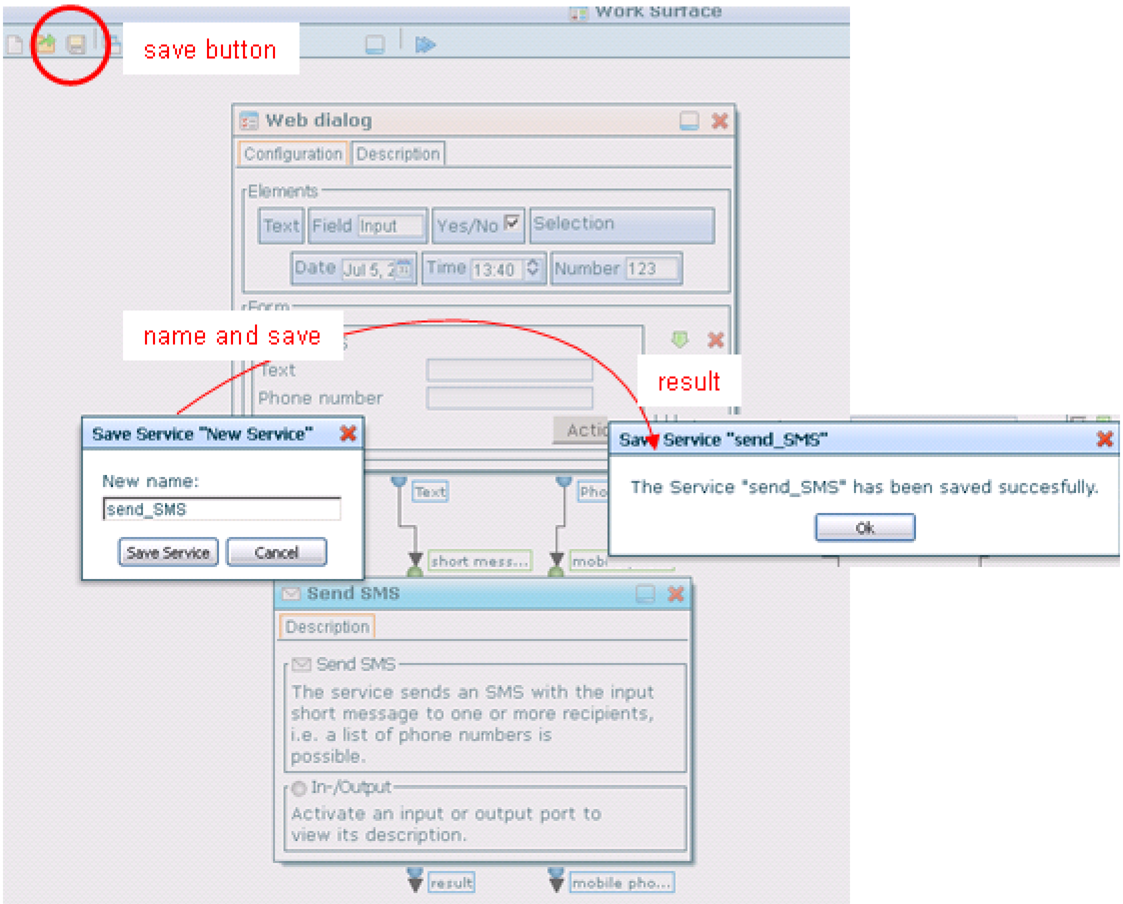
Send SMS Service - Step 3

Next, enter the message text and the phone number by using the field Element. You find two new output ports below the Web dialog box Connect the output ports with the corresponding input ports of the Send SMS.



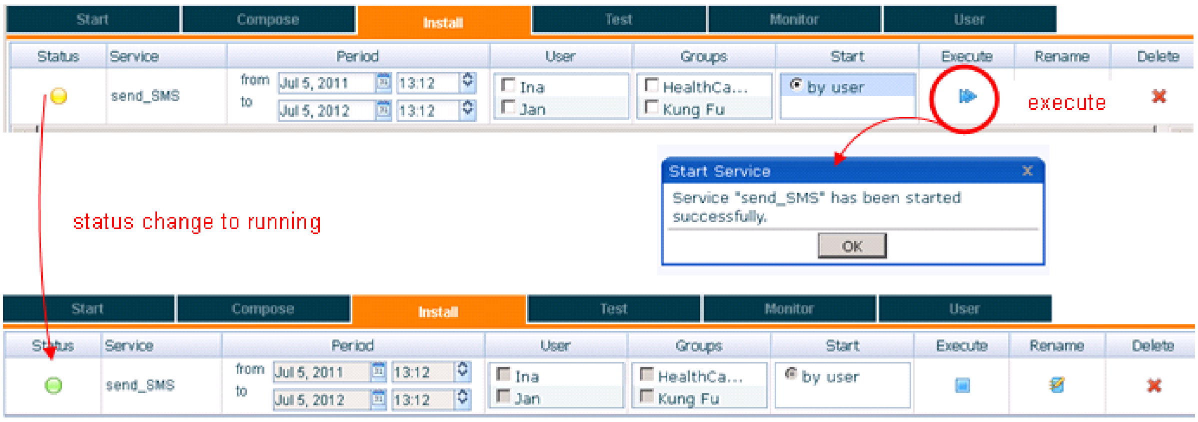
Send SMS Service - Step 4

That's it! Your first service composition is created. Now you save it by giving it a name. You get a result message in case of storage.



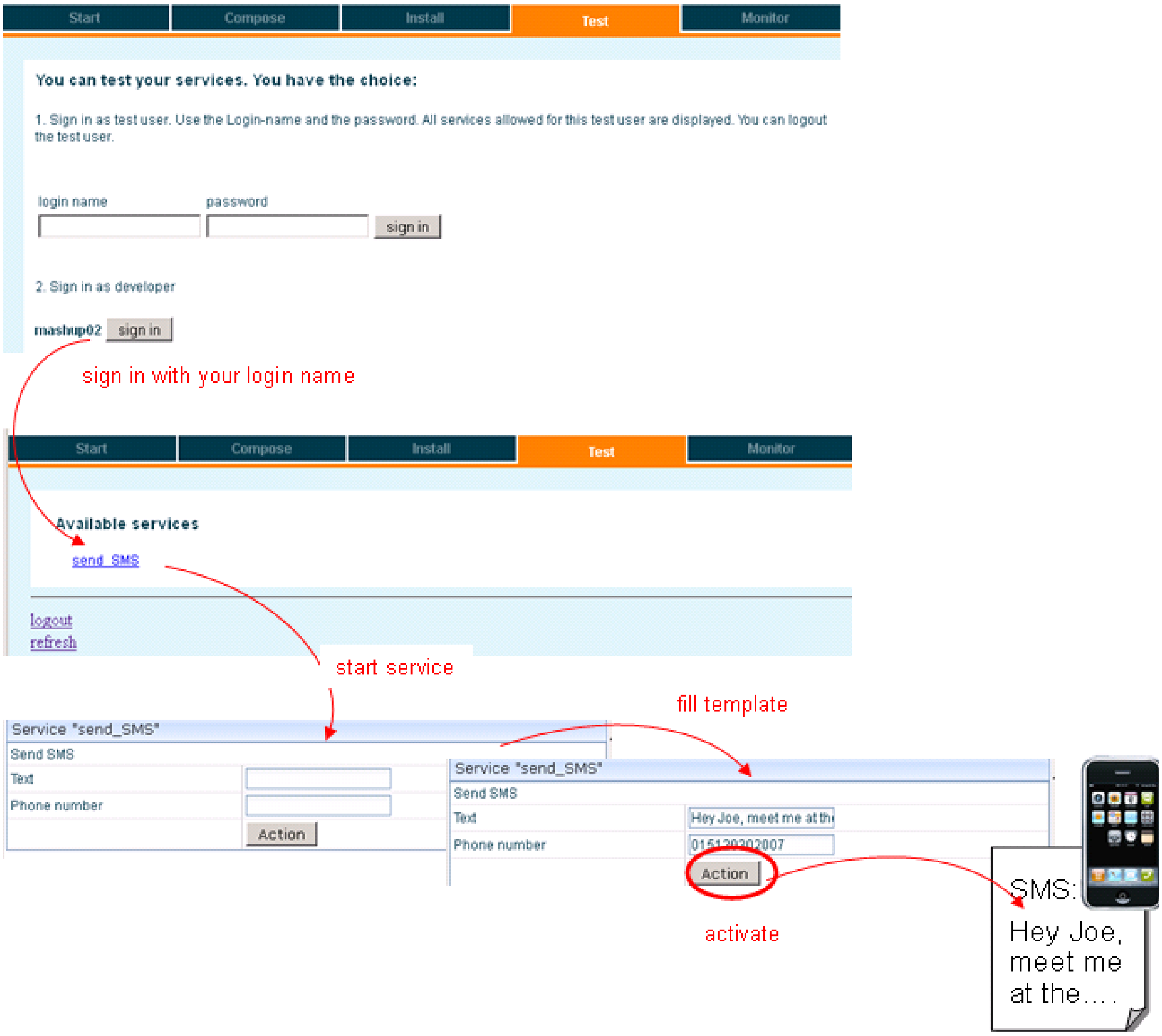
Send SMS Service - Step 5

Install the new service by pushing the execution button. If the status turns to green, the service can be executed.



Send SMS Service - Step 6

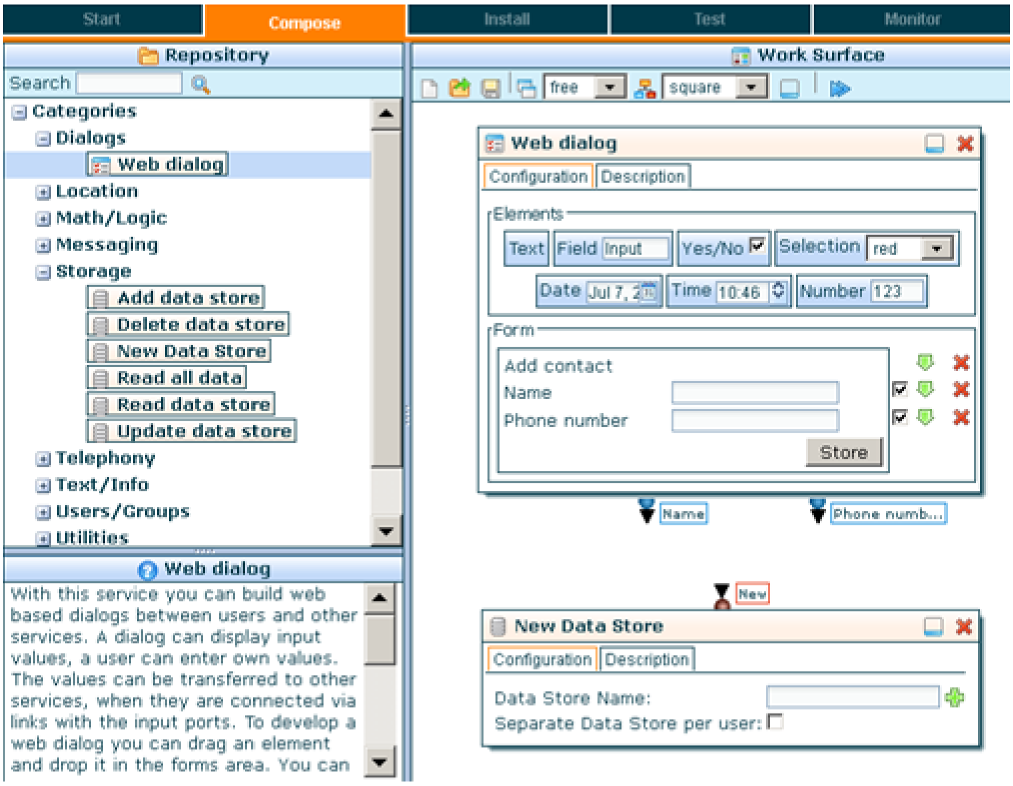
Test the new service and login under your user name as a developer. You find the new service as link and start it by pushing the link. A new tab is opened with the web dialog you have designed. Fill in data, activated the action button and the SMS is sent!



Send SMS Service - Step 7

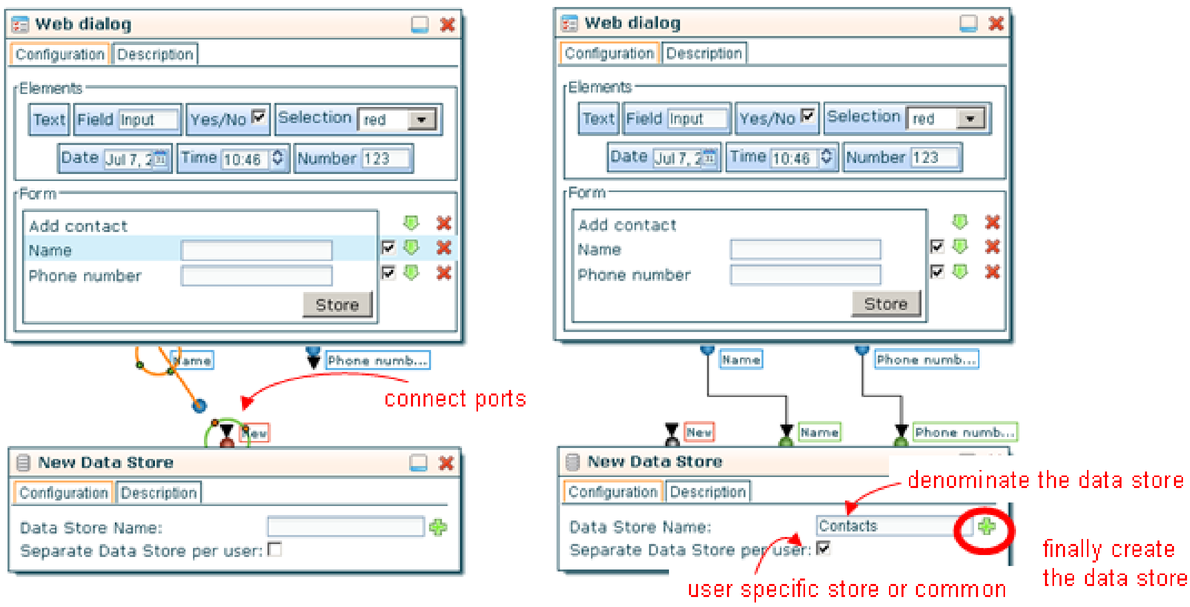
32.6.2 Using Data Stores

A data store can be used to store and retrieve data for your service composition, e.g. diary data, addresses, locations etc. Once you have created a data store you can add, modify, read or remove the data. A simple addressbook with the two services (add a new contact and read my contacts) illustrates the handling. For adding a new contact you drag the 'New data store' from the repository. The input port is connected with one or more output ports from other services. Here a 'web dialog' with a Field Element Name and Field Element Phone number are used in order to allow the user to enter the data



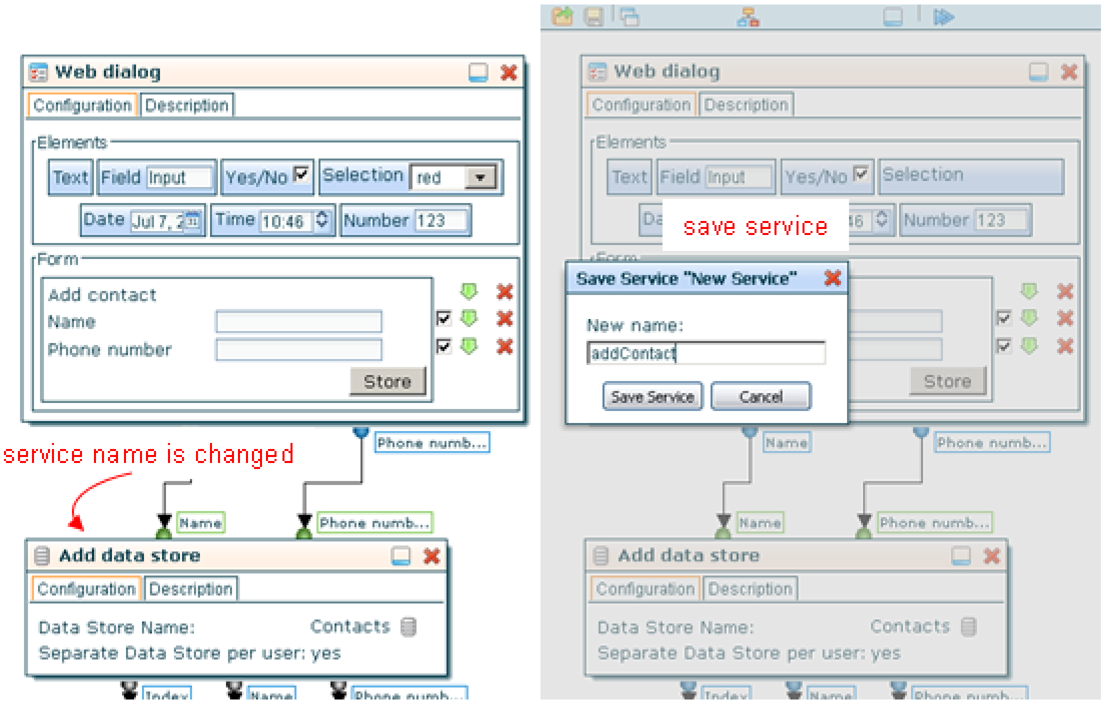
Using Data Stores - Step 1

After connecting of an output port (e.g. Name of ‘Web Dialog’) to the input port New (of ‘New Data Store’), the port is added to ‘New data store’. When all input data are defined for the data store, denominate it with a name (e.g. Contacts) for the data store. You can decide, whether the data store is user-specific (e.g. diary) or common (e.g. survey) by setting a flag.



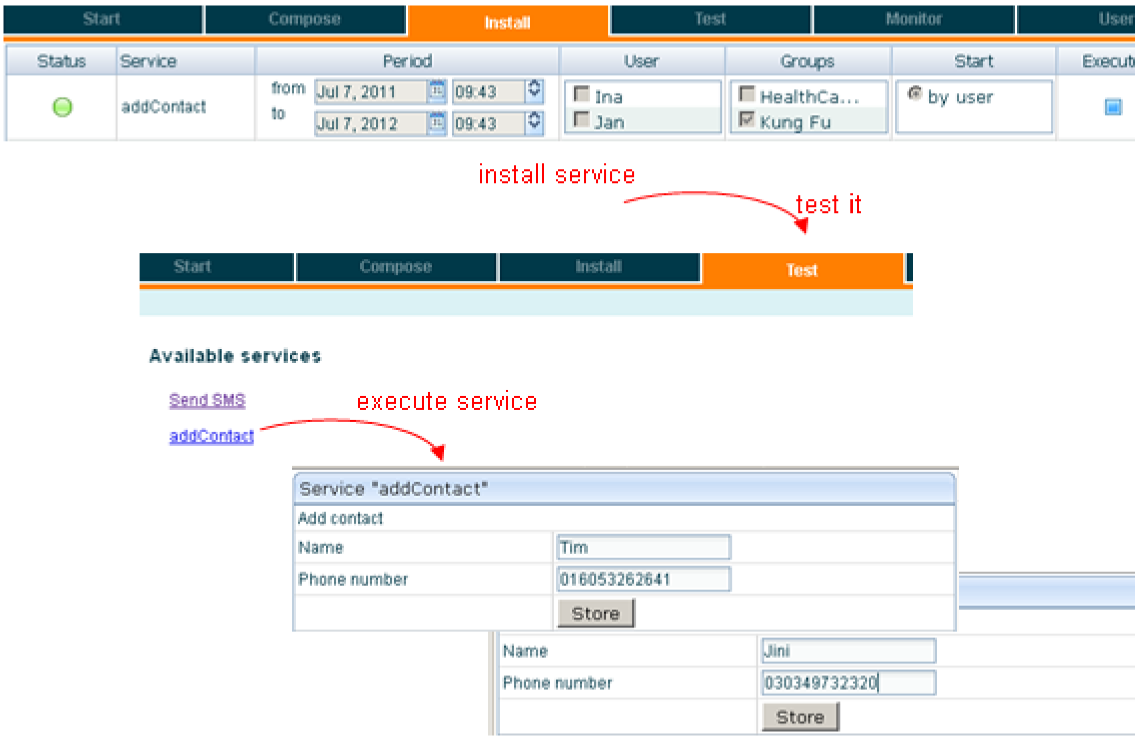
Using Data Stores - Step 2

Click the + to save the store. After storage the name of the service is changed to ‘Add data store’. Then save the new composition by giving it a name (e.g. addContact).



Using Data Stores - Step 3

You install it and test it by clicking on the link. Enter data in the dialog and repeat it with some contacts.

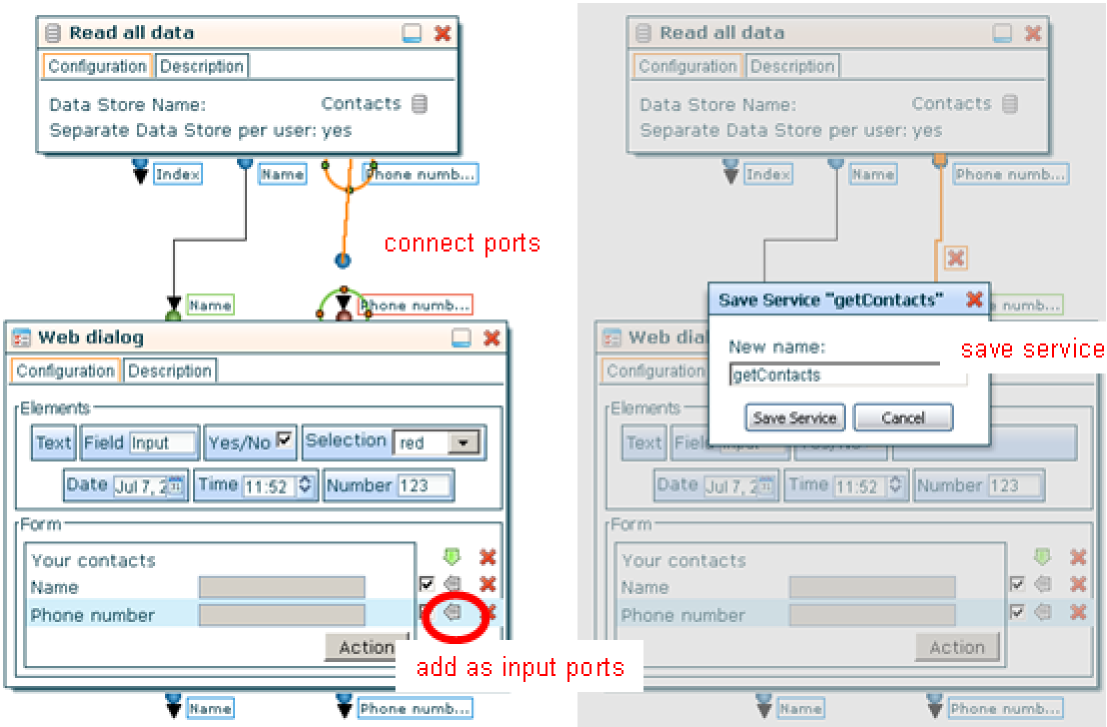


Using Data Stores - Step 4

For displaying the data of your contacts, create a new service for reading the contacts. Drop the 'Read all data' service and choose Contacts from the Data Stores. The service has the output ports to read the data of Contacts (and in additional port Index, which can be used to modify the data).

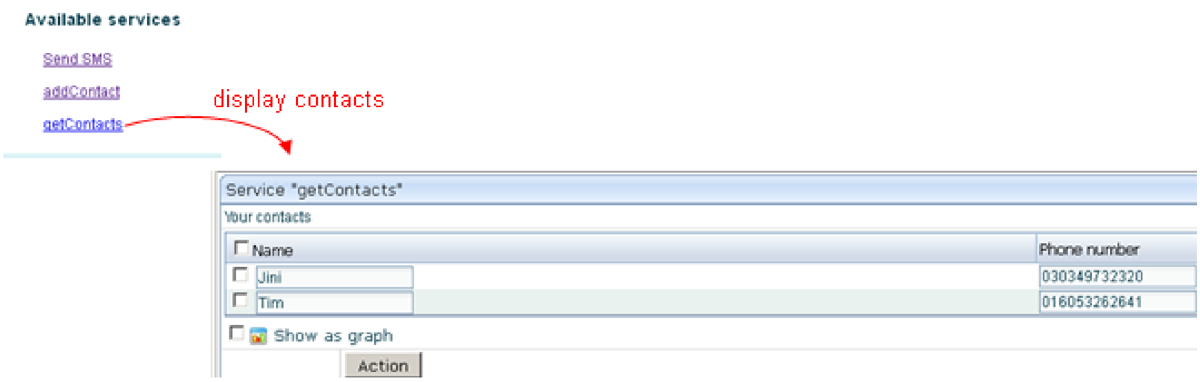
Using Data Stores - Step 5

To display the data you can use a 'Web dialog'. Configure the dialog by dragging and dropping the Elements. Mark the fields Name and Phone number as input ports by pressing the green arrow. Connect the corresponding ports and save the composition with the name getContacts.



Using Data Stores - Step 6

Install and test getContacts. You get a list of contacts, which you have added to your contact store.



Using Data Stores - Step 7

32.7 Basic Design Principles

The design goal of Mashup Factory was to implement a special purpose BPEL editor for non-experts using a graphical approach to express Service Orchestration. Any technical aspects of the BPEL specification should be hidden from the user experience by using a data flow oriented service composition approach. The underlying BPEL structures should be fully transparent to the end user.

32.7.1 Data flow paradigm

Web services are invoked by providing input data as parameters and retrieving the return values as result of the Web service call. This concept was remodelled to a data flow approach where the input data for a graphical representation of a Web service "flows" into input ports and the result data returned from Web service invocation is presented at output ports. These ports can be connected by virtually wiring the ports in the graphical editor as if the output data of one Web service flow into the input ports of other Web services. By wiring the different Web services the end-user defines the sequence in which the services will be called at runtime by the BPEL execution engine resulting in an Service Orchestration.

32.7.2 Typed ports

Web service parameters are typed and can even handle complex types. In order to keep this typed data handling in the graphical composition approach the input and output ports of the Web service building blocks are also typed. The wiring logic allows only to connect ports of the same type (since it isn't of much sense to "feed" a user name into a port which expects geo coordinates). The data typing uses a two level hierarchical design. The lower level denotes a primitive type like Integer or String the higher level abstracts from the primitive type by defining a semantically meaningful type like "username", "date" or "geo position". Only the semantically meaningful types are presented at the user interface of the service editor. The only exception are generic service components like the dialog service, that allows end-users to build customized forms which will be visually displayed at service execution time. For the sake of clearness only form components/fields with primitive types may be used in the form editor (as there are too many high level types). The wiring logic takes the underlying primitive types into account when a port of the form service is connected to an ordinary port (e.g. a form port of type String is allowed to connect to a port of type "phone number" because the underlying type is also String).

32.8 Detailed Specifications

32.8.1 Open API Specifications

The Service Mashup GE Mashup Factory is not exposed as a service but as a Rich Internet Application, accessed throught the end user's Web browser. Although some GE components are exposed as services, they only expose an API for internal consumption (within the GE), but it is not foreseen that they will be integrated by other GEs.

The Mashup Factory will access the Mediator and Repository REST APIs:

- FIWARE.OpenSpecification.Apps.MediatorREST
- FIWARE.OpenSpecification.Apps.RepositoryREST

33 FIWARE ArchitectureDescription Apps ApplicationMashup

You can find the content of this chapter as well in the wiki of fi-ware.

33.1 Copyright

33.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

33.3 Overview

Web application mashups integrate heterogeneous data, application logic, and UI components (widgets/gadgets) sourced from the Web to create new coherent and value-adding composite applications. They are targeted at leveraging the "long tail" of the Web of Services (e.g. the so-called Web APIs, which have proliferated during recent years and have doubled in number during 2012. See programmableweb.com) by exploiting rapid development, the Do-It-Yourself (DIY) metaphor, and shareability. They typically serve a specific situational (i.e. immediate, short-lived, customized, specific) need, frequently with high potential for reuse. It is this "situationality", which prevents them from being offered as 'off-the-self' functionality by solution providers.

Web application mashups can be manually developed using conventional web programming technologies, But this fails to take full advantage of the approach. Application mashup tools and platforms such as the one being specified by the FIWARE's *Application Mashup GE* are aimed at development paradigms that do not require programming skills and, hence, target end users (being them business staff, customers or citizens). They also help to leverage innovation through experimentation and rapid prototyping by enabling their users (a) to discover the best suited mashable components (widgets, operators and off-the-self mashuplets) for their devised mashup from a vast, ever-growing distributed catalogue, and (b) to visually mash them up to compose the application.

Key features of the *Application Mashup GE* to be covered by its open specification are:

- Support for a Platform-independent Mashup Definition Language (MDL) and a Widget Definition Language (WDL), which are needed to describe the application mashup and its building blocks so that any platform implementing the GE's open specifications will be able to instantiate and execute them. A mashup's MDL links to the WDL descriptions of its constituent widgets.
- Support for XML/RDF template schemas for both the WDL and the MDL languages, containing all widget- and mashup-related contextual information, plus all the preferences, state properties, and wiring/piping, context and rendering information (i.e. elements that manage the platform-widget interaction) required to support application mashup persistence and to instantiate and run the application mashup on a platform that conforms to these open specifications.
- USDL extensions to the WDL and MDL: WDL-RDF / MDL-RDF vocabularies for representing WDL/MDL data as part of a USDL offering.

- Support for a zipped file format (WGT) that allows mashable components to be conveniently stored and distributed.
- Support for *wiring*: a mechanism empowering end users to easily connect widgets in a mashup to create a fully-fledged event-driven dashboard/cockpit with RIA functionality. Different *wiring* editors are allowed provided that they give a MDL description of the resulting mashup.
- Support for *piping*: a mechanism empowering end users to easily connect widgets to back-end services or data sources through an extendable set of operators, including filters, aggregators, adapters, etc. Different *piping* editors (commonly offered as part of a *wiring* editor) are allowed provided that they give a MDL description of the resulting mashup.
- Support for visual rendering of the widgets in the application mashup UI. Different editors are allowed as long as they provide a description of the resulting mashup in MDL.
- MAC (widget, operator and mashup) life-cycle management support.
- An application mashup execution engine model capable of deploying and running a mashup from a MDL file. It provides support for managing mashup state persistence, and for managing the wiring and piping mechanisms.
- Support for interaction with a *catalogue* of mashable components: the catalogue empowers end users to store and share their newly created application mashups with other colleagues and users by communicating with the Store GE.

The Open Specifications of the FI-WARE's Application Mashup GE follows. These GE Open Specifications contain all the information required in order to build compliant products that can work as alternative implementations of the Application Mashup GE and therefore may replace any implementation developed in FI-WARE within a particular FI-WARE Instance.

33.3.1 Target usage

FI-WARE strives to exploit the composability of the application and services technologies in order to support cross-selling and achieve the derived network scaling effects in multiple ways. The platform enables composition either from the front-end perspective --application mash-ups- or the back-end perspective --composite services. Specifically, the Application Mashup GE targets composition from the front-end perspective and is expected to leverage the creation and the execution of value-added applications not only by application providers but also by intermediaries and end users acting as composers, a.k.a. prosumers. Prosumer are a consumer-side end users who cannot find an application that fits their needs and therefore modify/create an application mashup in an ad-hoc manner for their own consumption. As the capabilities and skills of the target users being considered are expected to be very diverse, all kinds of usability issues, conceptual simplification, recommendation and guidance, etc. are taken into consideration.

33.4 Basic Concepts

33.4.1 Key concepts and ideas

The Application Mashup GE describes a Web platform that helps users to easily and visually create and run their own Web application mashups. Its functionality can be divided into a client-side part running on the user web browser and a server-side part running on a web server.

The Application Mashup GE is based on the composition model presented in [1] (please refer to that publication for a detailed description of the underlying composition model of this enabler), which has been specifically designed to empower end users with few or no programming skills to create and share their own web composite applications in a fully visual fashion:

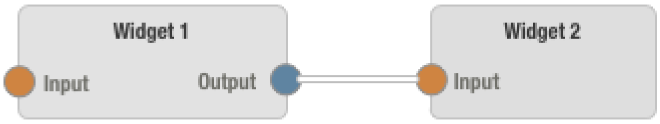
[1] Lizcano, D., Alonso, F., Soriano, J., and López, G. (2011) A new end user composition model to empower knowledge workers to develop rich Internet applications. In Journal of Web Engineering, Vol.10 No. 3, pp. 197-233, Rinton Press Inc. Sept. 2011.

Widgets are the key elements of the composition model that the Application Mashup GE must support. Together with connectors and mashups (mashups are considered as building blocks for other application mashups), they make up the complete set of Mashable Application Components (MAC, see Terms and Definitions) that the Application Mashup GE must support. A widget is a lightweight Web application that runs on the user's web browser, in the context of an Application Mashup GE implementation. Widgets are usually developed using current Web technologies (HTML(5), CSS, Javascript, ...) and they are bound to heterogeneous data coming from the Web (e.g. Web APIs). They can be regarded as the service front-end, because they offer users a graphical user interface (UI), so that they can easily get a visual representation of the service data and functionality to which the widget is bound.

The figure below shows an example of what iGoogle's widgets (one of the first products to implement this idea) look like:

An example of some isolated widgets coexisting in the same desktop

These *early* widgets *per se* are isolated applications that do not interact with each other. However, the Application Mashup GE aims at providing a mechanism to visually compose a fully-fledged web application from different widgets that can now interact with each other via events and data sharing. This mechanism is what the composition model calls **wiring**. The idea behind wiring is easy: widgets expose (data/event) inputs and (data/event) outputs, so that an output from one widget can be linked to other widgets' inputs following a composition technique based on pre- and post-condition mechanisms. This way, the Application Mashup GE manages the data/event flow between widgets. The mechanism allows for the use of event-driven *programming* features, e.g. a widget can send and event through one of its outputs on an event trigger. The figure below shows an example of the wiring metaphor:



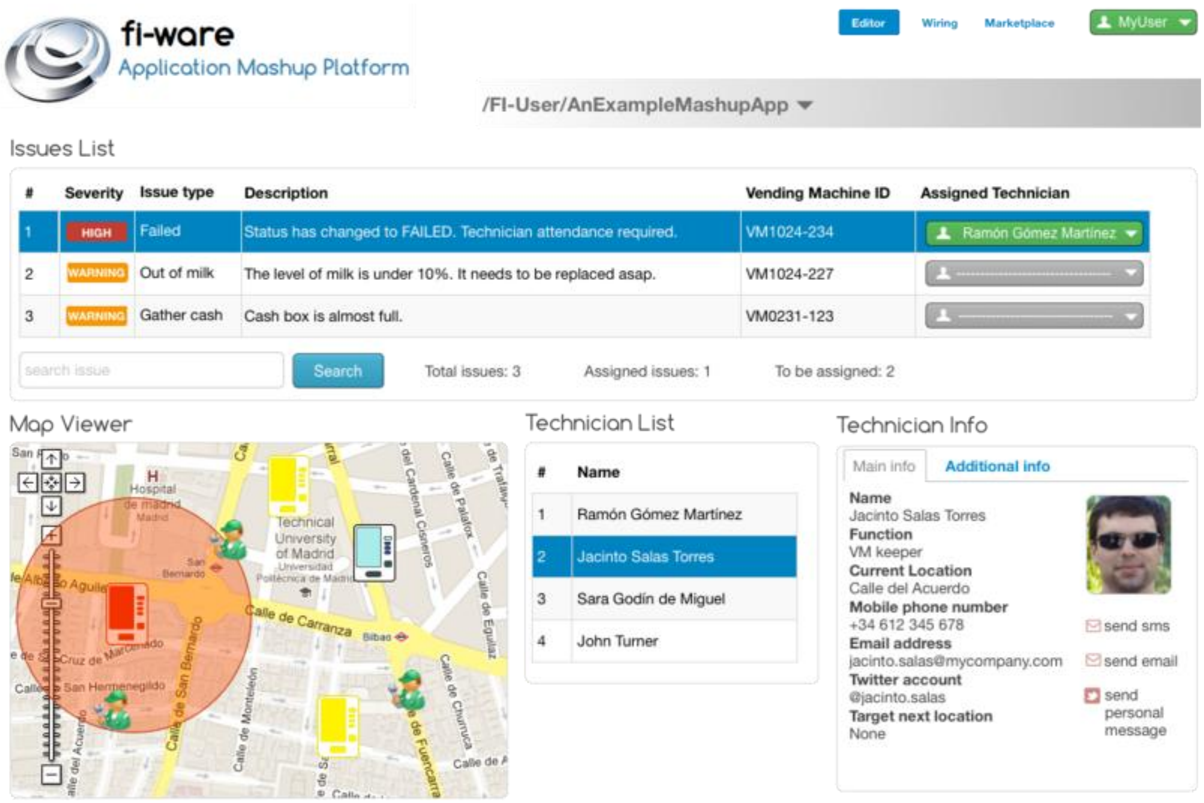
An abstract representation of how the Application Mashup GE could support the Wiring mechanism

Widgets supported by the Application Mashup GE must be able to access their data from services in at least the following two ways: *programmatically* or by means of *operators* and through a visual technique called *piping* that establishes how these operators can be combined to form a *pipe*.

The Application Mashup GE supports the invocation of services *programmatically* from the widget's code: through what we call *WidgetAPI* (see the Architecture and Open API Specifications sections). But, following the ideas from the composition model, it also supports operator use and the *piping* mechanism. This targets end users (i.e. users with few or no programming skills): an *operator* does not offer a GUI but, like widgets, has an abstract representation with both inputs and outputs and can thus be wired to widgets allowing the data flow between them. Operators are usually bound to some kind of data source (SOAP service, Web API, etc.). In other words, operators are configured out-of-the-box to get access to a backend service, but they can also be made to subscribe to and get events from a publish/subscribe system. They can also act as filters, aggregators, mediators, etc. when used in the *piping* technique to build a *pipe*.

To sum up, an implementation of the Application Mashup GE must support the process of visually creating a composite web application by composing different widgets using the wiring mechanism, which interconnects those widgets, and the piping mechanism, that makes use of operators to get access to new data, perform an operation on that data, and finally pass it to the widgets through their inputs.

The figure below shows and example of what a web application mashup looks like.



An implementation of the Application Mashup GE showing how the mashup developers can arrange widgets to create their own Web mashup

33.4.2 Example scenario

To illustrate what is expected from the Application Mashup Generic Enabler, we have borrowed the following example scenario from the FInest Use Case Project (http://www.finest-ppp.eu/). The scenario is part of its *Fish transport from Ålesund to Europe* use case:

"A fish producer needs to ship frozen/dried fish from Norway to a customer overseas. The scenario covers the feedering phase, i.e. the shipping from Ålesund to Northern Europe. The fish cargo is first delivered at the Port of Ålesund (ÅRH) and stored and stuffed in container at the terminal (Tyrholm & Farstad: TF). The shipping line NCL covers the North Sea voyage (feedering) from Ålesund to Hamburg/Rotterdam, and further shipped overseas by a deep-sea container shipping line (e.g. APL). The process involves customs and food health declarations. The transport set-up is mostly fixed."

As is: The Port updates the website with information on the port's services, capacity, resources, and weather (in practice, port calls info updated systematically). This serves as information source for customers (ship agents, terminal operators) and all other stakeholders.

Challenges: Much manual info registration, and a lot of work duplication.

For improvement in the future, the port envisions the following improvements:

- A marketing portal, like a resource hub accessible from the website, enabling online management of bookings, resources and services as well as communication and coordination with third party service provider systems.
- Automatic update of Webpages (“ship calling”, “at port”, “departure”, etc.) based on information from SafeSeaNet and actual data from AIS.
- Online registration of booking directly by the ship / ship agent.

In order to make these improvements, FInest demands from FI-WARE the following EPIC that will be covered by the Application Mashup GE:

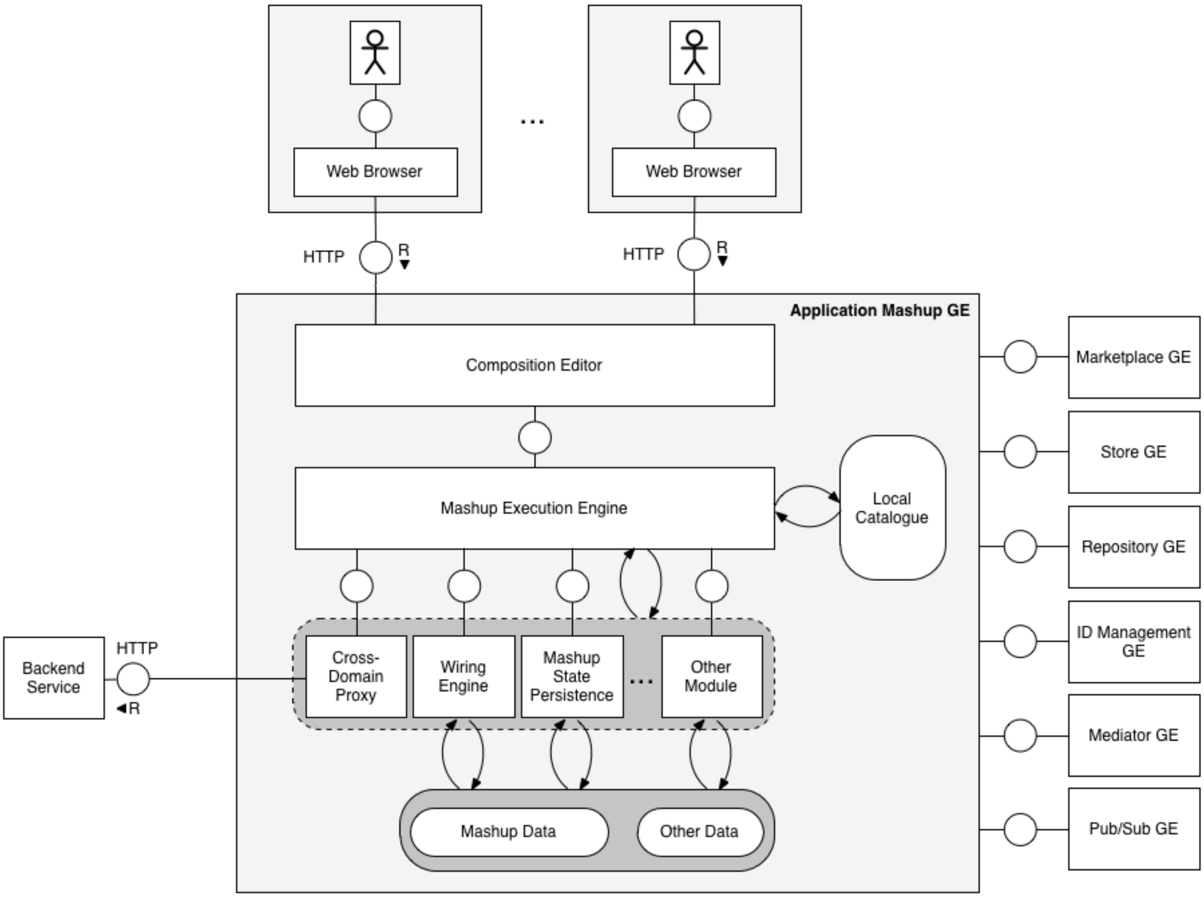
"*FInest.Epic.IoS.WidgetPlatformInfrastructure:* A visual portal website is needed where each user can add, remove and use widgets. Therefore, also a widget repository is also needed from which a user can select widgets from. An infrastructure should be provided to deploy new widgets in the portal. It should be easy for end users to use."

There follows a the description of how the Application Mashup GE can be used to help to deal with the envisioned improvements:

- The functionality and information sources are split into a set of widgets. There is one widget for each resource that will be made accessible from the website: management of bookings and registrations, management of resources, management of services. Widgets from a third party service provider capable of communicating and coordinating with their systems, event-driven widgets connected to SafeSeaNet and actual data from AIS, e.g. "ship calling", "at port", "departure", etc. are also added to the catalogue of available widgets.
- These widgets are shared and offered through a repository (or store, or marketplace) which the project stakeholders and customers (ship agents, terminal operators, etc.) can search to select and retrieve the offerings of their interest.
- Each customer and stakeholder involved in this scenario, regardless of their level of technical or programming skills can leverage the application mashup editor to visually build a customized cockpit with the most valuable data and operations for their work by adding, removing and using available widgets and mashuplets (off-the-shelf mashups that can be customized by adding and removing widgets to/from them). Moreover, they even can share the resulting application mashup for future use by other customers or stake holders (for further customization).
- A widget platform (or application mashup container) will serve as the envisioned visual portal website where these customers and stakeholders can easily deploy and use the widgets that make up the application mashup (i.e. the customized cockpit or information/operations dashboard that best fit their interests).

33.4.3 Architecture

This section describes the Application Mashup GE architecture. The diagrams use FMC (Fundamental Modelling Concepts) notation to facilitate the communication not only between technical experts but also between technical experts and business or domain experts. The Application Mashup GE provides the functionality necessary for developing and executing mashups. As the figure below shows, the core of the Application Mashup has three main components: the **Composition Editor**, the **Mashup Execution Engine**, and the **Local Catalogue**.



The Application Mashup GE Architecture

The **Composition Editor** component is the web-based tool with which end users interact via a web browser in order to create their own mashup applications. This component must, at least, offer end users a kind of *workspace* where they can spatially place or arrange widgets, plus an extra view of the wiring mechanism to set the interconnection between the arranged widgets. Because this component is a visual editor, this open specification does not set the visual appearance that this tool must have. It is up to the GE's implementation developer to create their own look & feel for this tool.

The **Mashup Execution Engine** component is probably the most important part to be developed. It coordinates widget execution and controls the data flow between widgets. It can access the Local Catalogue to deploy and execute stored widgets. The functionality of this component can be connected to and extended by a number of plug-in modules as shown in the Application Mashup GE Architecture figure. Module functionality is exposed to the widgets by means of the WidgetAPI (see Open API Specifications). Some of these plug-in modules must always be there:

- The **Cross-Domain Proxy** module: this component will provide widgets with a proxy to overcome the Javascript cross-domain problem.
- The **Wiring Engine** Module: this component manages the wiring mechanism.
- The **Mashup State Persistence** Module: this module is in charge of guaranteeing the persistence of the MACs under execution. This includes not only to storing the

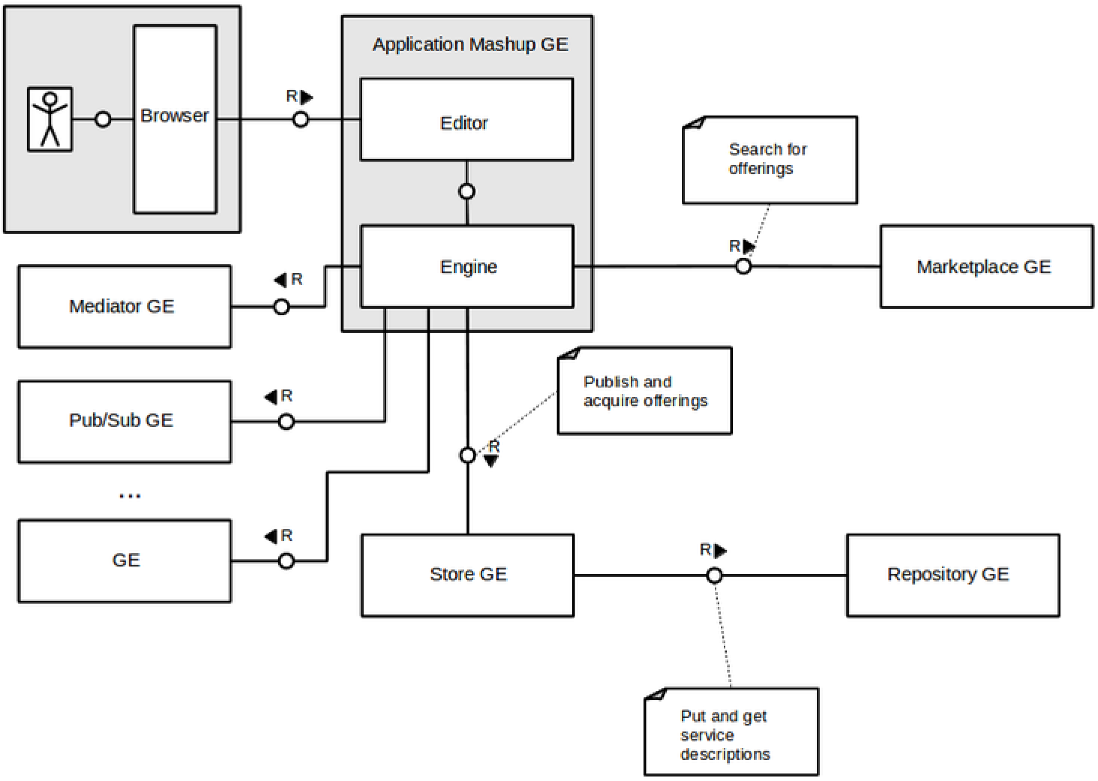
widgets and operators involved in the mashup and their state, but also their position in the editor view, their interconnections (wiring and piping) and so on.

It must be possible to enhance widget functionality by adding new modules to the Mashup Execution Engine. For example, a Publish/Subscribe module could be added to provide widgets with the ability to receive and publish data in a pub/sub fashion using the new added module.

All plug-in modules must be able to make use of internal storage (i.e. a database) for their specific persistence needs.

The third main component is the **Local Catalogue**. This component is where MACs, either purchased from the FI-WARE Store GE or installed (uploaded) by the end-user, are stored, configured, and set ready for deployment and execution. This component should be a kind of showcase for the logged user of the Application Mashup GE.

The Application Mashup GE is closely related to other FI-WARE Generic Enablers, especially related to the business infrastructure and the provision of data sources. The figure below depicts some of these relationships:



How the Application Mashup GE relates to the other Generic Enablers

The following sections describe the languages needed to support widgets, operators and mashups, including the USDL extensions for all business-related GEs to process any MAC as an offering, and the specific file formats used to store and distribute widgets. By implementing these artefacts, a concrete implementation of the Application Mashup GE will support an internal representation of MACs (widgets and mashups), which is necessary to interact with them.

33.4.4 Mashup and Widgets Definition Languages (MDL and WDL)

The Application Mashup GE should be implemented as a mashup platform that empowers users to create their own application mashups. Application mashups are made of a set of widgets interconnected with each other (that is, wired). To fully support widgets (and mashups) instantiation, the implementation of the Application Mashup GE must support a platform-independent widget and mashup languages.

The *Mashup Definition Language* (MDL) and the *Widget Definition Language* (WDL) are the chosen languages. They define all the inner information (metadata) regarding both a mashup and their widgets and their relationships/interconnections. This includes typical metadata, such as widget's name, vendor, version, last updated date, but also graphical information such as the location of the widgets on the editor canvas, widget width and height, etc.

The following sections shows the concept of "template". The template represents the definition of both languages and must be supported by the Application Mashup GE.

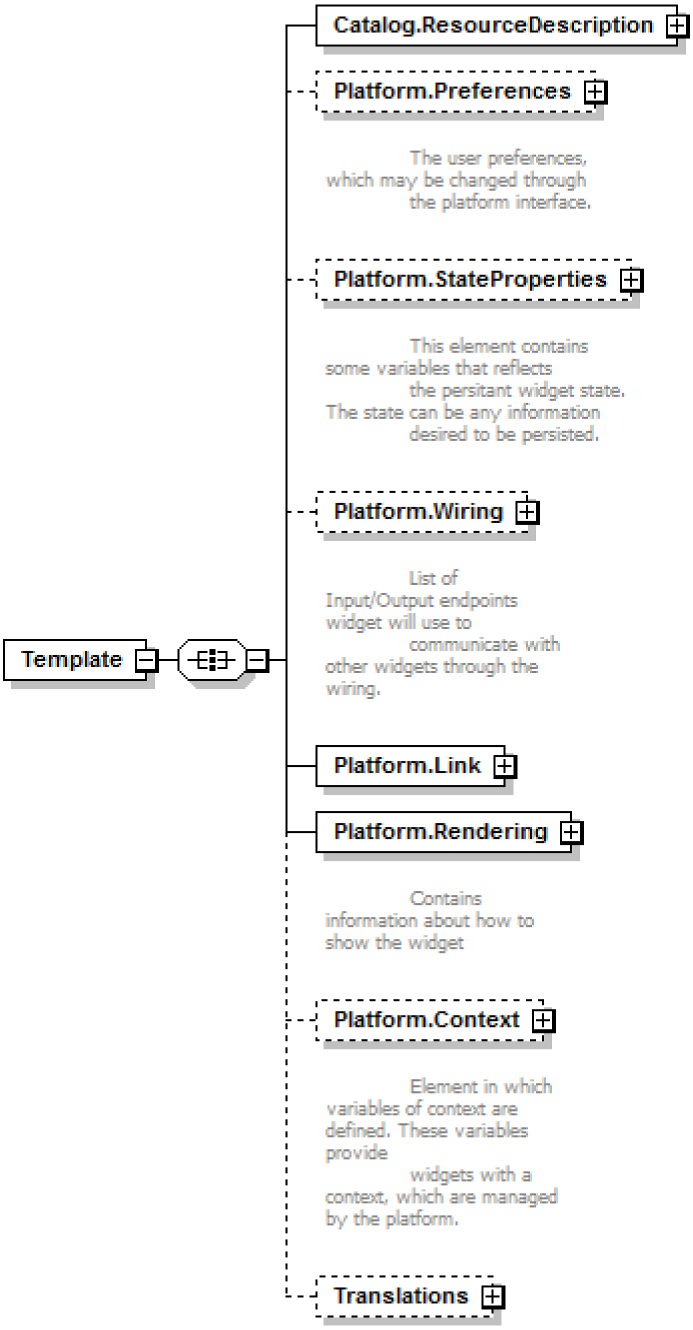
33.4.4.1 ***Mashup and Widget Template***

To internally represent and deal with both mashups and widgets, the Application Mashup GE must support their "templates". A template is an XML file that contains all mashup and widget-related contextual information needed by the Application Mashup plus a set of preferences, state properties, and wiring, context and rendering information. Both MDL and WDL templates have their associated XML Schema. The latest version of the XML Schemas described in the following sections are available at:

- MDL XML Template Schema (https://github.com/Wirecloud/wirecloud/blob/develop/docs/source/mashups/mashup_template_xml.xsd)
- WDL XML Template Schema (https://github.com/Wirecloud/wirecloud/blob/develop/docs/source/widgets/template/xml_template.xsd)

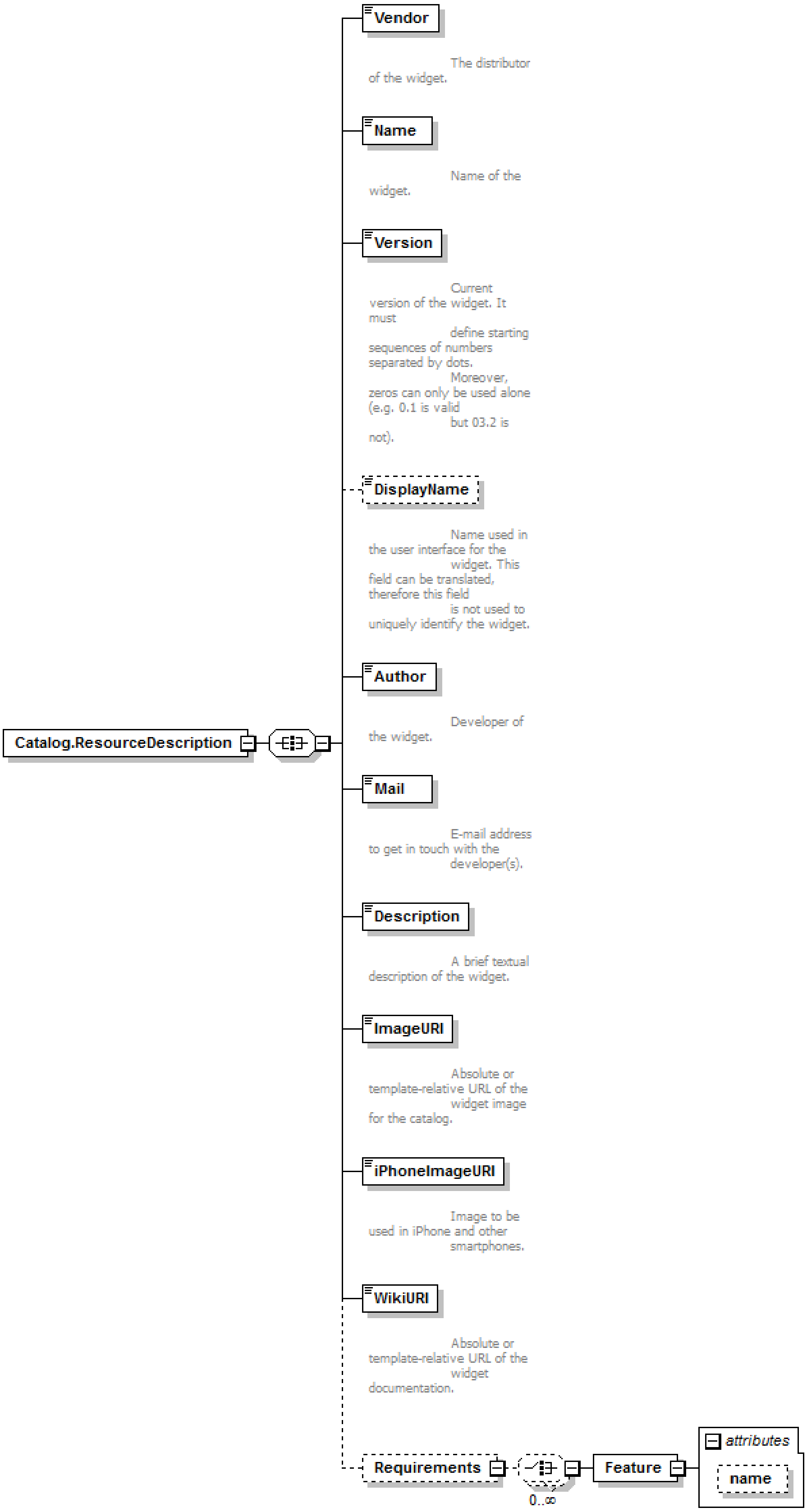
WDL Template description as a XML Schema Definition

First we describe a description of the widget template XML Schema, i.e. what WDL looks like. It uses the *http://morfeo-project.org/2007/Template* namespace for the root element, called **Template**. The figure below shows the `Template` element and the sequence of subelements that it contains.



The "Template" root element

The `Template` element defines all the widget-related contextual information in an XML element called **Catalog.ResourceDescription**. This is a mandatory element of the XML document. The figure below depicts what it looks like:



The "Catalog.ResourceDescription" element

This core element it is made up of the following attributes:

- **Vendor:** Company that distributes the widget. It cannot contain the character "/".
- **Name:** Name of the widget. It cannot contain the character "/".
- **Version:** Current widget version number. It must define starting sequences of numbers separated by dots. Zeros can only be used alone (e.g. 0.1 is valid but 03.2 is not).
- **DisplayName:** Name shown in the user interface of the widget. This field can be translated; therefore this field does not identify the widget.
- **Author:** Widget developers.
- **Mail:** Developer's e-mail address.
- **Description**: Full widget description to be shown in the catalogue.
- **ImageURI:** Absolute or template-relative URL of the image shown in the catalogue.
- **iPhoneImageURI:** Image to be used in iPhones and other smartphones.
- **WikiURI:** Absolute or template-relative URL of the widget documentation.

The *vendor*, *name* and *version* fields are the the widget's ID. Therefore, no such identifier can appear more than once in any collection of the Application Mashup GE stored resources (this includes widgets, mashups, operators, etc.).

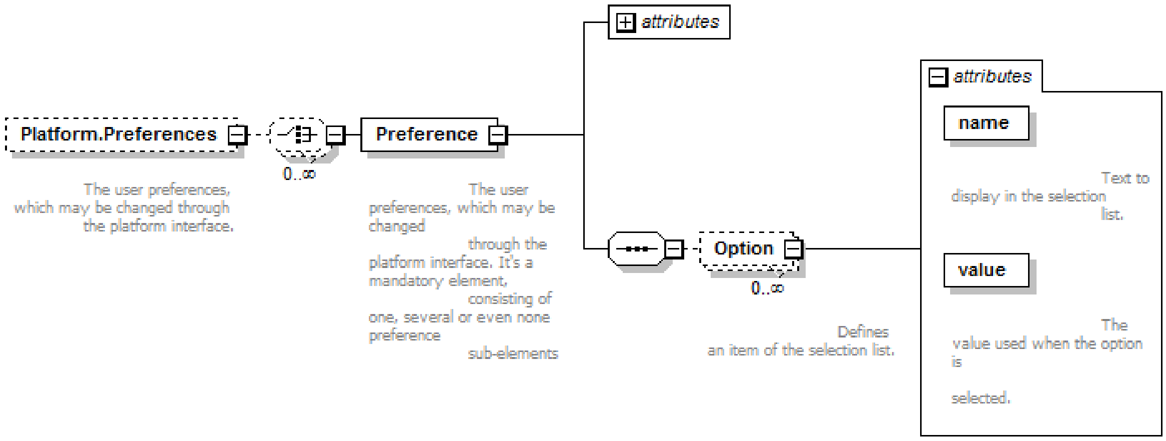
XML elements that manage the Platform-Widget interaction

To guarantee the platform-widget interaction, templates also define a set of variables that widgets use to get connected to the environment and set different platform options. Likewise, it also defines some other interface elements, such as the initial widget size. They are all managed by the platform, which will ensure their persistence.

Let us go through all these elements:

The Platform.Preferences element

The first platform-related element is the `Platform.Preferences` one:



The "Platform.Preferences" element

It defines user preferences, which may be changed through the platform interface. It is a mandatory element that is made up of one, many or none `Preference` sub-elements. This defines the actual user preference. It requires the following attributes:

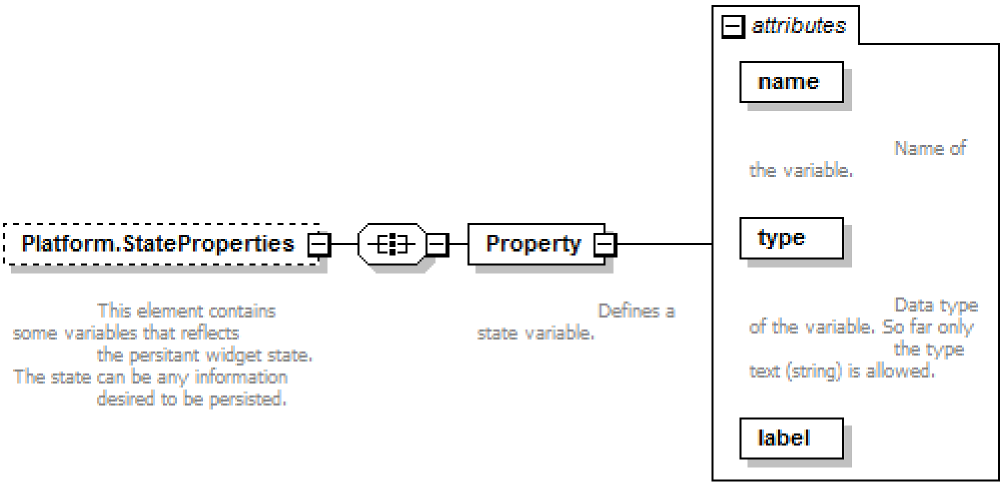
- **name:** name of the preference to be referenced in the source code.
- **type:** preference data type: text (string), number, Boolean, password and list.
- **description:** text that describes the preference.
- **label:** text that the preference will show in the user interface.
- **default:** preference default value.

If the *type* attribute is set to "list", the different choices are defined by means of the **Option** element. It has the following attributes:

- **name:** text to be displayed in the selection list.
- **value:** value to be used when the option is selected.

The Platform.StateProperties element

The next XML element is the **Platform.StateProperties** element. Its main purpose is to define a set of properties to store the state of the widget while it is executing, in order to have it available for future executions. Its structure is shown in the figure below:



The Platform.StateProperties element

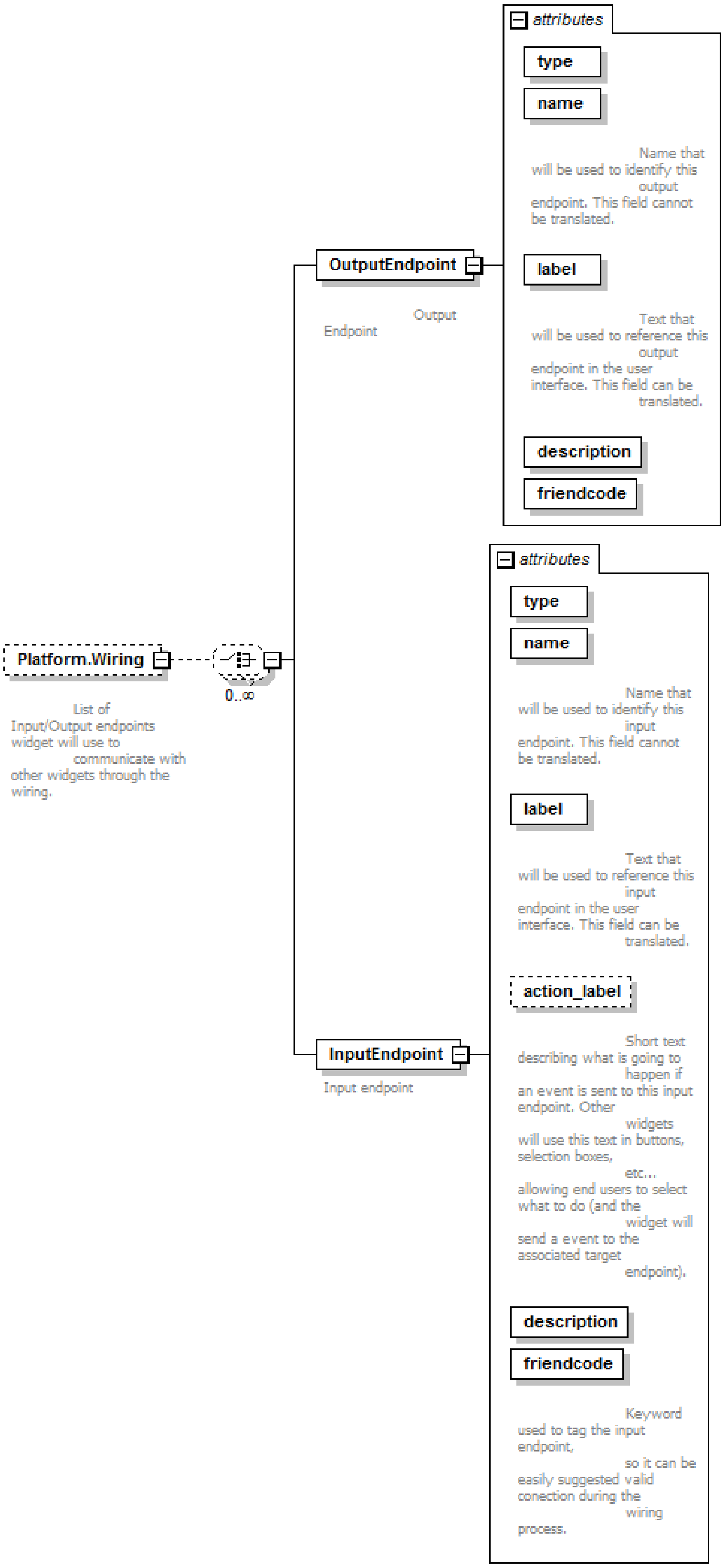
This element is required. It is made up of a list of **Property** elements and requires the following attributes:

- **name:** property name.
- **type:** property data type: only "text" (string) datatype does make sense in here.
- **label:** text to be displayed in the user interface.

The Platform.Wiring element

This is probably one of the most important widget template elements. It defines both the widget inputs and outputs needed to intercommunicate with other widgets. The Application Mashup GE implementation must take this information into account to manage and control the wiring mechanism and its internal data flow.

The figure below depicts the **Platform.Wiring** element:



The Platform.Wiring element.

This element may contain any number of **InputEndpoint** and **OutputEndpoint** elements:

Widgets may send data (events) through an output endpoint. To do so, they must declare the endpoint using the *`OutputEndpoint`* element. These elements have the following attributes:

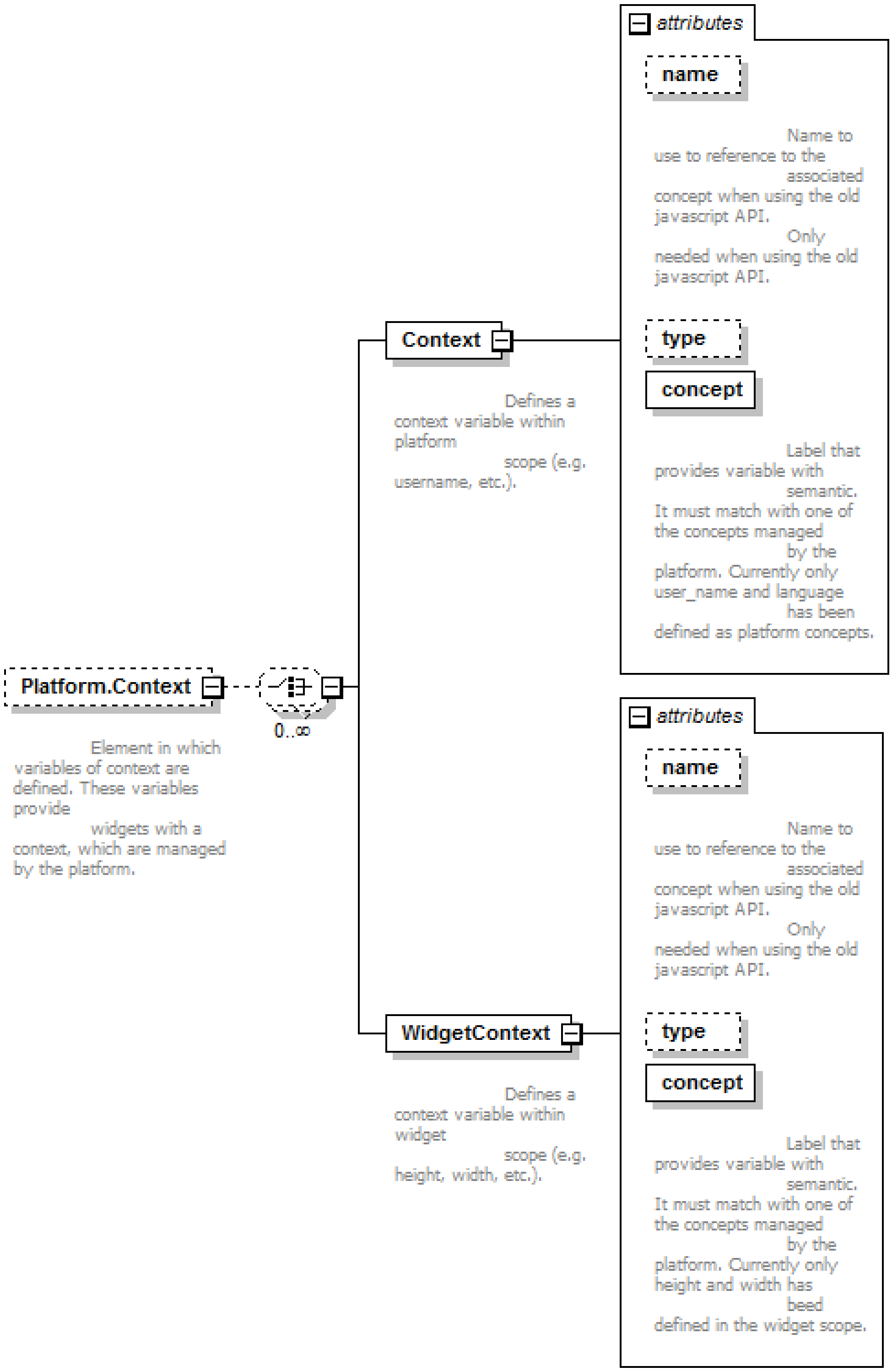
- **name:** output endpoint name.
- **type:** output endpoint data type: only "text" (string) datatype does make sense in here.
- **label:** text to be displayed in the user interface.
- **description:** text that describes the output.
- **friendcode:** keyword used as an output endpoint tag: it will help the platform to make suggestions in the wiring process.

On the other hand, widgets can receive asynchronous data through the input endpoints. These endpoints are meant to be used by the widget for receiving data (events) coming from other widgets. The required *`InputEndpoint`* elements requires the following attributes:

- **name:** input endpoint name.
- **type:** input endpoint data type: only "text" (string) datatype does make sense in here.
- **label:** text to be displayed in the user interface.
- **actionlabel:** short text that describes what is going to happen if an event is sent to this input endpoint. Widgets could use this text in buttons, selection boxes, etc... allowing end users to select what to do (and the widget will send an event to the associated target endpoint)
- **description:** text that describes the input.
- **friendcode:** keyword used as an input endpoint tag: it will help the platform to make suggestions in the wiring process.

The Platform.Context element

Widgets can have associated context information (i.e. usernames, current height and width...). The `Platform.Context` element defines which data the widget will be able to access and be notified if changed. The structure of this element is depicted in the figure below:



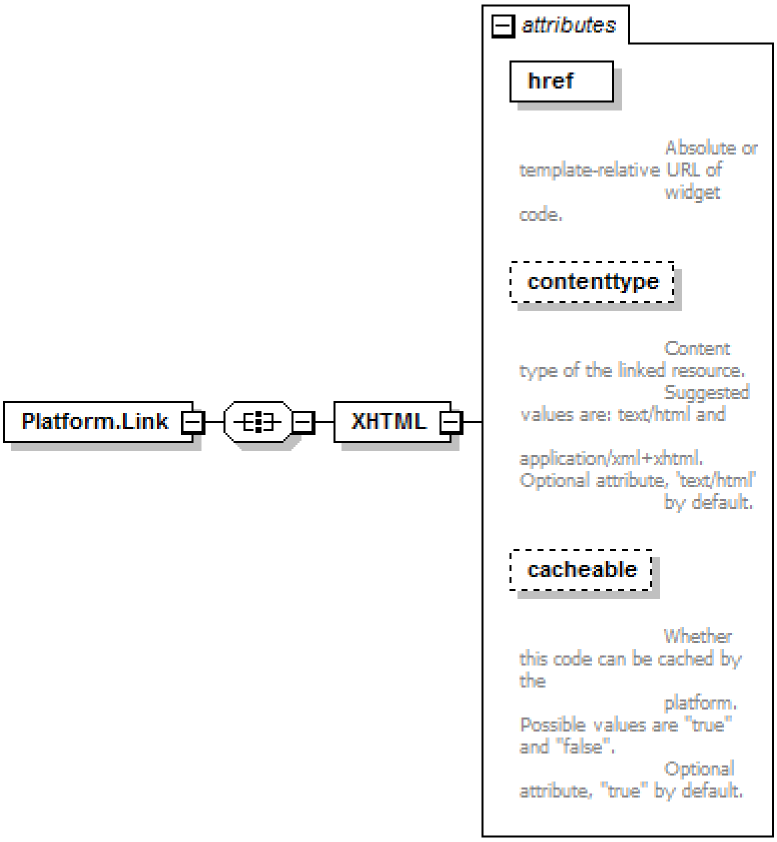
The Platform.Context element

This mandatory element can be followed by any number of these two child elements: **Context** and **WidgetContext**. The **Context** defines a platform-related context variable (i.e. username), whereas the **WidgetContext** defines a widget-related context variable (i.e. height). Both of them must have the following attributes:

- **name:** variable name.
- **type:** data type of the variable. Only "text" (string) datatype does make sense in here.
- **concept:** text that gives the variable meaning by annotating it with semantics. It must match with one of the concepts managed by the platform. Currently only *user_name* and *language* have been defined as platform concepts, and *height* and *width* in the widget scope.

The Platform.Link element

The actual source code of the widget must be linked to this template. To do this, the `Platform.Link` element is needed.



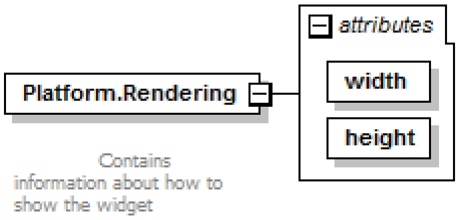
The Platform.Link element binds the template with the actual widget source code

It is made up of the **XHTML** element, which has the following attributes:

- **href:** absolute or template-relative URL of widget code.
- **contenttype:** linked resource content type: suggested values are: text/html and application/xml+xhtml. This is an optional attribute, with 'text/html' by default.
- **cacheable:** sets if the linked code can be cached by the platform: possible values are "true" and "false". This is an optional attribute, "true" by default.

The Platform.Rendering element

The last template XML element is `Platform.Rendering`. It specifies the default width and height of the widget once it is deployed in the user workspace.

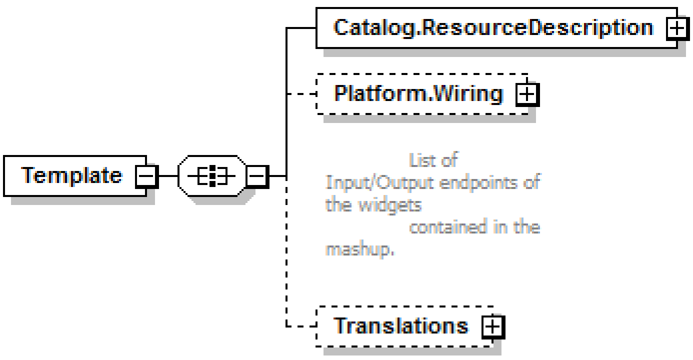


The Platform.Rendering element

Width and **height** are its only subelements. They represent the initial width and height of the widget.

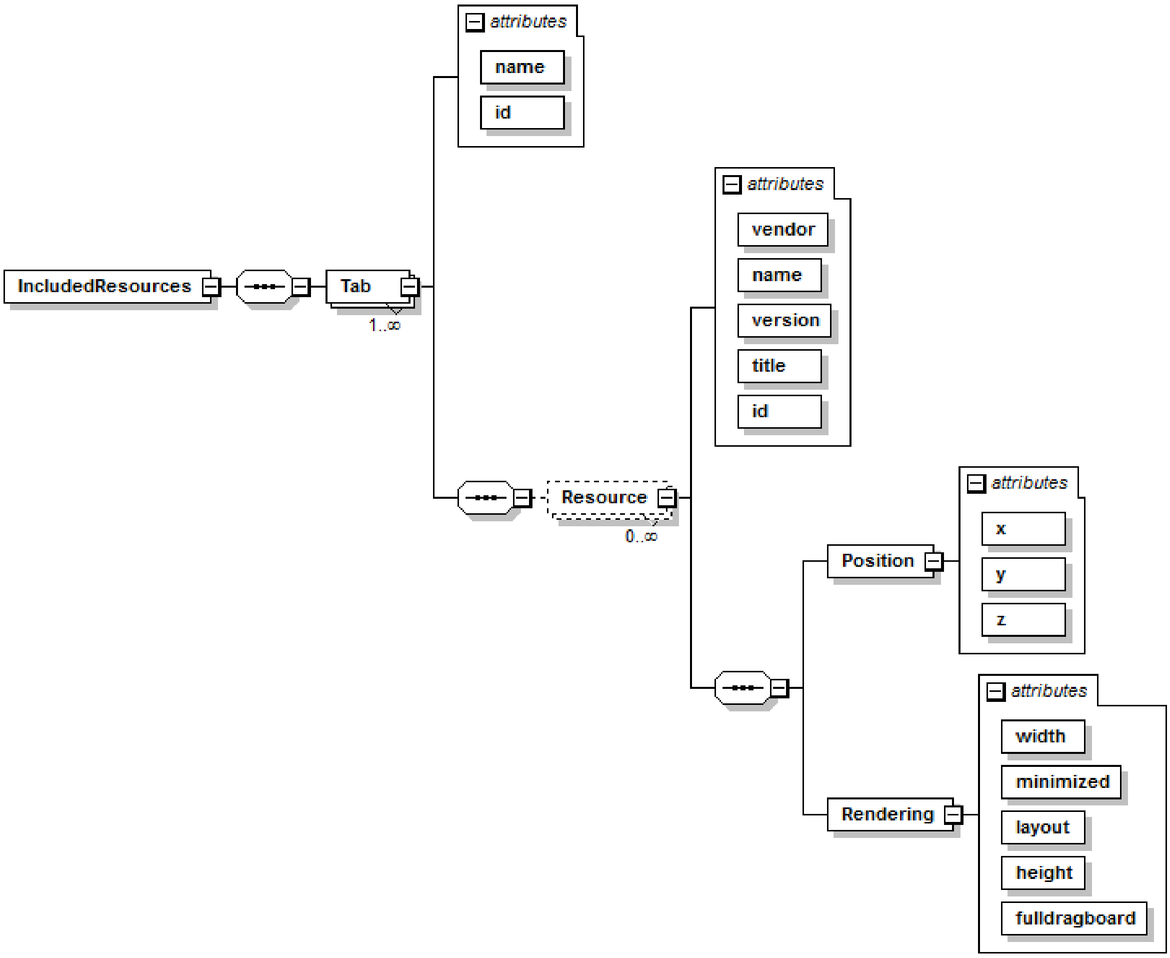
MDL Template description as a XML Schema Definition

The MDL XML template schema is quite similar to the WDL template and is used to describe a mashup composed of widgets. The figure below shows the Mashup `Template` element and its sequence of subelements.



The Mashup "Template" root element

The **Catalog.ResourceDescription** element has the same fields as in the widget template with an extra field called **IncludedResources** that is used to describe widgets within the mashup. The figure below depicts what it looks like:



The "IncludedResources" element

This element contains at least one **Tab** element that represents tabs in Application Mashup GE dashboard. It has the following attributes.

- **name:** the name of the tab
- **id:** the identification of the tab; this id is internal to the template.

The **Tab** element may contain any number of **Resource** elements which represent widget instances used in the mashup. It has the following attributes.

- **vendor** the widget distributor; it cannot contain the character "/".
- **name** name of the widget; it cannot contain the character "/".
- **version** current version of the widget; it must define starting sequences of numbers separated by dots, where zeros can only be used alone (e.g. 0.1 is valid but 03.2 is not).
- **title** name to be displayed in the widget dashboard.
- **id** the widget identification; this id is internal to the mashup template.

The **Resource** element is made up of a **Position** element and a **Rendering** element. The **Position** element describes the widget position into the dashboard. It has the following attributes.

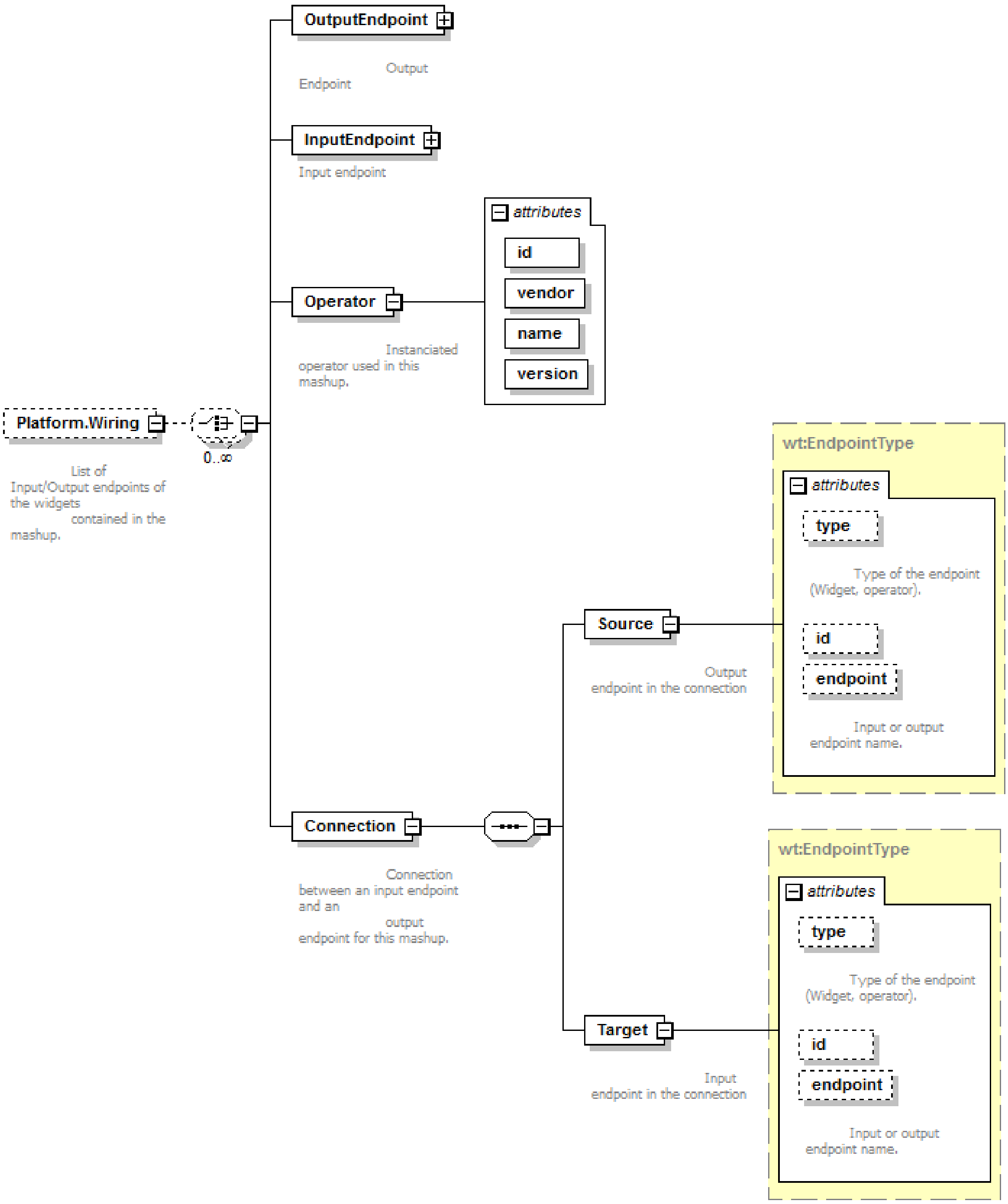
- **X:** the widget's x coordinate.
- **Y:** the widget's Y coordinate.
- **Z:** the widget's Z coordinate.

The **Rendering** element describes some characteristics of the widget representation. It has the following attributes.

- **width:** widget width in the dashboard.
- **minimized:** Boolean attribute that defines whether the widget is minimized in the dashboard
- **layout:** widget layout in the dashboard
- **height:** widget height in the dashboard
- **fulldragboard:** Boolean attribute that describes whether the widget is using all the dashboard.

The Platform.Wiring element

This element describes how widgets in the mashup are connected using their output and input endpoints.



The "Platform.Wiring" element

The **InputEndpoint** and **OutputEndpoint** elements define the same information as in the WDL template. The **Platform.Wiring** element contains all the input and output endpoints of all the widgets and operators in the mashup.
The **Platform.Wiring** element may contain any number of **Operator** elements. An **Operator** element defines an operator that is used in the wiring. It has the following attributes.

- **id:** identification of the operator; this id is internal to the mashup template.
- **vendor:** the distributor of the operator; it cannot contain the character "/".

- **name:** operator name; it cannot contain the character "/".
- **version:** current operator version; it must define starting sequences of numbers separated by dots where zeros can only be used alone (e.g. 0.1 is valid but 03.2 is not).

The **Platform.Wiring** element may contain any number of **Connection** elements. These elements describe which output endpoints are connected with which input endpoints. The **Connection** elements are composed of a **Source** element and a **Target** element. The **Source** element defines the output endpoint of the connection. It has the following attributes.

- **type:** type of the element that has the output endpoint; this attribute could has the values "widget" or "operator".
- **id:** id of the element that has the output endpoint; this id is the same as the id defined in the **Resource** element if the element is a widget, whereas this id is the same as the id defined in the **Operator** element if the element is an operator.
- **endpoint:** the name of the output endpoint. This name is the same as the defined in the **OutputEnpoint** element.

The **Target** element defines the input endpoint of the connection. It has the following attributes.

- **type:** type of element that has the input endpoint; the possible values of this attribute are "widget" or "operator".
- **id:** id of the element that has the input endpoint; this id is the same as the id defined in the **Resource** element if the element is a widget, whereas this id is the same as the id defined in the **Operator** element if the element is an operator.
- **endpoint** the name of the input endpoint; this name is the same as the defined in the **InputEnpoint** element.

33.4.4.2 *Mashup and Widgets as a USDL offering: USDL extension*

In order for both mashups and widgets to be offered as a USDL offering in the FI-WARE Store GE so that both widgets and mashups can be part of the data managed by the FI-WARE's Marketplace, Store and Repository Generic Enablers, the Application Mashup GE must make use of the following RDF(S) vocabularies, built upon Linked Data principles as Linked USDL extensions.

The first specification, WDL-RDF:

(https://github.com/Wirecloud/wirecloud/blob/develop/docs/source/widgets/template/rdf_template.rdf)

deals with the definition of the information that the Application Mashup GE must use to instantiate a widget, including its user preferences, state properties, wiring information, and so on.

The second specification, MDL-RDF:

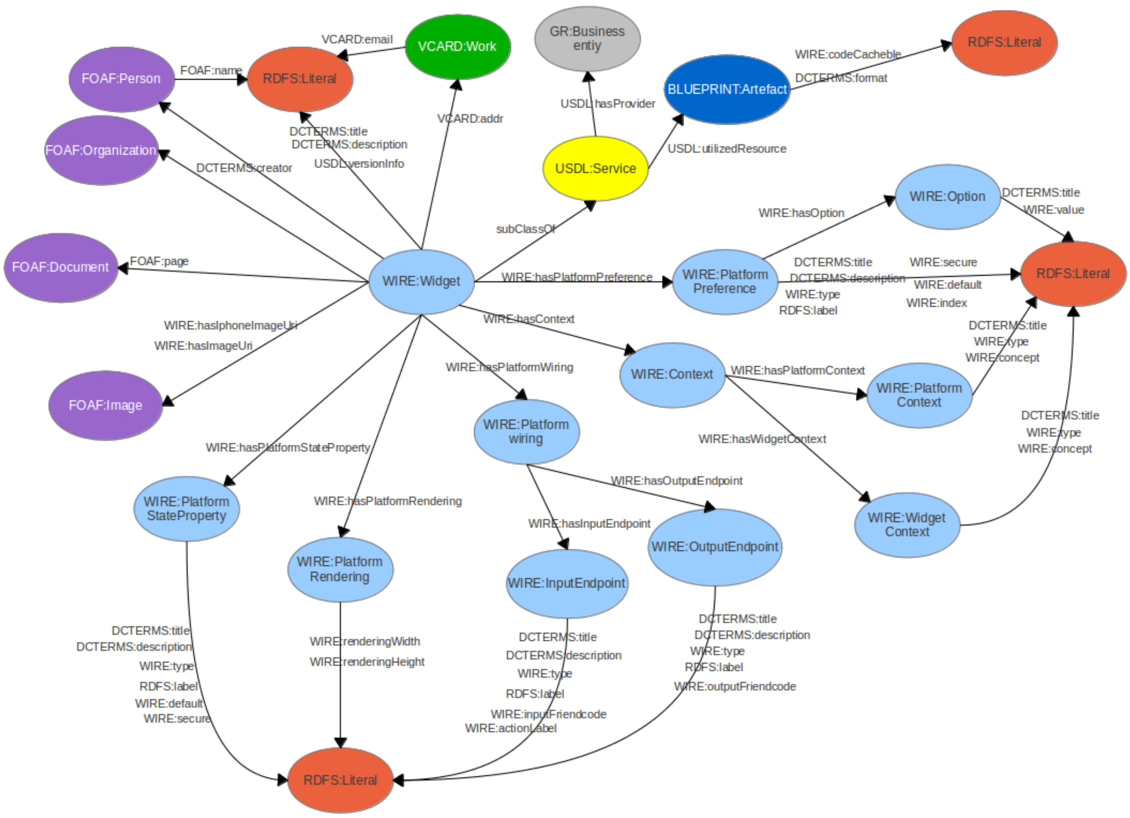
(https://github.com/Wirecloud/wirecloud/blob/develop/docs/source/mashups/mashup_template_rdf.rdf)

defines the mashup-related information needed to create an instance of the user workspace, including platform-specific information such as widget instances, wiring and piping between widgets, etc.

The following sections show both vocabularies to be used within the USDL offerings.

WDL-RDF

The diagram below shows the WDL-RDF vocabulary.



The WDL-RDF extension for USDL

The Application Mashup GE must support this vocabulary to provide a way to represent WDL information as part of a USDL offering.

Classes

Class: `wire:Widget`

This class represents a widget. This is the main class of the vocabulary.

URI: http://wirecloud.conwet.fi.upm.es/ns/widget#Widget

Properties include:

`dcterms:title, dcterms:description, dcterms:creator, usdl:hasProvider, usdl:utilizedResource, foaf:page, wire:hasPlatformPreference, wire:hasContext, wire:hasPlatformWiring, wire:hasPlatformRendering, wire:hasPlatformStateProperty, usdl:versionInfo, wire:hasImageUri, wire.hasiPhoneImageUri, wire:displayName, vcard:addr`

Subclassof: `usdl-core:Service`

Class: `wire:Operator`

This class represents an operator.

URI: http://wirecloud.conwet.fi.upm.es/ns/widget#Operator

Properties include:

`dcterms:title, dcterms:description, dcterms:creator, usdl:hasProvider, usdl:utilizedResource, foaf: page, wire:hasPlatformPreference, wire:hasContext, wire:hasPlatformWiring, wire:hasPlatformRendering, wire:hasPlatformStateProperty, usdl:versionInfo, wire:hasImageUri, wire.hasiPhoneImageUri, wire:displayName, vcard:addr`

Subclassof: `usdl-core:Service`

Class: `wire:PlatformPreference`

This class represents a user preference in the Application Mashup GE, that is, data users can see and configure. The Enabler must make this value persistent and provide users with tools to edit and validate this data.

URI: http://wirecloud.conwet.fi.upm.es/ns/widget#PlatformPreference

Properties include:

`wire:hasOption, dcterms:title, dcterms:description, rdfs:label, wire:type, wire:default, wire:secure`

Used with: `wire:hasPlatformPreference`

Class: `wire:Context`

This class represents the context of the widget.

URI: http://wirecloud.conwet.fi.upm.es/ns/widget#Context

Properties include:

`wire:hasPlatformContext, wire:haswidgetContext`

Used with: `wire:hasContext`

Class: `wire:PlatformWiring`

This class represents the wiring status of a widget.

URI: http://wirecloud.conwet.fi.upm.es/ns/widget#PlatformWiring

Properties include:

`wire:hasOutputEndpoint, wire:hasInputEnpoint`

Used with: `wire:hasPlatformWiring`

Class: `wire:PlatformRendering`

This class represents the widget size when it is instantiated.

URI: http://wirecloud.conwet.fi.upm.es/ns/widget#PlatformRendering

Properties include:

`wire:renderingWidth, wire.renderingHeight`

Used with: `wire:hasPlatformRendering`

Class: wire:PlatformStateProperty

This class represents a widget state variable that the platform needs to know in order to make it persistent.

URI: http://wirecloud.conwet.fi.upm.es/ns/widget#PlatformStateProperty

Properties include:

`dcterms:title, dcterms:description, wire:type, rdfs:label, wire:default, wire:secure`

Used with: `wire:hasPlatformStateProperty`

Class: `wire:Option`

This class represents an option that a user preference could have.

URI: http://wirecloud.conwet.fi.upm.es/ns/widget#Option

Properties include:

`dcterms:title, wire:value`

Used with: `wire:hasOption`

Class: `wire:PlatformContext`

This class represents a platform context variable.

URI: http://wirecloud.conwet.fi.upm.es/ns/widget#PlatformContext

Properties include:

`dcterms:title, wire:type, wire:concept`

Used with: `wire:hasPlatformContext`

Class: `wire:widgetContext`

This class represents a widget context variable.

URI: http://wirecloud.conwet.fi.upm.es/ns/widget#widgetContext

Properties include:

`dcterms:title, wire:type, wire:concept`

Used with: `wire:haswidgetContext`

Class: `wire:OutputEndpoint`

This class represents an output endpoint.

URI: http://wirecloud.conwet.fi.upm.es/ns/widget#OutputEndpoint

Properties include:

`dcterms:title, dcterms:description, rdfs:label, wire:type, wire:outputFriendcode`

Used with: `wire:hasOutputEndpoint`

Class: `wire:InputEndpoint`

This class represents an input endpoint.

URI: http://wirecloud.conwet.fi.upm.es/ns/widget#InputEndpoint

Properties include:

`dcterms:title, dcterms:description, rdfs:label, wire:type, wire:inputFriendcode, wire:actionLabel`

Used with: `wire:hasInputEndpoint`

Properties

Property: `wire:hasPlatformPreference`

This property states a user widget preference.

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#hasPlatformPreference

Domain: `wire:Widget`

Range: `wire:PlatformPreference`

Property: `wire:hasContext`

This property states the widget context.

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#hasContext

Domain: `wire:Widget`

Range: `wire:Context`

Property: `wire:hasPlatformWiring`

This property states the widget wiring status.

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#hasPlatformWiring

Domain: `wire:Widget`

Range: `wire:PlatformWiring`

Property: `wire:hasPlatformRendering`

This property states how the widget must be rendered.

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#hasPlatformRendering

Domain: `wire:Widget`

Range: `wire:PlatformRendering`

Property: `wire:hasPlatformStateProperty`

This property states a widget state variable.

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#hasPlatformStateProperty

Domain: `wire:Widget`

Range: `wire:PlatformStateProperty`

Property: `wire:hasOption`

This property states a user preference option.

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#hasOption

Domain: `wire:PlatformPreference`

Range: `wire:Option`

Property: `wire:hasPlatformContext`

This property states a platform-context variable of the context.

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#hasPlatformContext

Domain: `wire:Context`

Range: `wire:PlatformContext`

Property: `wire:hasWidgetContext`

This property states a widget-context variable of the context.

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#hasWidgetContext

Domain: `wire:Context`

Range: `wire:WidgetContext`

Property: `wire:hasOutputEndpoint`

This property states a widget wiring output endpoint.

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#hasOutputEndpoint

Domain: `wire:PlatformWiring`

Range: `wire:OutputEndpoint`

Property: `wire:hasInputEndpoint`

This property states a widget wiring input endpoint.

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#hasInputEndpoint

Domain: `wire:PlatformWiring`

Range: `wire:InputEndpoint`

Property: `wire:platformContextConcept`

This property states the platform-context variable concept.

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#platformContextConcept

Domain: `wire:PlatformContext`

Range: `rdfs:Literal`

Property: `wire:WidgetContextConcept`

This property states the widget-context variable concept.

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#platformWidgetConcept

Domain: `wire:WidgetContext`

Range: `rdfs:Literal`

Property: `wire:outputFriendcode`

This property states an output's *friendcode*.

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#outputFriendcode

Domain: `wire:OutputEndpoint`

Range: `rdfs:Literal`

Property: `wire:inputFriendcode`

This property states an input's *friendcode*.

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#inputFriendcode

Domain: `wire:InputEndpoint`

Range: `rdfs:Literal`

Property: `wire:actionLabel`

This property states an input's action label.

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#actionLabel

Domain: `wire:InputEndpoint`

Range: `rdfs:Literal`

Property: `wire:hasImageUri`

This property states the URI of the image associated with the widget.

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#hasImageUri

Domain: `wire:Widget`

Range: `foaf:Image`

Property: `wire:hasiPhoneImageUri`

This property states the URI of the image associated with the Widget if the platform is running on an iPhone.

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#hasiPhoneImageUri

Domain: `wire:Widget`

Range: `foaf:Image`

Property: `wire:displayName`

This property states the widget name to be displayed.

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#displayName

Domain: `wire:Widget`

Range: `rdfs:Literal`

Property: `wire:value`

This property states the widget configuration element value.

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#value

Range: `rdfs:Literal`

Property: `wire:type`

This property states the widget configuration element type.

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#type

Range: `rdfs:Literal`

Property: `wire:default`

This property states the widget configuration element default value.

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#default

Range: `rdfs:Literal`

Property: `wire:secure`

This property states whether or not a widget configuration element is secure.

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#value

Range: `rdfs:Literal`

Property: `wire:index`

This property states the logical order of elements of the same type.

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#value

Range: `rdfs:Literal`

Property: `wire:codeContentType`

This property states the widget code MIME type. The widget code URI is represented using `usdl-core:Resource`

URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#codeContentType

Domain: `usdl-core:Resource`

Range: `rdfs:Literal`

Property: `wire:codeCacheable`

This property states whether or not the widget code is cacheable.

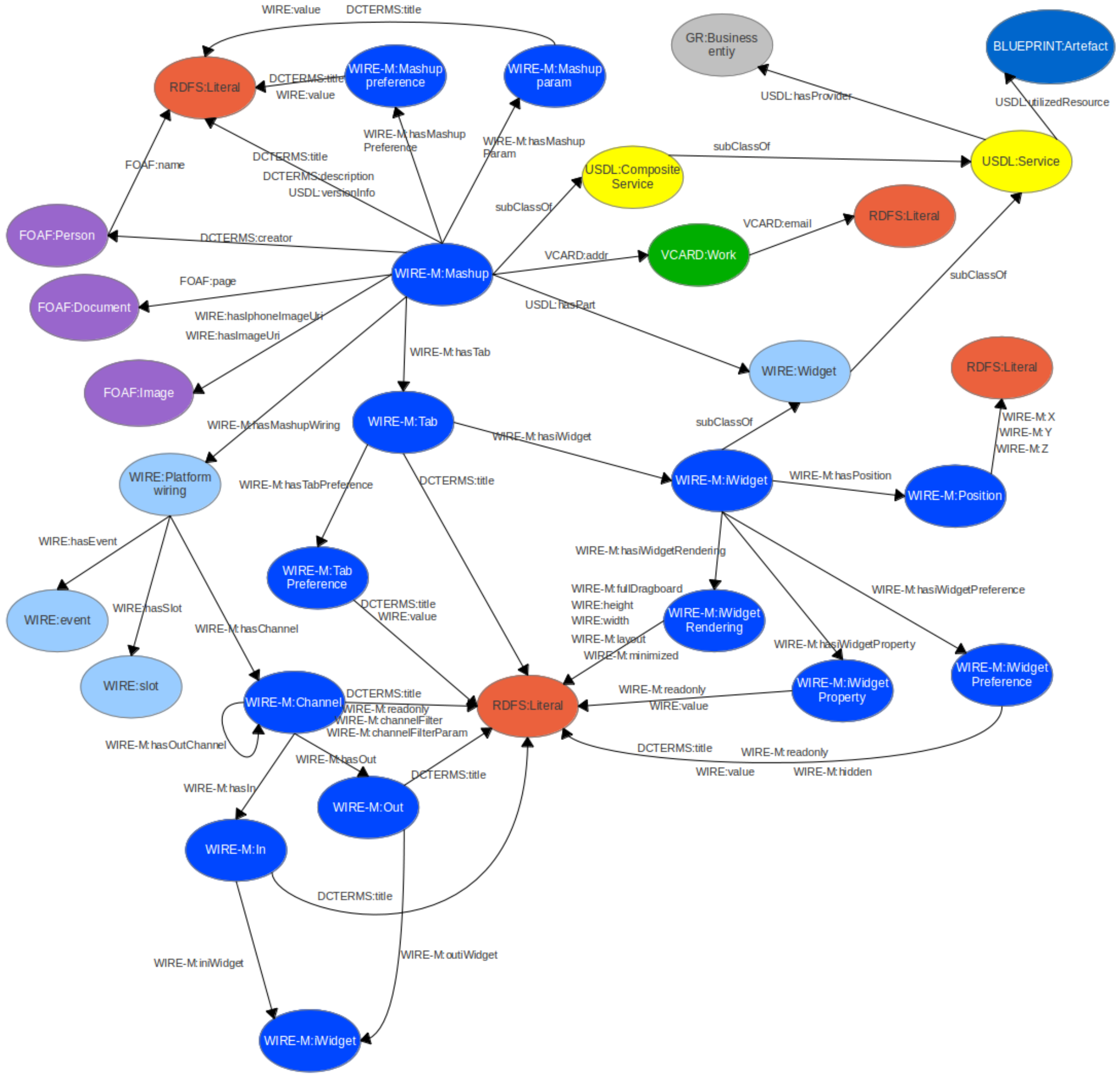
URI: http://wirecloud.conwet.fi.upm.es/ns/Widget#codeCacheable

Domain: `usdl-core:Resource`

Range: `rdfs:Literal`

MDL-RDF

The diagram below shows the MDL-RDF vocabulary.



The MDL-RDF extension for USDL

Like WDL-RDF, this vocabulary must be supported by the Application Mashup GE to provide a way to represent MDL information as part of a USDL offering.

Classes

Class: `wire-m:Mashup`

This class represents a mashup. It will be implemented as a workspace.

URI :http://wirecloud.conwet.fi.upm.es/ns/mashup#Mashup

Properties include:

`wire-m:hasMashupPreference, wire-m:hasMashupParam, wire-m:hasTab, wire-m:hasMashupWiring, wire:hasImageUri, wire:hasiPhoneImageUri, wire:version`

subClassOf: `usdl:CompositeService`

Class: `wire-m:Tab`

This class represents a workspace tab.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#Tab

Properties include:

`wire-m:hasiWidget, wire-m:hasTabPreference, dcterms:title`

Used with:

`wire-m:hasTab`

Class: `wire-m:iWidget`

This class represents a widget instance.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#iWidget

Properties include:

`wire-m:hasPosition, wire-m:hasiWidgetRendering, wire-m:hasiWidgetPreference, wire-m:hasiWidgetProperty`

Used with:

`wire-m:hasiWidget`

subClassOf: `wire:Widget`

Class: `wire-m:MashupPreference`

This class represents a mashup preference.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#MashupPreference

Properties include:

`dcterms:title, wire:value`

Used with:

`wire-m:hasMashupPreference`

Class: `wire-m:MashupParam`

This class represents a mashup parameter.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#MashupParam

Properties include:

`dcterms:title, wire:value`

Used with:

`wire-m:hasMashupParam`

Class: `wire-m:Position`

This class represents the position of a widget instance in the tab.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#Position

Properties include:

`wire-m:x, wire-m:y, wire-m:z`

Used with:

`wire-m:hasPosition`

Class: `wire-m:iWidgetPreference`

This class represents a widget instance preference.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#iWidgetPreference

Properties include:

`dcterms:title, wire:value, wire-m:readonly, wire-m:hidden`

Used with:

`wire-m:hasiWidgetPreference`

Class: `wire-m:iWidgetRendering`

This class represents a widget instance rendering.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#iWidgetRendering

Properties include:

`wire-m:fullDragboard, wire-m:layout, wire-m:minimized, wire:renderingHeight, wire:renderingWidth`

Used with:

`wire-m:hasiWidgetRendering`

Class: `wire-m:iWidgetProperty`

This class represents a widget instance property.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#iWidgetProperty

Properties include:

`wire-m:readonly, wire:value`

Used with:

`wire-m:hasiWidgetProperty`

Class: `wire-m:TabPreference`

This class represents a tab preference.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#TabPreference

Properties include:

`dcterms:title, wire:value`

Used with:

`wire-m:hasTabPreference`

Class: `wire-m:Connection`

This class represents a wiring connection between two widget instances or operator instances.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#Connection

Properties include:

`wire-m:hasSource, wire-m:hasTarget, dcterms:title, wire-m:readonly`

Used with:

`wire-m:hasConnection`

Class: `wire-m:Source`

This class represents a widget instance or operator instance that is the source of a connection.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#Source

Properties include:

`wire-m:sourceId, wire-m:endpoint, wire:type`

Used with:

`wire-m:hasSource`

Class: `wire-m:Target`

This class represents a widget instance or operator instance that is the target of a connection.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#Target

Properties include:

`wire-m:targetId, wire-m:endpoint, wire:type`

Used with:

`wire-m:hasTarget`

Class: `wire-m:iOperator`

This class represents an operator instance.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#iOperator

Properties include:

`wire-m:iOperatorId, dcterms:title`

Used with:

`wire-m:hasiOperator`

Properties

Property: `wire-m:hasMashupPreference`

This property states a mashup preference.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#hasMashupPreference

Domain: `wire-m:Mashup`

Range: `wire-m:MashupPreference`

Property: `wire-m:hasMashupParam`

This property states a mashup parameter.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#hasMashupParam

Domain: `wire-m:Mashup`

Range: `wire-m:MashupParam`

Property: `wire-m:hasTab`

This property states that a given tab is part of a workspace.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#hasTab

Domain: `wire-m:Mashup`

Range: `wire-m:Tab`

Property: `wire-m:hasiWidget`

This property states that a given widget instance is instantiated in a tab.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#hasiWidget

Domain: `wire-m:Tab`

Range: `wire-m:iWidget`

Property: `wire-m:hasTabPreference`

This property states a tab preference.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#hasTabPreference

Domain: `wire-m:Tab`

Range: `wire-m:TabPreference`

Property: `wire-m:hasPosition`

This property states the position of an widget instance in a tab.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#hasPosition

Domain: `wire-m:iWidget`

Range: `wire-m:Position`

Property: `wire-m:hasiWidgetPreference`

This property states a widget instance preference.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#hasiWidgetPreference

Domain: `wire-m:iWidget`

Range: `wire-m:iWidgetPreference`

Property: `wire-m:hasiWidgetProperty`

This property states a widget instance property.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#hasiWidgetProperty

Domain: `wire-m:iWidget`

Range: `wire-m:iWidgetProperty`

Property: `wire-m:hasiWidgetRendering`

This property states the rendering of a widget instance.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#hasiWidgetRendering

Domain: `wire-m:iWidget`

Range: `wire-m:iWidgetRendering`

Property: `wire-m:hasConnection`

This property states a wiring connection.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#hasConnection

Domain: `wire:PlatformWiring`

Range: `wire-m:Connection`

Property: `wire-m:hasSource`

This property states the source of a connection.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#hasSource

Domain: `wire-m:Connection`

Range: `wire-m:Source`

Property: `wire-m:hasTarget`

This property states the target of a connection.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#hasTarget

Domain: `wire-m:Connection`

Range: `wire-m:Target`

Property: `wire-m:targetId`

This property states the ID of a target.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#targetId

Domain: `wire-m:Target`

Range: `rdfs:Literal`

Property: `wire-m:sourceId`

This property states the ID of a source.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#sourceId

Domain: `wire-m:Source`

Range: `rdfs:Literal`

Property: `wire-m:endpoint`

This property states the ID of the widget instance or operator instance that is the source or target of a connection.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#endpoint

Range: `rdfs:Literal`

Property: `wire-m:hasiOperator`

This property states the wiring of an operator's instance.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#hasiOperator

Domain: `wire:PlatformWiring`

Range: `wire-m:iOperator`

Property: `wire-m:x`

This property states the x coordinate of a widget instance position.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#x

Domain: `wire-m:Position`

Range: `rdfs:Literal`

Property: `wire-m:y`

This property states the y coordinate of a widget instance position.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#y

Domain: `wire-m:Position`

Range: `rdfs:Literal`

Property: `wire-m:z`

This property states the z coordinate of a widget instance position.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#z

Domain: `wire-m:Position`

Range: `rdfs:Literal`

Property: `wire-m:fullDragboard`

This property states whether a widget instance occupies the whole space in the tab.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#fullDragboard

Domain: `wire-m:iWidgetRendering`

Range: `rdfs:Literal`

Property: `wire-m:layout`

This property states the layout of a widget instance.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#layout

Domain: `wire-m:iWidgetRendering`

Range: `rdfs:Literal`

Property: `wire-m:minimized`

This property states whether a widget instance is minimized in its tab.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#minimized

Domain: `wire-m:iWidgetRendering`

Range: `rdfs:Literal`

Property: `wire-m:hidden`

This property states whether a widget instance is hidden in its tab.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#hidden

Domain: `wire-m:iWidgetPreference`

Range: `rdfs:Literal`

Property: `wire-m:readonly`

This property states whether a mashup configuration element is read only.

URI: http://wirecloud.conwet.fi.upm.es/ns/mashup#readonly

Range: `rdfs:Literal`

33.4.4.3 *WGT zipped file format*

The Mashup Application GE relies on PKWare's Zip specification as the archive format for the self-packaged version of the Mashable Application Components. The packaging format acts as a container for files used by a MAC whereas the only initial requirement is to have a configuration document declaring metadata and configuration parameters for the MAC. This configuration file must use one of the metadata description languages supported by the Mashup Application GE (WDL and MDL in either of its flavours: XML and RDF). This configuration file must be present at the root of the Zip container and the name must be `config.xml` or `config.rdf`. Any relative path/URL included in the configuration document will use the root of the zip file as the base path/URL.

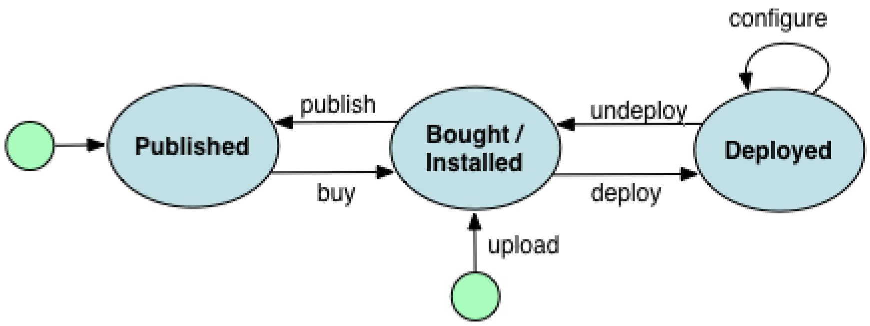
The Mashup Application GE should prohibit relative paths for accessing files outside the container. This is especially important as the Mashup Application GE may extract these files to the file system.

33.5 Main Interactions

This section describes in detail all the interactions that the Application Mashup GE must support both with users and with other FI-WARE GEs.

33.5.1 Life-cycle of a Mashable Application Component (mashup, widget and operators)

Note that Web mashups are aimed at leveraging the "long tail" of the Internet of Services by exploiting rapid development, the "Do-It-Yourself" (DIY) metaphor, and shareability. They typically serve a specific situational (i.e. immediate, short-lived, customized, specific) need, often with a high reuse potential. This need for sharing means that the Application Mashup GE should be fully compliant with the FI-WARE's Marketplace, Store and Repository Generic Enablers. The fact that a MAC can be offered in a Store before being used in the Application Mashup GE results in the definition of the following MAC life-cycle:



Lifecycle of a MAC (mashup, widget and operator)

Mashable Application Components (i.e, mashups, widgets and operators) must pass through the following states:

- **Published**
- **Bought/installed**
- **Deployed**

The init state for a MAC means that the MAC is neither published in the store, bought, nor installed in the user local repository. A MAC is **published** when it is made available to Store customers. Users that are interested in using a **published** MAC, can buy the MAC, thus transferring it to a *bought* state. Once *bought*, the MAC is automatically **installed** in the local catalogue of the Application Mashup GE. An alternative is to *upload* a MAC that the users have developed and which tehy do not have to buy. This is why the state is named **bought/installed**. Once a users have *uploaded* the MAC to the local catalogue, they can proceed to *publish*the MAC. Once the MAC is installed, it can be **deployed** in the user workspace. *Bought* and *installed* MACs must be **deployed** in the user workspace before they can be *configured*.

33.5.2 Interaction diagrams

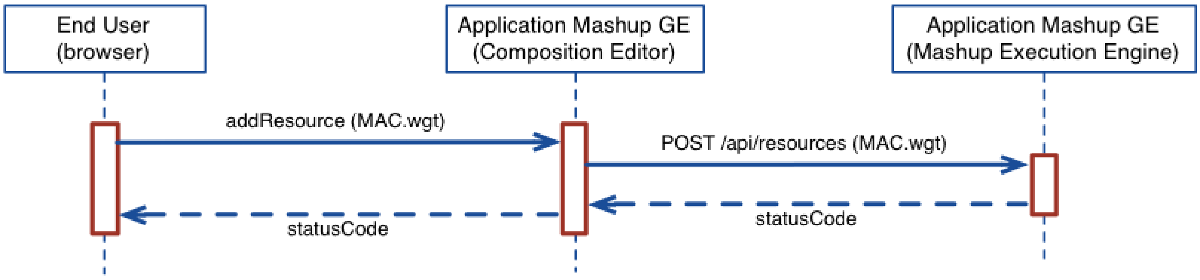
The Application Mashup Generic Enabler must be designed and developed to enable, at least, the interactions shown in this section. They cover the user-platform interactions needed to visually create a web application mashup, plus the main interactions between the platform and other generic enablers.

33.5.2.1 *User-Platform Interactions*

The interactions that the Application Mashup GE must support with regard to its users are as follows.

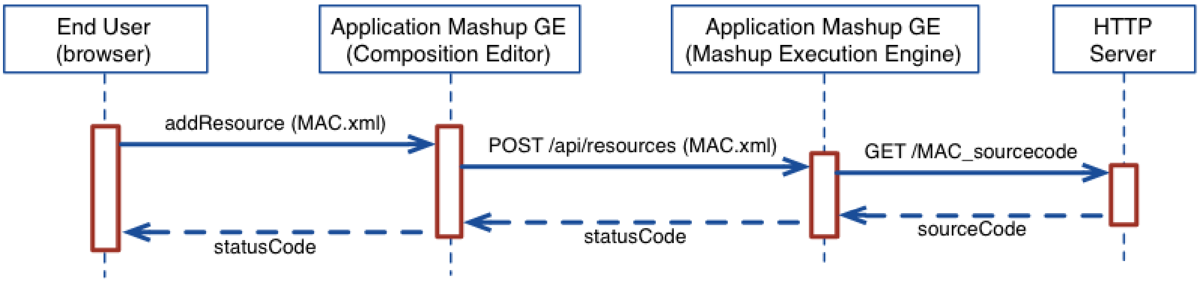
Upload a Mashable Application Component to the Local Catalogue

MAC developers must be able to upload their own developed resources to the local catalogue of the Application Mashup GE. End users must also be able to upload the MACs they already have stored in their local hard disk to the local catalogue. The implementation of the GE must enable users to select their new *.wgt packaged MAC and upload it to the Local Catalogue.



Uploading a packaged MAC to the Local Catalogue

It should also be possible to upload the XML template of the MAC, where the Application Mashup GE is in charge of getting and storing the linked source code of the MAC in the Local Catalogue.



Uploading a MAC from its XML template to the Local Catalogue

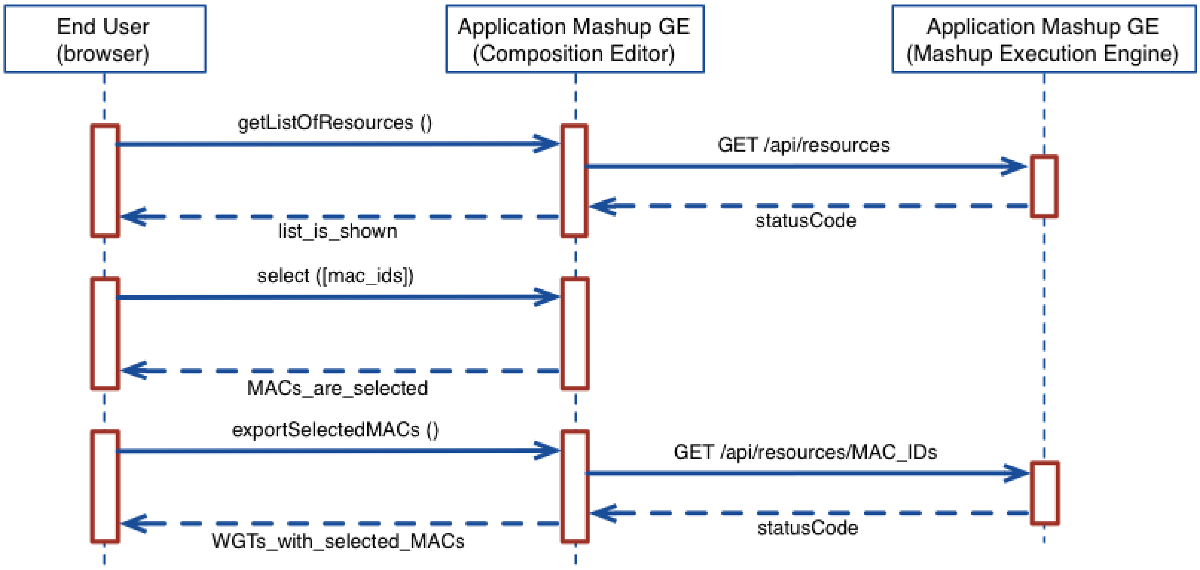
Regardless of the upload method, this interaction must result in the uploaded MAC being stored in the local catalogue, ready for configuration and/or deployment. In other words, the MAC will be at the **Bought/Installed state** of its lifecycle. A `HTTP/1.1 201 Created` Status Code response will be received if the interaction went well.

Note that there are two possible scenarios if the uploaded MAC is a mashup:

- If the widgets within the mashup are currently installed in the local catalogue, the uploaded mashup will reference the widgets and will run out-of-the-box.
- If some of the widgets have a commercial license, and the license has to be bought for the widget to be used, the uploaded mashup will be installed, but the GE must warn and notify users that they have to buy the licensed widgets.

Export a MAC from the local catalogue

Platform users must be able to export (download) any of the MACs they have installed in the local catalogue. Users should select the MACs they want to export and click on an export button for the Application Mashup GE implementation to generate a packaged version of the selected MACs, enabling users to download MACs using their Web browser.



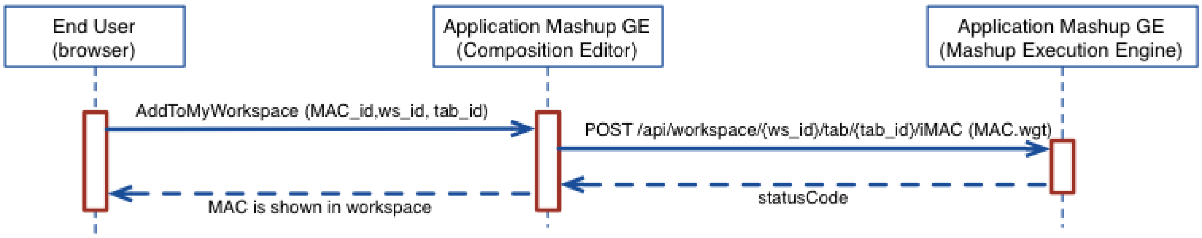
Exporting (downloading) a packaged MAC from the Local Catalogue

Deploy a MAC to get a new runtime instance

The ***Deploy MAC*** functionality instantiates a mashup or widget, that is available in the local catalogue in the Mashup Execution Engine. This operation is invoked by the mashup developer from the Composition Editor. The call should include the following information:

- *macID*: id of the mashup for instantiation.
- *wsID*: id of the active user workspace.
- *tabID*: id of the current tab within the active user workspace.

As a result of this invocation, the engine will get the MAC template from the local catalogue and will execute the MAC. It will also be available in the Mashup Editor mashup developer workspace. Its state will switch to **deployed**.

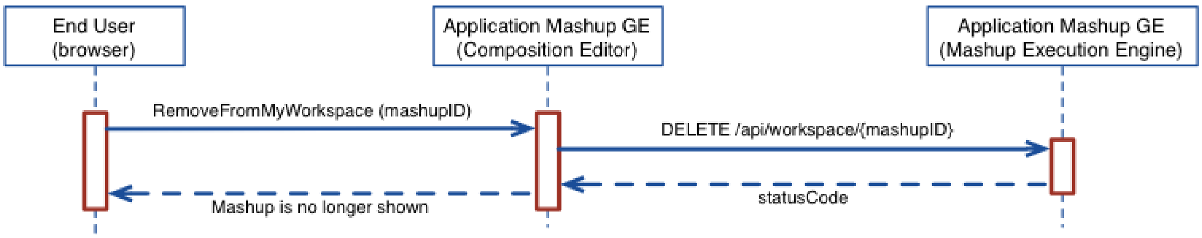


Interaction with the Mashup Execution Engine to deploy both mashups and widgets

Undeploy a MAC. Remove an instance from execution

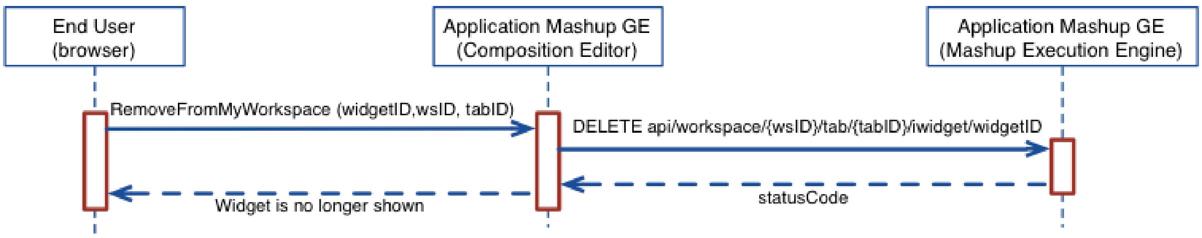
The operation ***Undeploy MAC*** removes a mashup application or a widget from execution. Mashup developers invoke this operation from the mashup composition editor when they want to delete the whole mashup application or just a single widget.

The mashupID is required to undeploy a mashup:



Interaction needed to undeploy a mashup

The widgetID, workspaceID and tabID are required to undeploy a widget.



Interaction needed to undeploy a widget

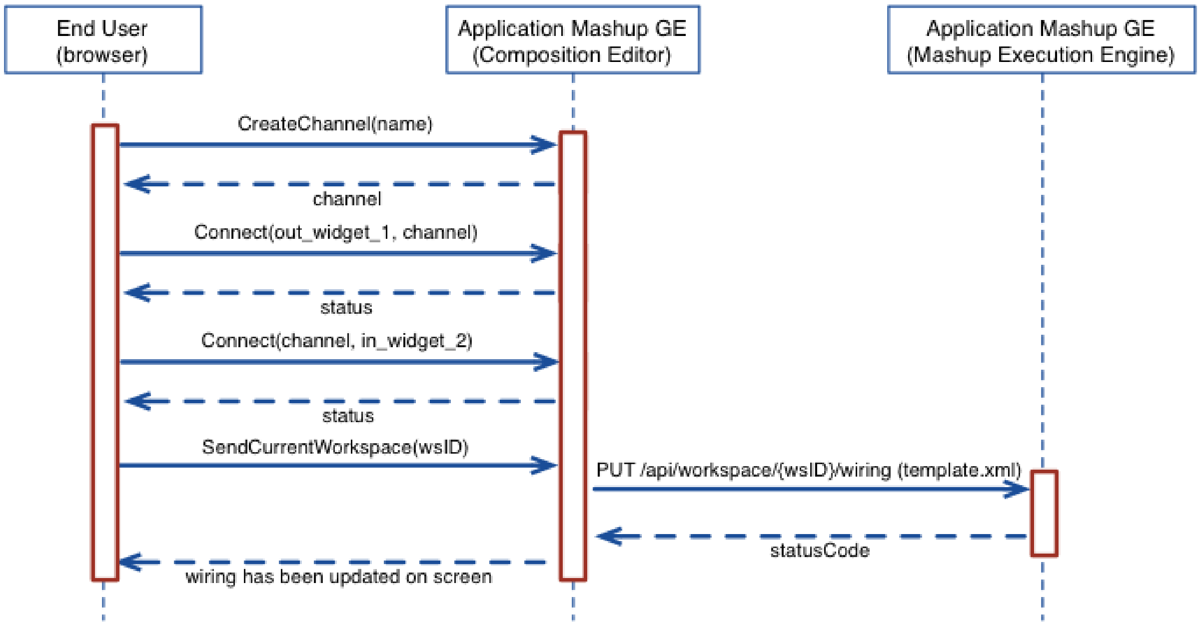
As a result of this operation, the mashup or widget will stop executing, and thus it will reach the **bought/installed** state in its lifecycle.

Interconnect widgets/operators using wiring

Wiring functionality enables users to connect one or more widgets/operators to one or more other widgets/operators by means of a channel. The Composition Editor will help users to connect one or more of the possible outputs of a widget/operator with the input of a another widget/operator. This way, data flows between MACs allowing the mashup application to act as an information and process dashboard.

Besides, piping deals with how operators can bind to a specific backend service to gain access to data provided by the service. Then, users can wire the operator outputs either to other operators in order to perform some kind of data filtering/adaptation processes or to other widgets to consume the data.

The figure below shows how users set up channels between widgets/operators for wiring/piping.



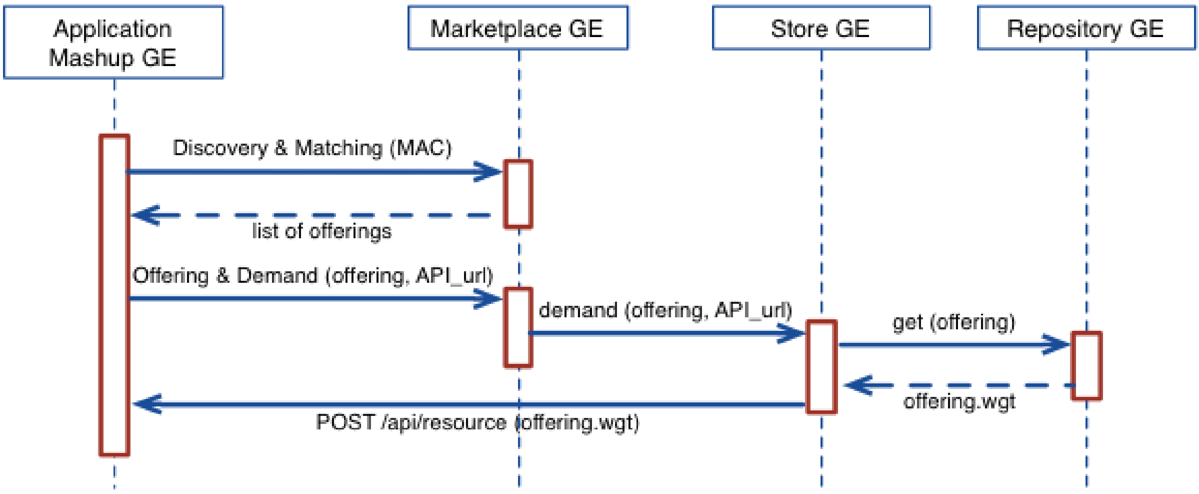
Widget Interconnection Management process

33.5.2.2 *Interactions with other FI-WARE Generic Enablers*

The Application Mashup GE is closely related with those Business-related FI-WARE GEs. The main interactions are as follows.

Buy MACs from a Store

Buying both mashup applications, widgets and operators from an external FI-WARE Store is one of the topics to be taken into account by the Application Mashup GE. Once users have used the Marketplace GE to search and select the MAC that they want to buy, the Application Mashup GE must provide the Store with a URL (through the Application Mashup RESTful API) and request the uploading of the bought MAC. The Store GE will be the actor that uploads the MAC to the local catalogue. Once the MAC has been uploaded, it is automatically installed in the local catalogue. From this point onwards, it will be possible for the MAC to be instantiated and thus executed.

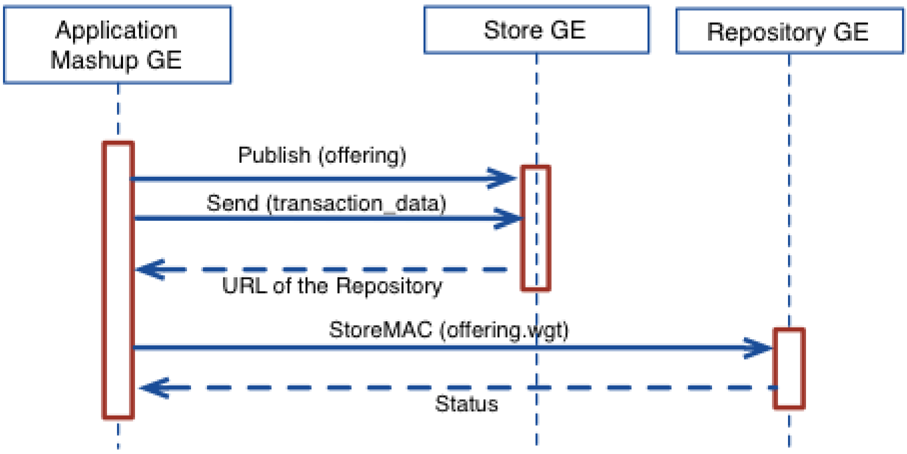


Buying a new MAC from a Store

Publish a MAC to a Store

Once users have either developed and uploaded a new widgets to the local catalogue or created their own mashup application, the Application Mashup GE must enable users to contact the Store GE to publish their brand new MAC.

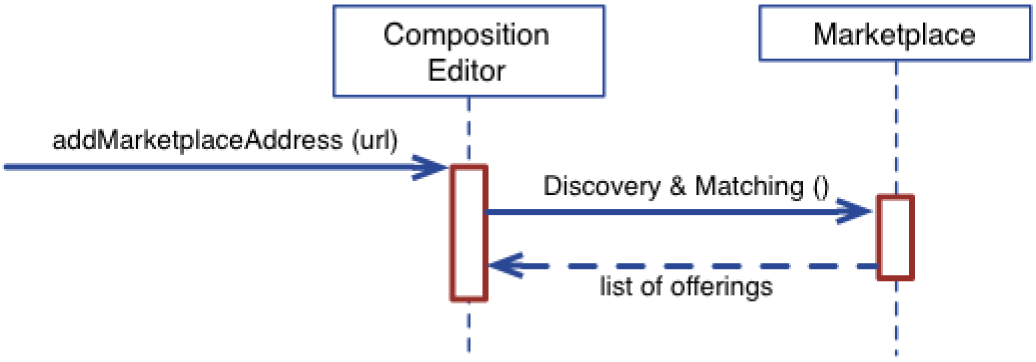
The Application Mashup GE must enable users to notify the Store GE that they intend to publish the MAC. The Store GE is responsible for the entire transactions process (i.e. sending of pricing data, card number, etc.), which is out of the scope of the Application Mashup GE. However, at some point of the interaction, the Application Mashup must send the packaged MAC to the Store/Repository for storage when it is set to **published**.



Publishing a new MAC to a given Store

Add Marketplace

The Application Mashup GE must be able to include new Marketplaces to search for new MACs. To do this, Marketplaces must be accessible from the Editor and new ones could be added from time to time. This functionality adds the Marketplace URI to the Composition Editor's internal list of marketplaces.



Adding the URL of a new Marketplace

33.6 Basic Design Principles

- **API Technology Independence**

  The API abstracts from the specific implementation technology. Implementations using more than one type of platform and framework should be possible.

- **Web Browsers should not limit the functionalities of the Application Mashup GE**

  HTML5, CSS and JavaScript must be used to fully exploit the brand new Web applications capabilities.

- **User-matched interaction abstraction level**

Editors could cater for different user expertise (from technical experts skilled in the composition language to domain experts without technical expertise or even simple end users with no programming or technical skills) and roles (from composed service creators, to resellers and finally prosumers) by hiding complexity behind different types of building blocks, trading off flexibility for simplicity.

34 FIWARE ArchitectureDescription Apps Light-weightedSemantic-enabledComposition

You can find the content of this chapter as well in the wiki of fi-ware.

34.1 Copyright

34.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications. Atos Spain strives to make the specifications of this Generic Enabler available under IPR rules that allow for a exploitation and sustainable usage both in Open Source as well as proprietary, closed source products to maximize adoption.

34.3 Overview

Light-weighted Semantic-enabled Composition is a tooling suite that aims at simplifying the development of domain specific business process as service compositions by exploiting the full potential of semantic technologies. Development of business process in BPM realm requires multi-disciplinary teams that cope with the domain specific knowledge about processes, entities, roles, etc and the ICT technology required to implement them. In particular the number and complexity of SOA-related technologies hamper the implementation of business processes as aggregations of open and Internet accessible services.

In this sense, it is desirable that we simplify the access to these composition technologies to the domain specific experts, as unskilled end users. This can be achieved when the technology itself is capable of producing service compositions without requiring complex information structures, which otherwise can be obtained from end users in a more human readable format.

34.3.1 Target usage

Light-weighted Semantic-enabled Composition is addressing situations where domain specific business processes require to be implemented as service compositions within a certain organization. Business processes are decomposed into different tasks, each one executed by an external service (some provided within the same organization, but others provided by third parties). Commonly, designing and implementing a business process as a service composition requires multidisciplinary teams, ranging from domain business experts (as modelers) to service engineers and integrators. The complexity of these teams, in terms of expertise is required by the modeling activity itself, since modeling a business process as a service composition requires of:

- Business analysts with knowledge on the concrete domain concepts, entities, agents, procedures, etc.

- Business modeling experts, with knowledge on BPM languages, methodologies and practices.
- Service engineers acting either as service providers and/or consumers, who provide or consume services on the domain, and exploit the technological infrastructure that make them possible.
- Service Integrators, who orchestrate services into compositions that offers more complex functionality by aggregating composable functionality.

The involvement of those multidisciplinary teams, their associated procedures and methodologies and required service composition technologies make service composition a complex, time consuming and prone to error activity.

A typical scenario of service composition to reify a business process is as follows. The business analyst describes the domain specific scenario, the required business process and related concepts, entities, roles, etc. The business modeler creates a business process model which is iteratively refined by interacting with the business analyst. Ideally this activity can be performed by the same role. The business process model is implemented as service composition by the service integrator, who aggregates services provisioned by service providers. Commonly, iteratively interactions between the business modeler and the service integrator are conducted to refine and amend the service composition implementation of the business process.

This GE aims at simplifying the implementation of business processes as service compositions, by enabling business modelers to create directly service compositions without requiring the involvement of service integrators. In this way, the business modelers describes the service composition by using domain specific languages (DSL) or vocabularies, in particular ontologies to describe the semantics of the tasks that the composition is decomposed on.

The technical activities conducted by the service integrator are semi-automatically performed by the GE. That is, the GE does not prompt the business modeler to provide technical information required to implement the executable service composition, but the GE obtains that information by exploiting the semantic descriptions attached to the service composition by the modeler from the domain specific ontologies. In this way, a full executable service composition is created without requiring a technical expertise on BPM techniques.

34.3.1.1 *User roles*

For the purpose of this description, two main separate roles are identified:

- Domain business modelers, who has knowledge about the domain within where the business process as service composition will be executed. Therefore, they have the required domain specific knowledge to model the business process as service composition by using domain specific vocabularies (or ontologies)
- Service providers, who will deploy and host the service composition modeler by the business modeler, within the execution environment. They will also take care of composition management at runtime.

34.4 Basic Concepts

This section introduces the most relevant functional concepts considered in the Light-weighted Semantic-enabled Composition GE, which are related to the functional components described in the next section about GE architecture.

Light-weighted Semantic-enabled Composition GE is a tooling suite supporting the implementation of domain business processes as service compositions, executed in backend execution environments as other SOA services. Conceptually, this approach can be decomposed in the following main concepts:

34.4.1 BPMN Composition Edition

A domain business process is decomposed into single working units or tasks, connected logically by a work flow and a data flow, according to the BPMN specification . Each task is performed by an external service, provided by either the same composition provider (same organization) or by an external third party, accessible through open Web standards.

34.4.2 Light-weighted Semantic-enabled Composition

This approach for the modeling of a service composition describes each composition task using semantic descriptions according to some standardized semantic schema (WSMO/OWL-S) and in particular to some light versions (WSMOLite /MicroWSMO). A practical procedure is to annotate each task with concepts selected from a domain specific ontology. These annotations constitute the semantic goal description of the task, that is, the intended purpose of the task. Task goal is matched against the available semantic service descriptions within a semantic knowledge base repository and the matching services are return ranked so the business modeler selects one as task binding or leaves the system to select the best scored.

34.4.3 Semi-automatic Execution Composition Generation

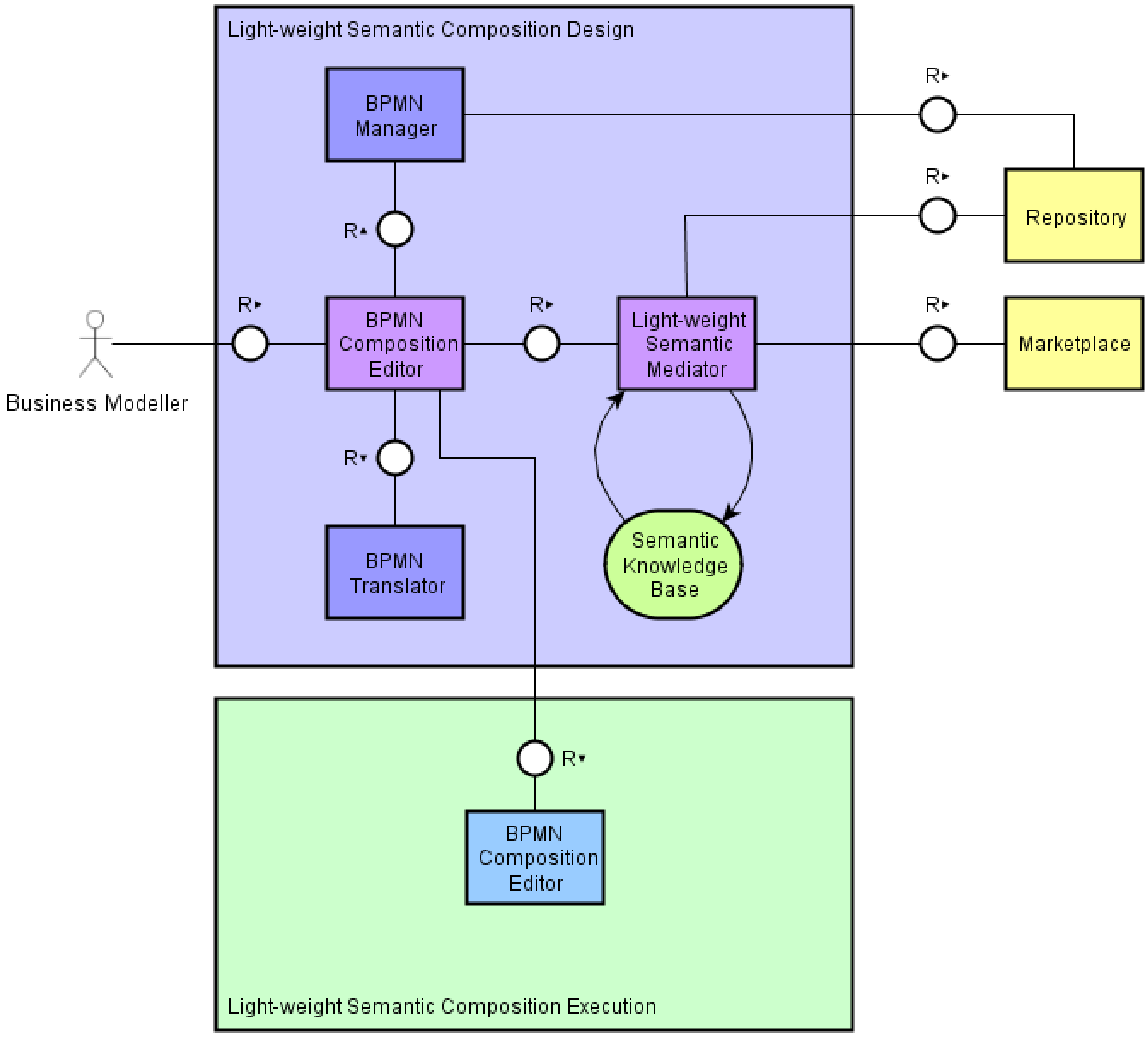
A semantically annotated service composition with tasks bound cannot be executed. A complete service composition including the technical bindings and the data flow mappings needs to be generated out of the semantically annotated composition

34.4.4 Composition Deployment and Execution

The composition is ready to be consumed as soon as the service integrator takes the generated executable service composition, selects a target execution environment and deploys the composition into the target.

34.5 Light-weighted Semantic-enabled Composition Architecture

Light-weighted Semantic-enabled Composition GE architecture is depicted in next figure. This architecture is split into two main functional layers: design and execution.



Light-weighted Semantic-enabled Composition GE Architecture

The design layer provides functional support for the modeling of service compositions. This GE constrains service composition modeling to use BPMN 2.0 for both the graphical (modeling) and execution semantics. That is, this GE assumes that service compositions models are instances of the BPMN 2.0 standard metamodel.

The following functional components are part of the design layer of this GE:

- BPMN Composition Editor enables business modelers to create service compositions using the BPMN 2.0 Graphical notation and execution semantics. It provides typical GUI to create, edit and manage service compositions, requiring the business modeler to have some background in BPM modeling.
- Light-weighted Semantic Mediator complements the BPMN Composition Editor by adding additional semantic-enabled modeling aids that simplifies the modeling process. These assisting features allow business modelers to describe the composition tasks, bind matching services, generate the data flow mapping, and so on.
- Semantic Knowledge Base complements the Light-weighted Semantic Mediator with a semantic repository of semantic service descriptions, domain specific vocabularies

(DSL, ontologies), service composition annotations (tasks, compositions themselves), etc. This component provides content access features, including querying and reasoning.

- BPMN Manager provides BPMN model management, including features to validate and complete BPMN models with required executable information (service bindings, data flow mappings, etc)
- BPMN Translator provides translation capabilities to other executable composition formats, such as BPEL 1.2/2.0
- Composition Deployer acts as a proxy between the Light-weighted Semantic-enabled Composition layer and the Composition Execution layer. It manages the service composition deployment process into the selected Composition Execution layer.

Components in the design layer interact with some other GE components such as the Marketplace and Repository:

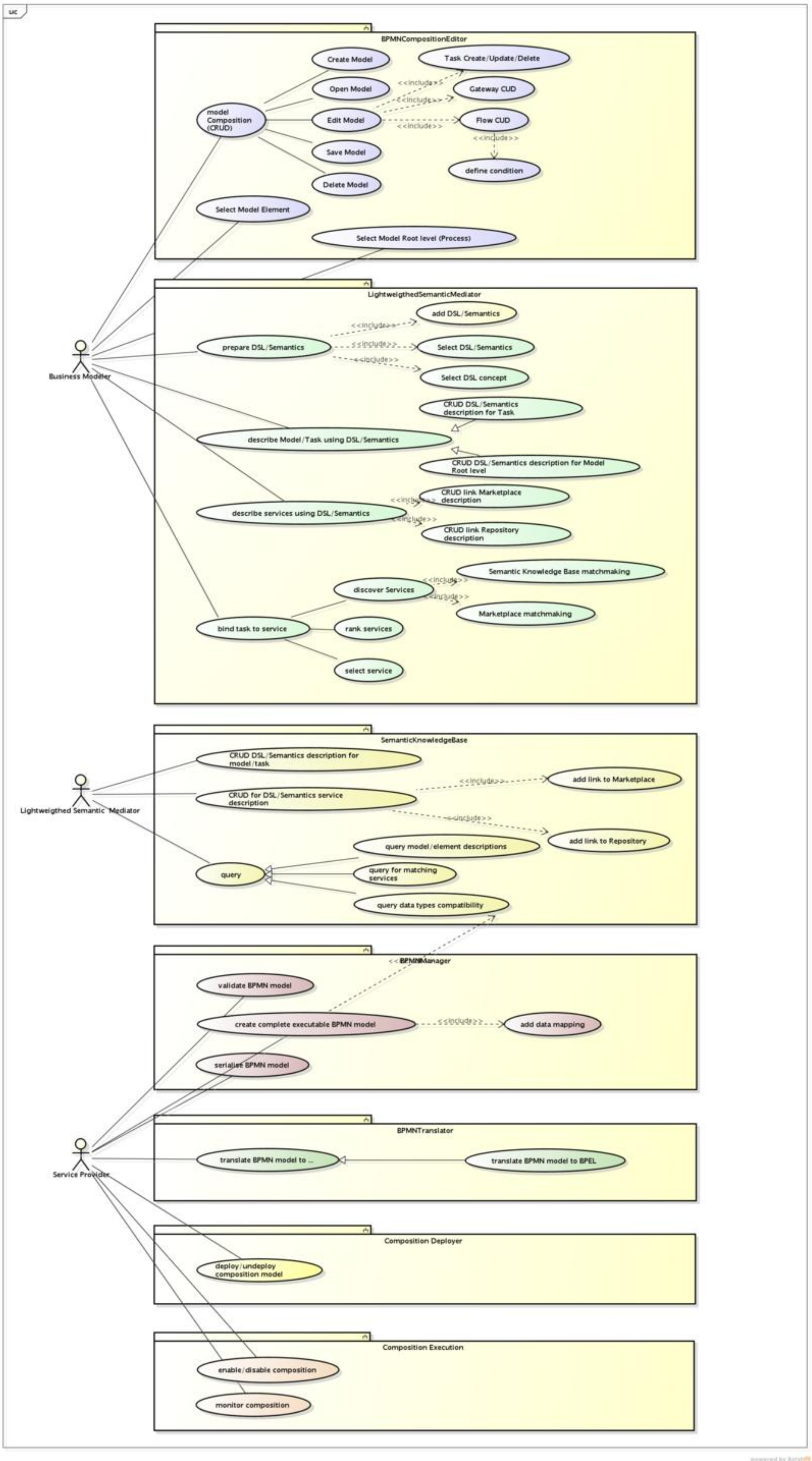
- Marketplace is used to query and retrieve USDL service descriptions for those services matched to composition tasks through semantic matchmaking, thanks to links contained within their semantic descriptions.
- Repository contains the technical descriptions (i.e. WSDL , WADL , etc) of services aggregated in the composition and the composition themselves: the composition models (BPMN) and their technical descriptions (WSDL, WADL, etc)

The Composition Execution layer contains the Composition Execution component, which deploys, enables and executes the service compositions, upon remote invocation by a service consumer.

Main interactions performed by the components that comprise this GE are described in next section, grouped by a functional classification

34.6 Main Operations

A functional classification of Light-weighted Semantic-enabled Composition GE main features is depicted in the next UML use case diagram, and described in more detail in next paragraphs.



Light-weighted Semantic-enabled Composition GE functional classification

34.6.1 Model Composition

This GE provides support for service composition modeling, assuming BPMN 2.0 as graphical notation and execution semantics. Through the BPMN Composition Editor, business modelers can create or open composition models, modify them (edit) and manage them (save, delete). Composition models are stored within a Repository.

Composition edition also includes support for create/update/delete features for composition elements, such as service tasks, gateways (exclusive, parallel), flows and events (start, end).

Composition editor allows to select the composition itself or concrete composition elements.

34.6.2 Prepare DSL/Semantics

In this GE approach, the business modelers describe composition models and their elements by annotating them with concepts taken from concrete domain specific languages DSL (or vocabularies) which provide concrete semantics. From operational point of view, it is common to use ontologies as DSL or vocabularies. The Light-weighted Semantic Mediator enables the business modeler to:

- Register new DSL/Ontologies within it
- Select a concrete DSL/Ontology for a given domain modeling context. Select a concrete DSL/Ontology concept within the domain ontology. These concepts are used to annotate and describe a composition model and their elements.

34.6.3 Describe Model Composition/Task using DSL/Semantics (Business modelers)

The Light-weighted Semantic Mediator enables the business modeler to describe the composition model and its elements using semantic annotations.

In the scope of a composition task, the annotations constitute a description of the task. In other words, they describe the goal of the task. This goal will be used in the service matchmaking process to look for services whose semantic description will match it. A semantic task description is constituted by several annotations of certain type according to the semantic schema used to represent the goal (i.e MSM)

In the scope of the composition itself, the annotations constitute a description of the global composition requirements, preferences and contextual information.

34.6.4 Describe Service using DSL/Semantics (Service Providers)

Light-weighted Semantic-enabled Service Composition GE approach assumes that composable services are described using light semantics. Those semantic service descriptions are available within the Semantic Knowledge Base, and are provided by service providers. A service composition created by applying this GE approach is a service by its own, whereby the business modeler, acting as service provider, is required to provide this semantic description. Same applies to any other third party service intented to be composed by others.

The Light-weighted Semantic Mediator enables service providers to create semantic descriptions compliant to the semantic schema used by the complete GE solution. The concrete schema is left for the implementation, but it should be consistent along with all components that use it.

The schema includes links to the business oriented description stored in the Marketplace, and the technical description stored in the Repository.

34.6.5 Task binding

One of the main jobs in service composition modeling is to bind every task the composition is divided out to a matching service that performs the task. A business modeler can conduct this task binding per task or for the whole composition. The Light-weighted Semantic Mediator enables the modeler to discover matching services based on task goal criteria, rank them according to preferences or non-functional requirements (NFR) and select one service, which is bound to the task. Those activities are typically performed by querying the Semantic Knowledge Base.

34.6.6 Validate, generate, translate executable BPMN composition model

Next step in service composition modeling consists on filling the missing information that the composition model requires before being shipped for deployment and execution. Examples of missing information are:

- Task binding technical description. For each BPMN 2.0 service task, a concrete task binding information has to be included, by inspecting the technical description (i.e. WSDL)
- Data flow mapping, including IO mappings at task and composition level

Once the service composition model has been completed with missing required executable information, the composition model is validated (BPMN 2.0 compliance validation) and serialized (for storage and deployment).

Optionally, the composition model can be translated from its original BPMN 2.0 format to another mappable format, such as BPEL 1.2/2.0 . This is required when the select target environment for execution is not BPMN 2.0 compatible.

34.6.7 Deploy composition model

Full executable validated composition models can be deployed into the selected target Composition Execution environment, using the Composition Deployer. Once deployed, the service composition is enabled, being ready to received incoming requests from service consumers.

Similarly, deployed service compositions can be undeployed anytime.

34.6.8 Composition Execution

During the execution time, deployed services can be enabled or disabled any time through the Composition Execution UI. Besides, running compositions (enabled) can be continuously monitored and monitoring data can be collected for given time frames.

Next paragraphs detail main operations using UML sequence diagrams for the most relevant scenarios concerning the light-weighted semantic modeling of a service composition.

34.6.9 Modeling a BPMN Composition

A business modeler, through the BPMN Composition Editor, can either:

- Create a new BPMN composition model

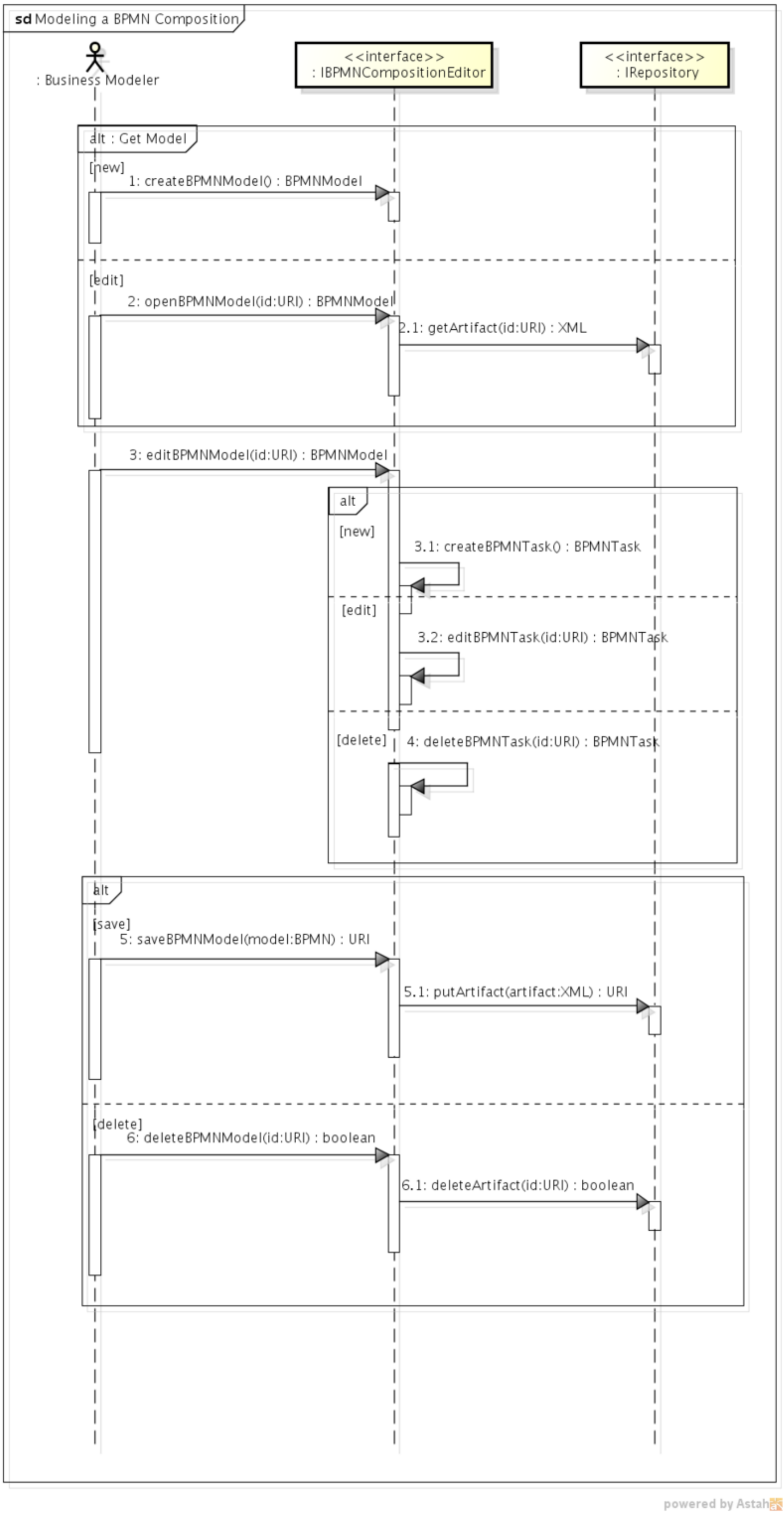
or

- Open an existing one from the Repository, using a unique model Id. The unique model id is provided by the Repository during the model saving.

The business modeler can work on this composition model, editing the model and its elements. This activity includes create/edit/delete operations on model elements such as tasks, gateways, flows, events, etc. Each model element is uniquely identify by a unique identifier (within the model) provided by the editor upon creation.

Anytime, during the modeling of the composition, the business modeler can either:

- Save the composition model into the Repository. This process requires to serialize the BPMN composition model according to the BPMN 2.0 serialization standard (XSD/XMI)
- Delete the model from the Editor and Repository (in case the model was previously saved). Models are identified within the repository by a unique identifier.



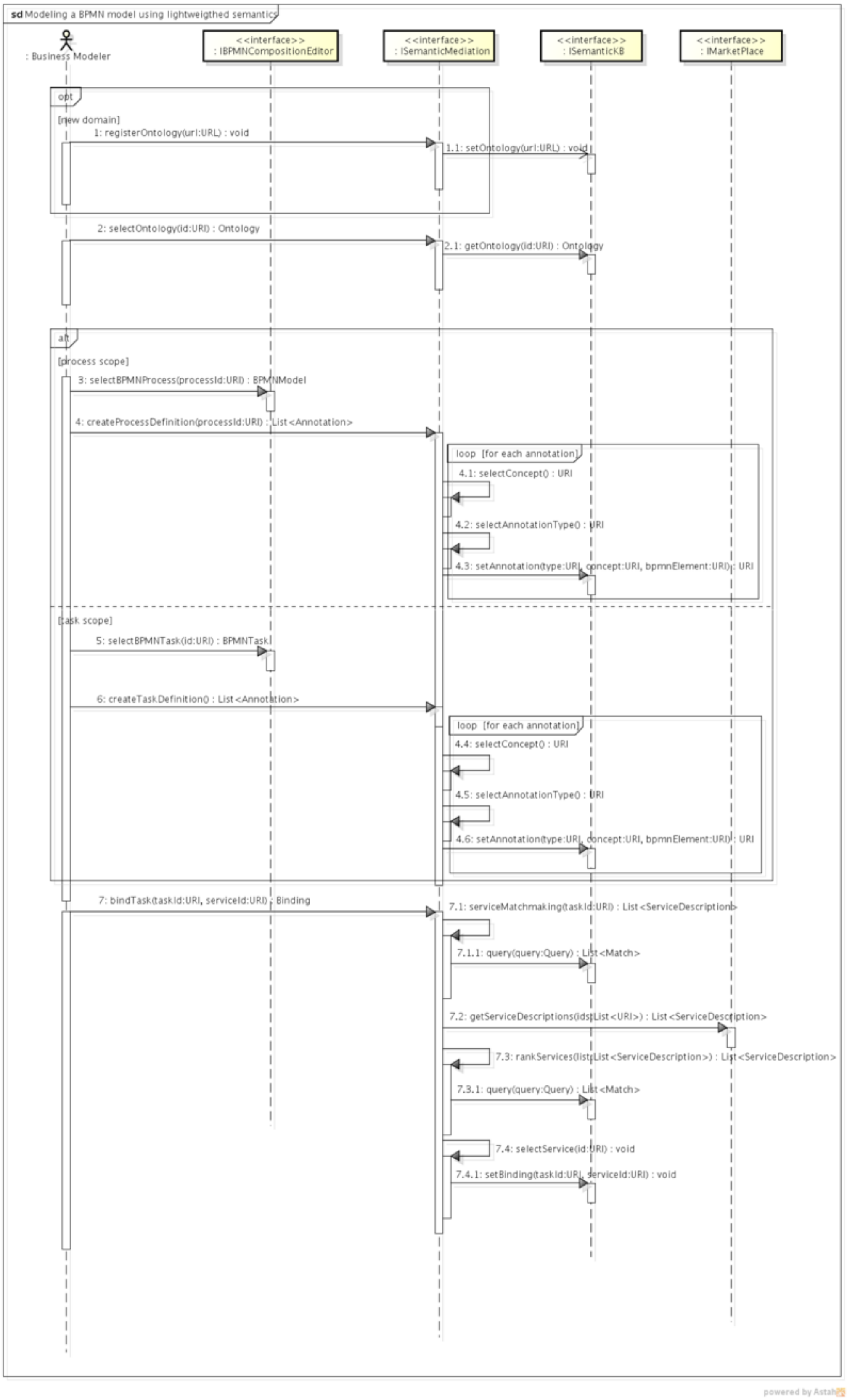
Modeling a BPMN Composition

34.6.10 Modeling a composition using light-weighted semantics

A composition work-flow and its task decomposition can be modeled by using the BPMN Composition Editor itself, but once each task has to be bound to concrete services or the data flow mapping has to be designed, is when the light-weighted semantic composition approach comes up.

This GE encourage modelers to describe composition task by attaching light-semantic annotations, according to some pre-established task goal schema (for instance WSMOLight/MicroWSMO) which are taken from domain specific ontologies (or any other domain specific language (DSL) or vocabulary). Based on this semantic task goal, the semantic matchmaking activity determines the best matching service and bounds the task to it.

Next UML sequence diagram describes the overall process in detailed, decomposed by operation and involved component.



Modeling a composition using light-weighted semantics

34.6.10.1 *Domain Ontologies preparation phase*

A business modeler selects the suitable domain specific ontology that will be used to describe the composition tasks, depending on the concrete context the composition will be applied.

If the domain ontology has not being previously registered within the Light-weighted Semantic Mediator, the modeler registers it by giving the ontology URL. The ontology is downloaded from that URL and stored within the Semantic Knowledge Base. The ontology has to be accessible in a compatible format with the Semantic Knowledge Base (i.e. serialization format such as RDF/XML , N3 , etc

Any time during the composition modeling the business modeler can switch from one domain ontology to another by selecting them in the Light-weighted Semantic Mediator. The selected ontology is loaded from the Semantic Knowledge Base. Ontologies are uniquely identified within the GE implementation by an URI.

34.6.10.2 *Lightweighted semantic composition modeling*

A business modeler can start this activity either by annotating the composition itself (global annotations) or concrete composition elements (tasks in particular).

Global annotations describe the composition global requirements, preferences and contextual constrains, as stated by the selected semantic annotation schema (i.e. WSMOLite/MSM). Each annotation (concretised by an selected ontology concept, given its URI and its type) is stored within the Semantic Knowledge Base, given the annotation type, annotation concept and process URI.

Tasks are light-weighted sematically annotated to describe them, according to the selected semantic annotation schema (i.e. WSMOLite/MSM) as well. Each annotation is stored within the Semantic Knowledge Base, given the annotation type, annotation concept and task URI.

Once the business modeler has described all the tasks within the model using light-weighted sematic annotations, he proceeds to bind each task to a concrete external service. This can be done automatically for the whole composition, whereby all tasks are automatically bound the the best ranked compatible service matched by the matchmaking process, or semiautomatically, task by task where selection is conducted by the business modeler. Nonetheless, this process either manual or automatic is similar.

The Light-weighted Semantic Mediator is invoked to search for services whose semantic descriptions are matching the task goal description. Using this local task goal description, a semantic query (i.e. SPARQL) is prepared and sent to the Semantic Knowledge Base, which returns a list of unranked matches (candidate service descriptions). Base on global annotations (requirements, preferences and context constrains), candidate services are filtered out (according to requirements and context contraints) and ranked (based on preferences), by querying and reasoning the Semantic Knowledge Base.

Finally the best ranked candidate service is automatically selected or the business modeler selects one by inspectind the service candidate list, using descriptions obtained from the Marketplace. Selected service is bound to the task in the Semantic Knowledge Base.

34.6.11 Process a complete executable BPMN composition model

A semantically fully processed composition model, once its tasks have been bound to semantically annotated services, contains all the information required to create an executable composition model.

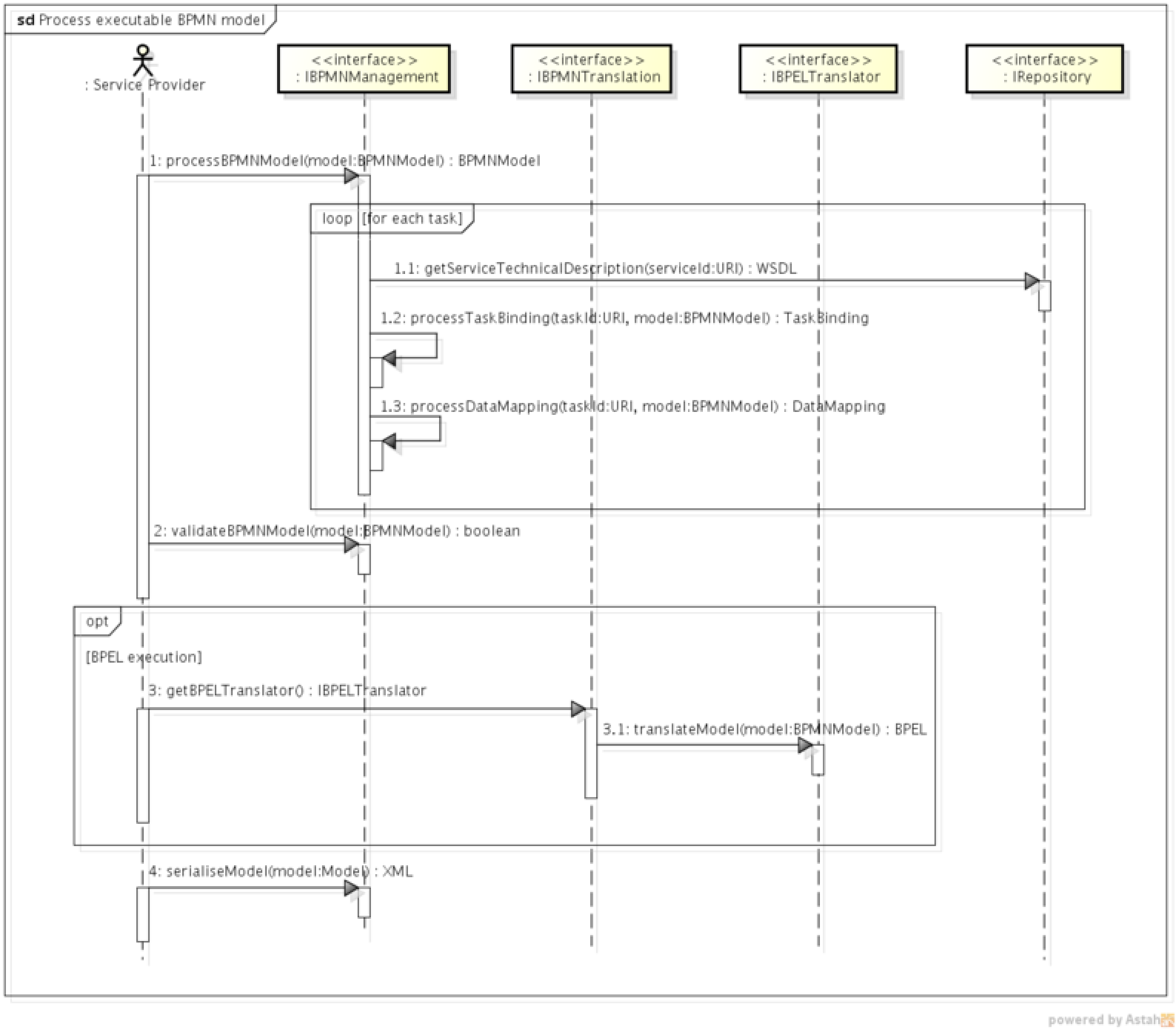
This procedure is initiated by the service provider through the Light-weighted Semantic Composition Editor, by invoking to process the BPMN model. This procedure conducts two main jobs for each composition task:

- Task binding is included in the BPMN composition model, according to the BPMN specification, by getting the technical required information from the technical description stored within the Repository
- Task data flow mapping (IO mapping) is included in the BPMN composition model, according to the BPMN specification as well. Data flow mapping considers all data objects available before reaching this task following the work flow.

Once the BPMN executable composition model has been created, it is validated against the BPMN specification.

Optionally, the BPMN executable composition model can be converted into another executable model, compliant to another composition language, such as BPEL 1.2/2.0. This could be required by the target Composition Execution Environment.

Finally, the executable composition model is serialized into XML or other standardized serialization schema for interchange as determined by the selected target execution language (BPMN XSD/XMI, BPEL XSD, etc)

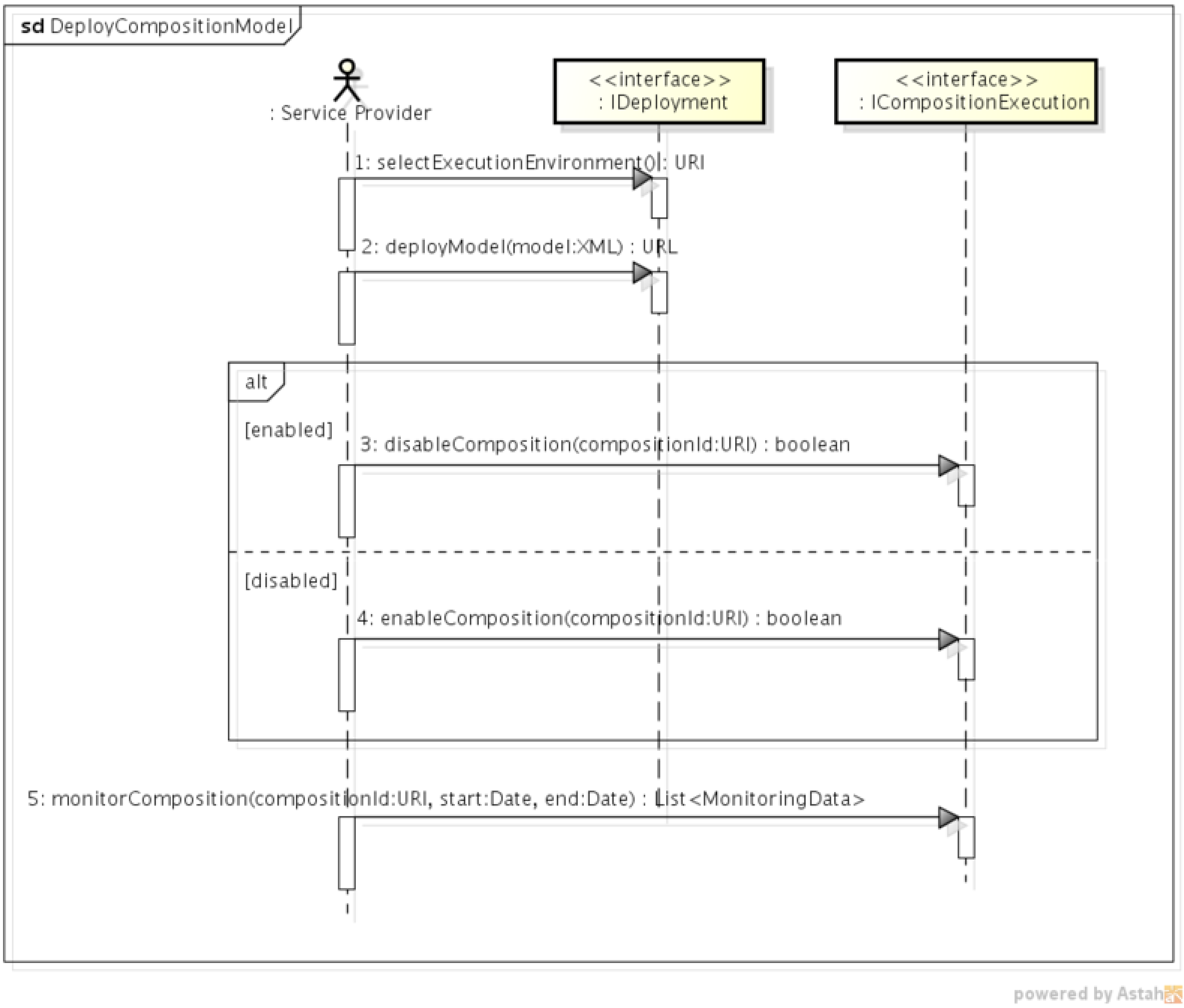


Process a complete executable BPMN composition model

34.6.12 Deploy a service composition model

A service composition model can be deployed within a selected target. A service provider selects the target execution environment and deploys the current composition model edited in the Light-weighted Semantic Composition Editor. The Deployer returns the URL where the deployed composition is listening as service or any other required information to invoke it.

Anytime after deployment, the service provider can enable or disable the composition (depending on its status) through the Composition Execution UI. Moreover, through this UI, the service provider can monitor a composition within a specified time frame.



Deploy a service composition model

34.7 Design Principles

Sources:

- SOA4All D1.1.1 Design Principles for a Service Web
- Evaluation of Service Construction

This GE follows the following design principles:

- **Machine and human based computation principle**

This GE allows a semi-automatic (human and machine) service composition modeling

- **Template-based composition**

Recently created process models (and some fragments) can be reused in future modeling tasks

- **Reusable**

This GE is completely domain independent concerning the knowledge intensive reusability feature.

- **Composability principle**

The GE design allows it to split a complex process-modeling problem into several smaller ones that the GE resolves separately by its specific agents.

- **Openness principle**

This GE can be easily extended either by coding/replacing architectural components

- **Ontology based principle**

This GE extensively uses ontology based knowledge.

34.8 Detailed Specifications

34.8.1 Open API Specifications

The Light-weighted Semantic-enabled Composition GE is not exposed as a service but as a Webtop application (GUI), accessed throught the end user Web browser. Although some GE components are exposed as services, they only expose an API for internal consumption (within the GE), but it is not foreseen they will be integrated by other GEs.

The Light-weighted Semantic-enabled Composition GE will access the Marketplace and Repository REST APIs:

- FIWARE.OpenSpecification.Apps.MarketplaceSearchREST
- FIWARE.OpenSpecification.Apps.RepositoryREST

34.9 Re-utilised Technologies/Specifications

The Light-weighted Semantic-enabled Composition GE relies on following technical specifications:

- BPMN 2.0
- WSMOLite
- MicroWSMO
- BPEL 2.0
- USDL
- WSDL 2.0
- WSDL
- Minimum Service Model (MSM)
- BPMN to BPEL
- BPMN XSD/XMI serialization formats

- [RDF/XML format]
- [N3 format]
- [SPARQL 1.1]

35 Security Architecture

You can find the content of this chapter as well in the wiki of fi-ware.

35.1.1 Introduction

The internet and digital technologies are transforming our society by driving economic growth, reduces barriers to trade, connecting people across the world and providing new ways to communicate and co-operate.

Our dependence on cyberspace is increasing and while cyberspace fosters open markets and open societies, unfortunately this very openness can also make us more vulnerable to a growing number of criminals, hackers, activists, foreign intelligence services, who want to exploit children and the vulnerable, to harm individuals, enterprises, public and private organizations by compromising or damaging revenues, reputations, privacy, business, critical infrastructures, sensitive data…

Cyber criminals demonstrate their ability to adjust quickly to new developments like mobile phone, “PlayStations”... The collective impact of this threat now has the potential to cause significant damage to online economies, affecting individuals and societies.

For the first time in 2012, in its Global Risks 20212 report (Seventh Edition), the World Economic Forum put cyber attacks fourth in the Top 5 Global Risks in terms of Likelihood (page 12). In this report, we read (page 11 –“The Dark Side of Connectivity”): “ The impacts of crime, terrorism and war in the virtual world have yet to equal that of the physical world, but there is fear that this could change. Hyper connectivity is a reality. With over five billion mobile phones coupled with internet connectivity and cloud-based applications, daily life is more vulnerable to cyber threats and digital disruptions”.

Also Future Internet services will always be exposed to different types of threats that can lead to severe misuse and damage. Creating secured and trusted services without sacrificing much of the desired functionality, usability, performance and cost efficiency is a great challenge, especially in a dynamic environment where new threats and attack methods emerge on a daily basis.

FI-WARE will provide an environment in which a diverse range of services are offered by a diverse range of suppliers, and users are likely to unknowingly invoke underlying services in a more dynamic and ad hoc manner. Moving from today’s static services, we will see service consumers that transparently mix and match service components depending on service availability, quality, price and security attributes. Consequently, the applications seen by the end-users may be composed of multiple services emanating from many different providers, and the end user cannot really know if the security solutions implemented by a service provider are compliant with security policy claimed.

In this context it becomes essential to have means of security monitoring extremely efficient and respond quickly to attacks. Of course, the Security monitoring covers more common threats, like toll-fraud, impersonation, service high jacking etc. To defend ourselves, Future Internet services need more intelligent early attack detection and support for decision and rapid action making faced with constantly evolving threats. This is one of the challenges of FI-WARE.

The current landscapes of service delivery ecosystems do not fully address principals such as openness, usability and simplicity. FI-WARE aims to balance between simplified service usage and end user trust (including underlying security) in the service. FI-WARE will be designed in a flexible manner in order to reflect generic as well individual requirements. By that FI-WARE will be easily adaptable to upcoming needs. Furthermore this also is supported by including social interactions being part of the working community, e.g. by offering a “security market place” where anyone interested could contribute. A typical example of such a marketplace can be the sharing of vulnerable configuration descriptions within a community of users, allowing faster reactions and even prevention from potential attacks exploiting these vulnerabilities.

35.1.2 Architecture Overview

The overall ambition of the Security Architecture of FI-WARE is to demonstrate that the Vision of an Internet that is "secure by design" is becoming reality. Based on achievements to date and/or to come in the short-term (both from a technological but also a standardization perspective) we will show that "secure by design" is possible for the most important core (basic) and shared (generic) security functionalities as anticipated by the FI-WARE project and in accordance with the requirements of external stakeholders and users. The “secure by design” concept will, therefore, address both the security properties of the FI-WARE platform itself and the applications that will be built on top of it. As such, the Security Architecture will focus on key security functionalities such as identity management or security monitoring to be delivered as so-called generic security enablers that will be integrated with the design and implementation of the FI-WARE.

Security, Privacy and Trust in FI-WARE is mainly focusing on delivering tools and techniques to have the above-mentioned security needs properly met. Furthermore a decision making support and the automation of countermeasures allow alleviating the workload of users and administrators while raising their security awareness.

The high-level Reference Architecture sketched in Figure below is formed by four main modules:

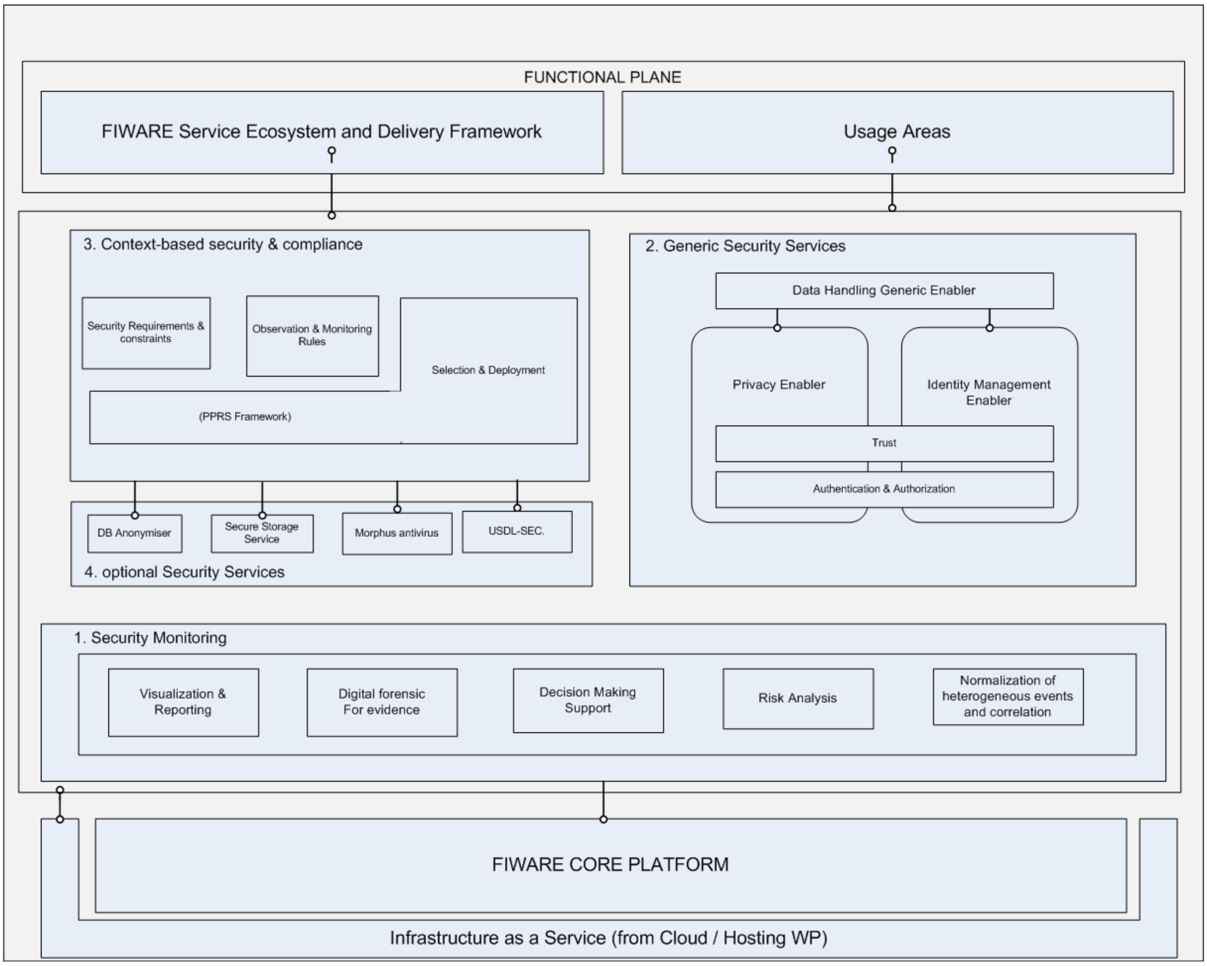
1. Security monitoring,

2. Generic Security Services: Identity Management, Privacy, Data Handling,

3. Context-Based Security and Compliance,

4. Optional Generic Security Services: Secure Storage Service, Morphus antivirus, DB Anonymiser.

Theses services will be instantiated at runtime.



FI-WARE High Level Security Architecture

35.1.3 Architecture Description of GEs

- FIWARE.ArchitectureDescription.Security.Security Monitoring
- FIWARE.ArchitectureDescription.Security.Context-based security & compliance
- FIWARE.ArchitectureDescription.Identity Management Generic Enabler
- FIWARE.ArchitectureDescription.Security.Privacy Generic Enabler
- FIWARE.ArchitectureDescription.Security.Data Handling Generic Enabler
- FIWARE.ArchitectureDescription.Security.Optional Security Enablers

35.1.4 Other Open Specifications

- FIWARE.OpenSpecification.Security.USDL-SEC (PRELIMINARY)

36 FIWARE ArchitectureDescription Security Security Monitoring

You can find the content of this chapter as well in the wiki of fi-ware.

36.1.1 Overview & Architecture

The Security Monitoring GE is part of the overall Security Management System in FI-WARE and as such is part of each and every FI-WARE instance. The target users are: FI-WARE Instance Providers and FI-WARE Application/Service Providers.

Security monitoring is the first step towards understanding the real security state of a future internet environment and, hence, towards realizing the execution of services with desired security behaviour and detection of potential attacks or non-authorized usage.

This generic enabler deals with the security monitoring and beyond, up to pro-active cyber-security i.e. protection of “assets” at large. It allows to assess the real security state of a future internet environment and, hence, towards realizing the execution of services with desired security behaviour and detection of potential attacks or non-authorized usage.

The main concerns of Security Monitoring are:

1. Detect vulnerabilities and identify risks

2. Score vulnerabilities impact and assess risks

3. Analyze events to correlate and detect threats and attacks

4. Treat risks and propose counter-measures

5. Visualize result alarms and residual risks in order to allow efficient monitoring from the security perspective.

We detail in the following the interactions between the components of the Security Monitoring Architecture, as well as their respective connections to the FI-WARE framework. We start by the three blocks composing the input for the Heterogeneous Event Normalization Service. The aim of this service is to normalize heterogeneous events so that they can be processed by the Service oriented SIEM. In order to be correlated by the SIEM, the events must be pertinent for the risk analysis.

Events into the front of this service are:

1. Context-Based Security & Compliance violation events, from GE provided by ATOS and SAP in WP8

2. Secure Storage Service events from GE provided by TCS in WP8.

3. Cloud, Internet of Things and Interface-to-networks events from GE provided in WP4, WP5 and WP7.

As for the Heterogeneous Event Normalization Service itself, it is part of WP8 and provides inputs for Service-level SIEM, Forensics Framework, and eventually Complex-Event Processing in Data/Context Management of WP6.

The Service-level SIEM provides its results directly to the Visualization Framework. Complex-Event Processing on the other hand, serves as input for both the Forensics and Visualization Frameworks. It can be deduces that the CEP can potentially be bypassed.

The Security Monitoring enabler is intended to be used to assess compliance to the security requirements of Business Framework for the Applications and Services Ecosystem and Delivery (WP3).

Security Monitoring employs the Complex Event Processing from Data/Context Management (WP6).

Mulval Attack Paths Engine in Security Monitoring utilizes the Vulnerability Collections from Cloud/IOT/I2ND, the Vulnerabilities Database (NVD), and the Configuration Management Database (CMDB) in WP4, the latter being also involved. From the internal viewpoint of Security Monitoring, the MulVAL Attack Paths Engine includes in its entries the Vulnerability Scanners operating on the network, and the Fuzzer block for assessing the applications' security.

Business-oriented Vulnerability requires as input the Configuration Management Database (CMDB) in WP4, and the vulnerability scoring it provides is employed by the MulVAL Attack Path Engine, along with the other inputs of the latter.

By combining the input from the Botnet Tracking System with the one from the MulVAL Attack Paths Engine, the Counter-Measures App yields the proposed output to the Visualization Framework for further monitoring and decision-making purposes.

The decision making support will aim to provide some help to the security operator by proposing several possible countermeasures / remediation that could be deployed in the monitored system / services. To facilitate the decision making processes, assets contribute to early warning of harmful events, for the detection of suspicious behaviour, for correlation of heterogeneous security events and for the computation of critical attack paths. In addition, the man-machine interfaces ensure that solutions are effectively designed for end-users, providing them with increased efficiency. This would include advanced visualisation techniques to provide a more complete picture to handle complex situations efficiently.

Finally, a digital forensics for evidence consist to develop capabilities to trace illegal activity in cyberspace back to its origin. Correlating events provides the means to support the search for evidence process. Timeframe analysis will can be useful in determining when events occurred. For this, we will can review the time and data contained in the file system metadata, linking error logs, connections logs, security events, alarms and files of interest to the timeframes relevant to the investigation.

The Security monitoring GE Chapter meets the requirements of ISO 27001 (to see 4.2 Establishing and managing the ISMS).

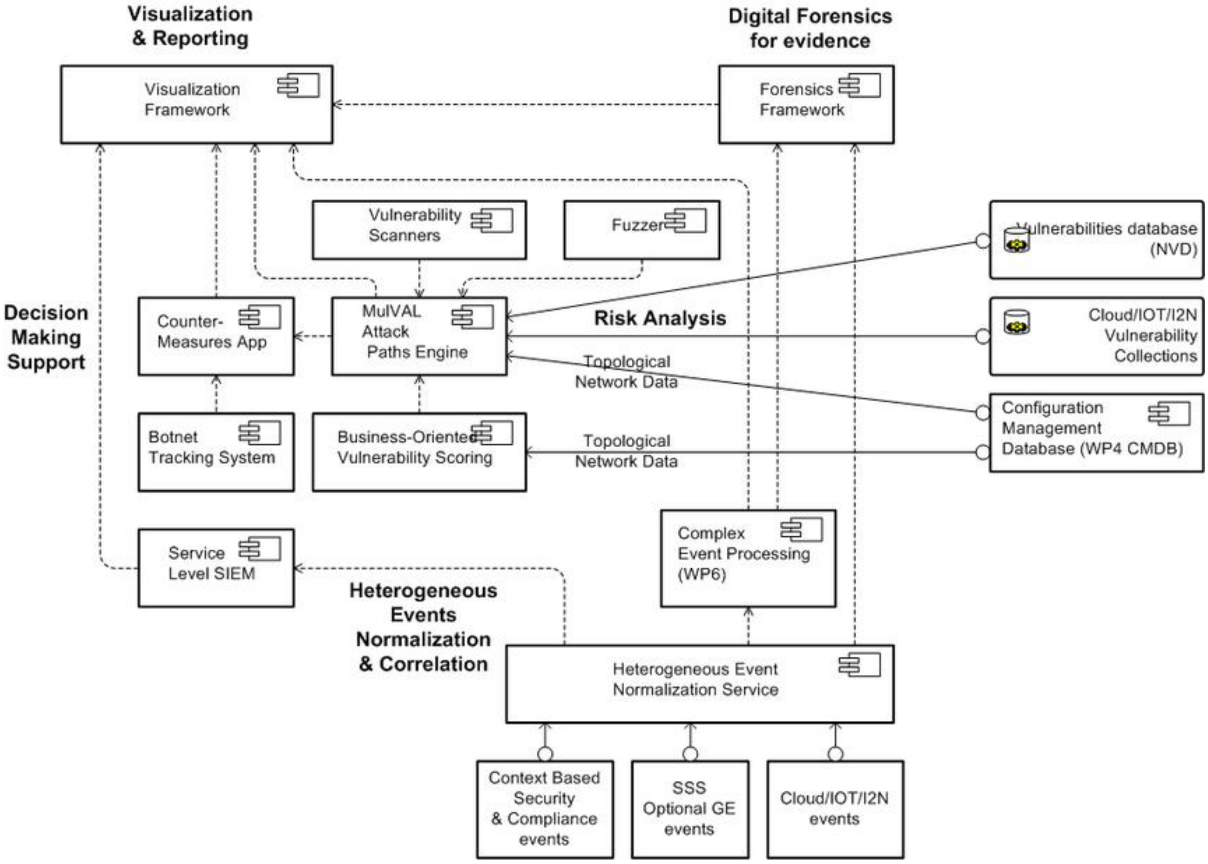
Among others things, it provides an answer to "c" paragraph (...Identify a risk assessment methodology that is suited to the ISMS), to "d" paragraph (identify the risks); "e" paragraph (Analyse and evaluate the risks); to "f" paragraph (Identify and evaluate options for the treatment of risks) and to "g" paragraph (Select control objectives and controls for the treatment of risks.)

In conclusion, the security monitoring enabler is composed of the following functionalities:

- **Normalization of heterogeneous events and correlation**. This functionality covers the normalization and correlation of massive and heterogeneous security events.
- **Risk analysis**. Considering the threat profiles and the related system vulnerabilities, a risk profile is built for each threat, containing qualitative values which measure the impact of the outcome of threats to the organization.

- **Decision making support**. Countermeasures can be selected in order to mitigate the risks, for instance implementing new security practices within the organization, or taking the actions necessary to maintain the existing security practices or fixing the identified vulnerabilities.
- **Digital forensics for evidence**. it deals with the acquisition of data from a source, the analysis of the data and extraction of evidence, and the preservation and presentation of the evidence. The digital evidence is intended to facilitate the reconstruction of events found to be malevolent or helping to anticipate unauthorized actions.
- **Visualization and reporting**. It will provide a dynamic, intuitive and role-based User System Interface for the various stakeholders to use in order to understand the current security situation, to make decisions, and to take appropriate actions.

The GE as envisaged will address security monitoring and beyond, up to pro-active cyber-security i.e. protection of “assets” at large. The figure below provides a high-level initial architectural sketch of the Security Monitoring GE as envisaged in FI-WARE.



36.1.2 Basic Concepts

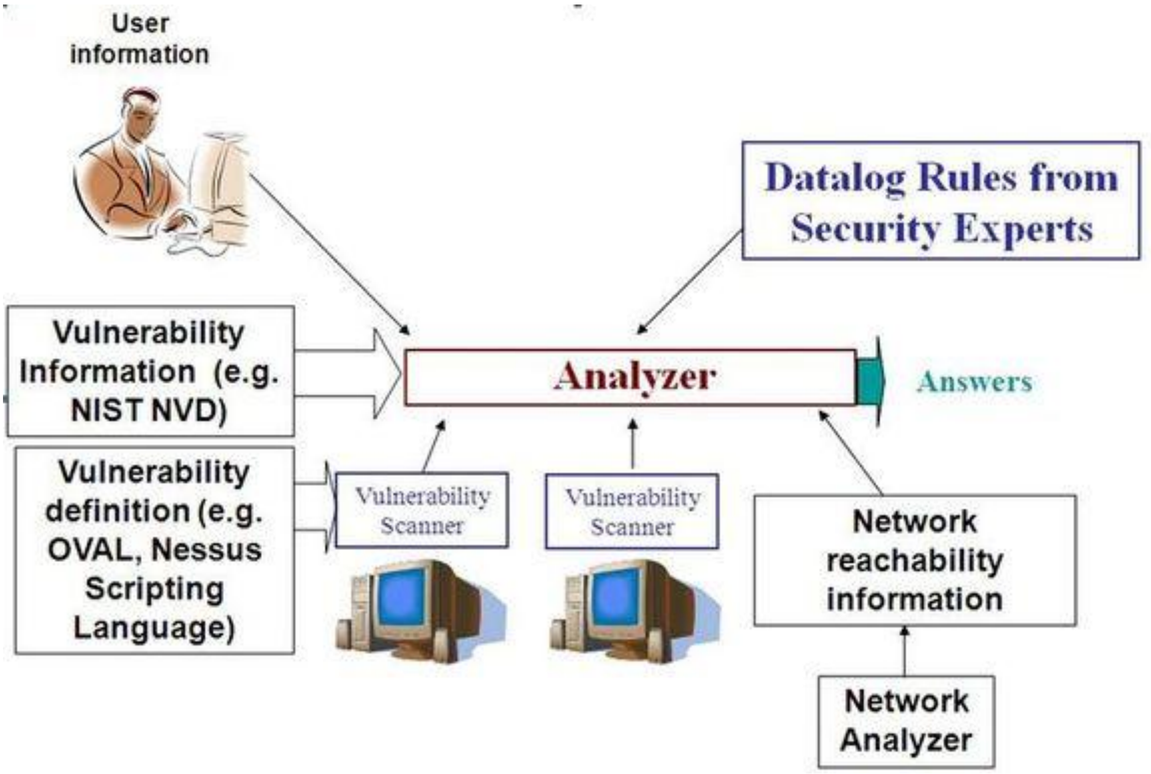
MulVAL Attack Paths Engine

To determine the security impact that software vulnerabilities have on a particular network, one must consider interactions among multiple network elements. For a vulnerability analysis tool to be useful in practice, the model used in the analysis must be able to automatically integrate formal vulnerability specifications from heterogeneous vulnerability sources.

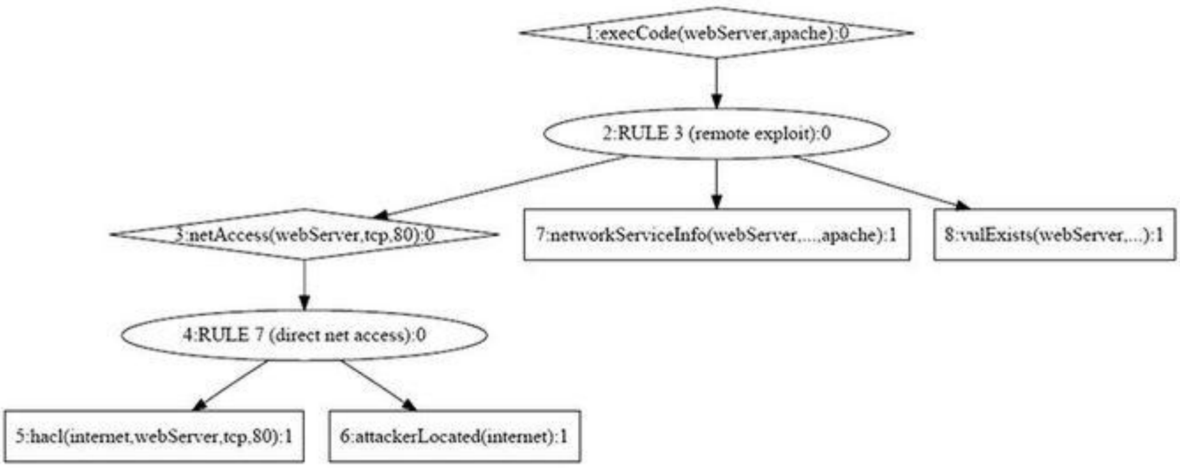
The MulVAL Attack Paths Engine is an end-to-end framework and reasoning system that conducts multihost, multistage vulnerability analysis on a network. The MulVAL Attack Paths Engine adopts Datalog (a query and rule language for deductive databases) as the modeling language for the elements in the analysis (bug specification, configuration description, reasoning rules, operating-system permission and privilege model, etc.). It has leveraged existing vulnerability-database and scanning tools by expressing their output in Datalog to feed the Attack Path Engine.

The inputs to the MulVAL Attack Paths Engine's analysis are:

- Advisories: What vulnerabilities have been reported and do they exist on my machines?
- Host configuration: What software and services are running on my hosts, and how are they configured?
- Network configuration: How are my network routers and firewalls configured?
- Principals: Who are the users of my network?
- Interaction: What is the model of how all these components interact?
- Policy: What accesses do I want to allow?



The current MulVAL Attack Paths Engine data model relies on the exploit range (local or remote) and the privilege escalation consequence data that are stored in NIST NVD. The figure below shows a logical attack graph computed by the MulVAL Attack Paths Engine.



The MulVAL Attack Paths Engine uses Datalog (a subset of Prolog) to produce logical attack graphs. It takes as input a set of first-order logical configuration predicates and produces the corresponding attack graph. These configuration predicates include network specific security policies, binding information and vulnerability data gathered from vulnerability databases. The MulVAL Attack Paths Engine identifies possible policy violations through logical inference.

Attack graph presents a qualitative view of security discrepancies:

- It shows what attacks are possible, but does not tell you how bad the problem is.
- It captures the interactions among all attack possibilities in your system.

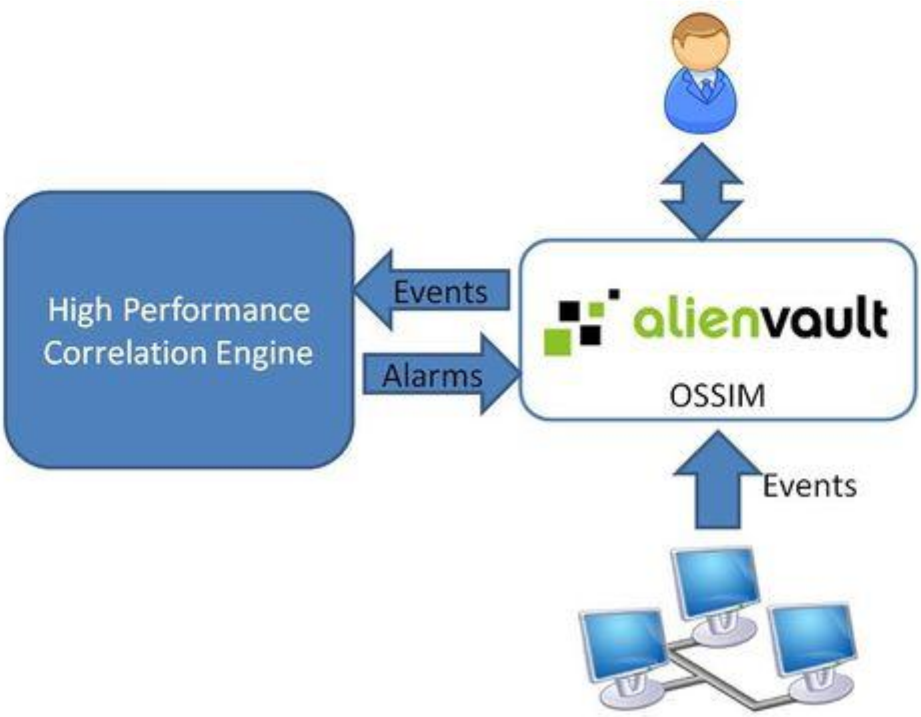
CVSS provides a quantitative property of individual vulnerabilities:

- It tells you how bad an individual vulnerability could be.
- But it does not tell you how bad it may be in your system.

The idea is to use CVSS to produce a component metric, i.e. a numeric measure on the conditional probability of success of an attack step. The MulVAL Attack Paths Engine aggregates the probabilities over the attack-graph structure to provide a cumulative metric, i.e. the probability of attacker success in your system. Suppose there is a “dedicated attacker” who will try all possible ways to attack your system. If one path fails, he will try another. The cumulative metric is the probability that he can succeed in at least one path.

Service Level SIEM

Limitations of current SIEM (Security Information and Event Management) systems are mainly in line with performance and scalability leading to the inability to process vast amounts of diverse data in a short amount of time. Next generation of SIEM solutions should overcome these performance limitations of its predecessors allowing in this way to monitor more systems, to process more complex rules or even to correlate events at different layers. To achieve the above commented goals, the SIEM to be included in FI-WARE will incorporate a high performance parallel correlation engine that will improve drastically the correlation capabilities of the current SIEM solutions available in the market. In the context of FI WARE this high performance correlation engine will be built on top of the OSSIM (Open Source Security Information Management - http://www.ossim.net) however integration with other tools, such as Prelude or Sentinel, could be analysed



High performance correlation engine

Botnet Tracking System

The NXDOMAIN-based Analysis focuses on the detection of «domain flux botnets», where the C&C domain names are frequently changed in order to escape from classical block-lists like the ones provided by DNSBL (Domain Name System Blacklists). This analysis relies on the observation of the behaviour of such botnets, and the way the bots try to locate their C&C servers. In order to find the domain name attached to the C&C server, the bots will request several domain names first, determined by more or less complex Domain Generating Algorithms (DGA): time-based, pseudorandom characters, dictionary based generation, etc. At a given time, such algorithms will generate a list of possible domain names to request, amongst them only one or few will be effectively reserved by the botmaster.

Because only few domain names are really associated to an IP address, bots will generate several DNS requests - answered by DNS errors - until finding an active domain. The target of the proposed solution is to detect abnormal error rates in order to identify and track the underlying botnet. Advantages of such approach is that:

- The DNS error traffic represents only a small portion of the whole DNS flow, thus ensuring a better scalability of our approach, a faster detection and a “less intrusive” analysis for the end-users.
- The DNS errors present a very limited meaning by themselves. Such analysis would not allow users’ profiling, and limit in that way the privacy impact.
- The DNS error traffic presents a very high dispersion compared to the successful traffic. As for example, there will be a huge number of users doing requests to www.orange.com, while the probability for a user to request the non-existing domain whzejdqmvnt.dynserv.com is very low. Such characteristic makes DNS errors traffic easier to analyse in order to detect abnormal behaviours.

Fuzzer

To allow IoT developers to assess the security of their applications, we developed a fuzzing tool for the 6LoWPAN protocol.

Fuzzing is a software testing technique, that involves providing valid, invalid, unexpected or random information as input of an application. Then, the program itself will react to these inputs, reporting exceptions or crashes, failing in the normal behavior or keeping the normal flow. The technique goal is to take advantage of the low-cost computation power to find out unexpected scenarios that lead to situations that escape from the normal flow scenario and produce an unexpected behavior.

In our current case, the target application is either the protocol itself or the application that resides on a remote device; so, in this case, we will be sending messages through the network to test the target device's behavior.

6LoWPAN is the acronym of IPv6 over Low power Wireless Personal Area Networks, also it’s the name of the IETF working group in charge of the protocol.

The 6LoWPAN group has defined an encapsulation and header compression mechanism that defines a set of compression/decompression rules taking advantage of the most common messages sent through a typical wireless sensor network, and exploiting some features of the underlaying layer, IEEE 802.15.4, and the upper layer, IPv6. Furthermore, duplication of information is avoided, allowing the protocol to send really short messages, and in this way, helps the devices saving energy. The protocol also defines some special rules to fragment long IPv6 messages, being able in this way, to send minimal length IPv6 message.

Note on interaction between the Fuzzer and the IoT Work Package

The Protocol Adapter GE provides an adaptation layer between the Gateway and any IoT device, for devices that include an IP stack and suppoort the CoAP protocol (from the IETF "CoRE" group), and the IoT Work Package concentrates on the application layer, and relies on existing standards for the lower network layers.

6LoWPAN & RPL are simply one of these standards, that allow to communicate with IoT devices using IPv6, and they are also being defined by the IETF (by the "6lowpan" and "roll" groups).

So, there is actually no conflict between this GE and the IoT Work Package, as they don't target the same layers.

The Fuzzer can be used as-is by Use Cases that decide to deploy devices that use the 6LoWPAN stack, and it can support any protocol for which a scapy module exists.

And in the event Use Cases decide to adopt other standards, and have an interest in the Fuzzer, we can also discuss the possibility for us to implement the necessary modules.

Countermeasures

The decision making support will look for the possible topological solutions and the software updates that could reduce / cut a given attack path. A remediation DB is needed; it will be built on publicly-available Security Advisories (for example, coming from CERT-EU); it will be external to the GE, as the Vulnerability DB. Only clear input will be considered, i.e. the advisories which are making a non ambiguous link between one or several CVE-ID and a software update. Remediation will be correlated to the vulnerabilities mentioned in the attack path. For each node, a software or a topological update will be proposed. As a result, a defence path will be produced, mirroring the submitted attack path.

Visualization framework

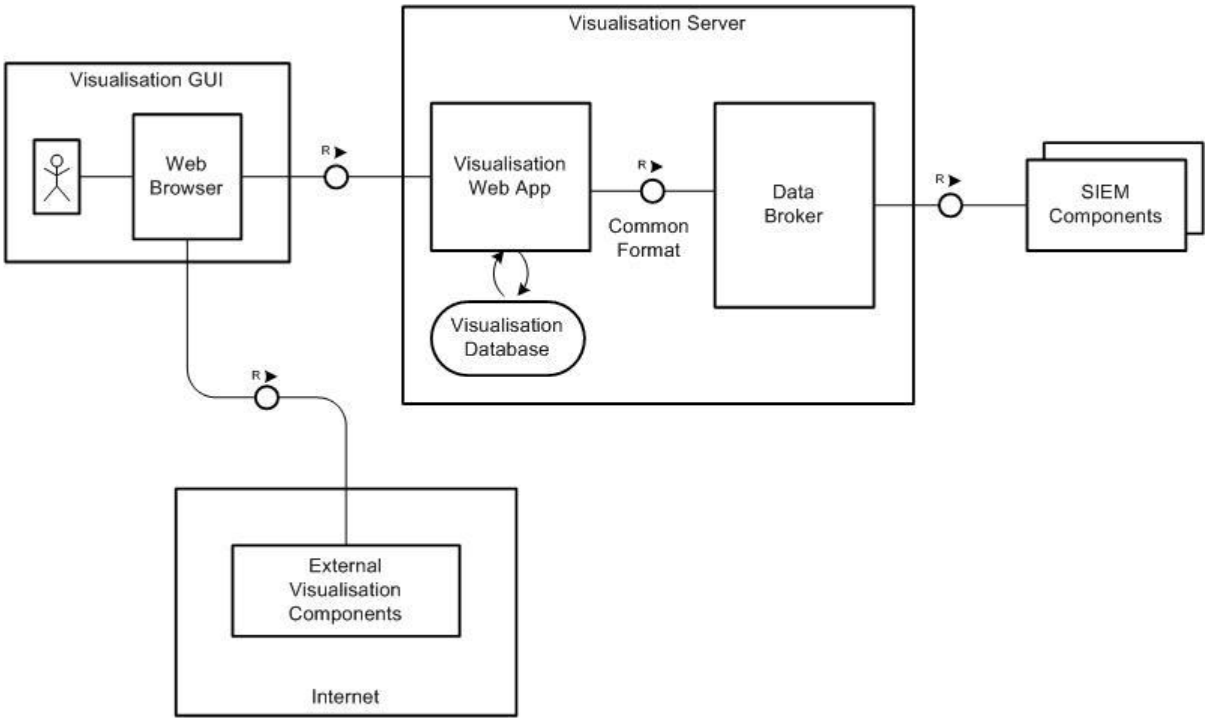
Systems that monitor the security of a network, such as network probes as part of an Intrusion Detection Systems (IDS), can generate a large amount of data. It is generally agreed that one of the most effective ways in which large volumes of data are presented to a human is by the use of visual analytics techniques. The INTERSECTION Visualisation Framework aims to enable large quantities of data to be presented to users in ways that aid their understanding of it. It is a flexible framework that allows the easy combination of multiple sources of data and enables the easy combination of third party and bespoke visualisations. The Visualisation Framework, business-oriented, allows the user to choose which data is visualised and which visualisation techniques are used.

In addition, it is extensible, allowing the addition of new data sources, data processing and visualisations. The design of the system is built around the concept of web-based mash-ups, which combine content from multiple sources into an integrated experience, and rich Internet applications (RIA) have a similar set of features and functionality to desktop applications.

The functionality of the Visualisation Framework can loosely be considered in two parts: the Data Broker, which collates and manages data; and the Visualisation Web Application, which provides and controls the visualisation. The Visualisation Framework's Data Broker interfaces with the various data sources to collect data. This can be achieved via a message queue, through accessing an external database or via some other means, depending on the source of this data. Data routes are created, as required, between the various data sources and end points. In all cases, the end points are adapters that transform incoming data into a common form, used throughout the visualisation component. Data is also stored in the visualisation database to allow a user to review historical as well as current data.

The Visualisation Web Application provides a number of key functions:

- Serves up pages to a user allowing them to set-up and interact with visualisations.
- Provides access to locally stored visualisations and facilitate the use of third party visualisations through the Internet.
- A conduit between data held on the server and the user who is accessing the visual analytics system from a web-browser.
- Allows the user to choose, configure and map data to visualisation axes as required.



36.1.3 Main Interactions

36.1.3.1 *MulVAL Attack Paths Engine*

The MulVAL Attack Paths Engine uses the report of vulnerability scanners. Scanner can run asynchronously on each host and which adapts existing tools such as OVAL to a great extent—and an analyzer, run on one host whenever new information arrives from the scanners.

An OVAL scanner takes such formalized vulnerability definitions and tests a machine for vulnerable software. The result is converted into Datalog clauses like the following:

vulExists(webServer, ’CAN-2002-0392’, httpd).

Namely, the scanner identified a vulnerability with CVE id CAN-2002-0392 on machine webServer. The vulnerability involved the server program httpd. However, the effect of the vulnerability—how it can be exploited and what is the consequence — is not formalized in OVAL. NVD the vulnerability database developed by the National Institute of Standards and Technology (NIST), provides the information about a vulnerability’s effect through CVSS Impact metrics. The relevant information is converted from CVSS into Datalog clauses such as:

vulProperty(’CAN-2002-0392’, remoteExploit,privilegeEscalation.

The MulVAL Attack Paths Engine models elements in Datalog. The model elements are recorded as Datalog facts. The MulVAL Attack Paths Engine requires all Datalog facts to be defined prior to performing any analysis. Missing or incorrect facts will result in a misleading analysis of the system being modelled. The following table shows the elements modelled by the MulVAL Attack Paths Engine and their Datalog fact statements sorted by the DAP layer in which they belong.

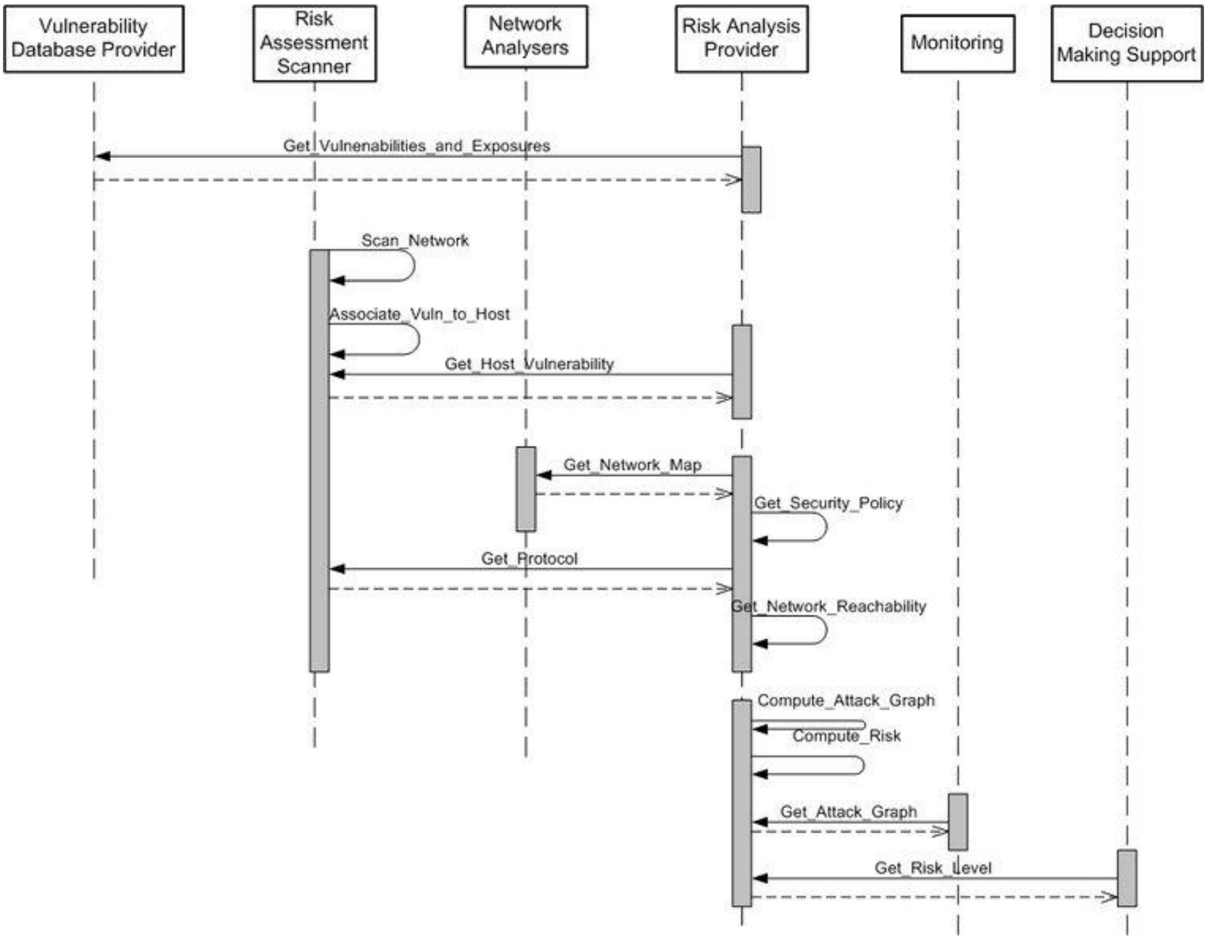
Attack Path Engine Model Element	Datalog Fact
Threat Agent Intention	malicious(Principal)
Services that run on Hosts	networkService(Host, Program, Protocol, Port, Account)
Vulnerabilities to Services running on Hosts	Vulnerabilities to Services running on Hosts
Client Programs that run on Hosts	clientProgram(Host, Program, RunAccount) setuidProgram(Host, Program, OwnerAccount)
clientProgram(Host, Program, RunAccount) setuidProgram(Host, Program, OwnerAccount)	vulProperty(CVE_id, ExploitRange, Consequence) ExploitRange = local or remote Consequence = confidentiality loss, integrity loss, denial of service, or privilege escalation
Starting location of attacks	Starting location of attacks
Hosts / Accounts on Hosts	Implied in other Datalog facts
Principals having Accounts on Hosts	Principals having Accounts on Hosts
Paths on Hosts	filePath(Host, Owner, Path)
Remotely mounted Paths on Hosts	nfsExport(Host, Path, Access, Client) nfsMounted(Client, ClientPath, Server, ServerPath)
Data residing in Paths on Hosts	dataBind(Data, Host, Path)
Access Control Rights on Paths on Hosts	
Policies where Principals can access Data	allow(Principal, Access, Data)
Connectivity between Hosts	hacl(Host, Host, Protocol, Port)

How the MulVAL Attack Paths Engine Datalog facts interrelate is recorded as Datalog reasoning rules that are shown in the following table.

Attack Trace Engine Model Element	Datalog Rule
Remote service exploitation resulting in privilege escalation u¯ ig vulnerable services.	execCode(Attacker, Host, Priv) :- vulExists(Host, CVE_id, Program), vulProperty(CVE_id, remoteExploit, privEscalation), networkService(Host, Program, Protocol, Port, Priv), netAccess(Attacker, Host, Protocol, Port), malicious(Attacker)
Remote client exploitation resulting in privilege escalation using vulnerable client programs	execCode(Attacker, Host, Priv) :- vulExists(Host, CVE_id, Program), vulProperty(CVE_id, remoteExploit, privEscalation), clientProgram(Host, Program, Priv), malicious(Attacker)
Local client exploitation resulting in privilege escalation using vulnerable client programs.	execCode(Attacker, Host, Owner) :- vulExists(Host, CVE_id, Program), vulProperty(CVE_id, localExploit, privEscalation), setuidProgram(Host, Program, Owner), execCode(Attacker, Host, SomePriv), malicious(Attacker)
Local user exploitation resulting in privilege escalation using Trojan programs	execCode(Attacker, Host, Owner) :- accessFile(Attacker, Host, write, Path), filePath(Host, Owner, Path), malicious(Attacker)
Local file access exploitation	accessFile(Principal, Host, Access, Path) :- execCode(Principal, Host, Owner), filePath(Host, Owner, Path)
Remote file access exploitation using NFS	accessFile(Principal, Host, Access, Path) :- malicious(Principal), execCode(Principal, Client, root), nfsExport(Server, Path, Access, Client), hacl(Client, Server, rpc, 100003)
Multi-hop network access	netAccess(Principal, TargetHost, Protocol, Port) :- execCode(Principal, InitiatingHost, Priv) hacl(InitiatingHost, TargetHost, Protocol, Port)
Policy Violations	policyViolation(Principal, Access, Data) :- access(Principal, Access, Data), not allow(Principal, Access, Data)

With the occurrence of new vulnerabilities, assessment of their security impact on the network is important in choosing the right countermeasures: patch and reboot, reconfigure a firewall, dismount a file-server partition, and so on.

The next figure show the sequence diagram with the interaction beetwen thhe others components:



Vulnerability Data Interface Description

-<definition class="vulnerability" id="oval:org.mitre.oval:def:99" version="4">

```
 -<metadata>
   -<title>
     IE v6.0 Content Disposition/Type Arbitrary Code Execution
    </title>
```

-<affected family="windows">

```
  <platform>Microsoft Windows 2000</platform>
  <product>Microsoft Internet Explorer</product>
```

</affected> <reference ref_id="CVE-2002-0193" ref_url="http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0193" source="CVE"/> -<description>

```
  Microsoft Internet Explorer 5.01 and 6.0 allow remote attackers to
execute arbitrary code via malformed Content-Disposition and
Content-Type header fields that cause the application for the
spoofed file type to pass the file back to the operating system for
handling rather than raise an error message, aka the first variant
of the "Content Disposition" vulnerability.
</description><
```

-oval_repository>

```
 -<dates>
   -<submitted date="2004-01-27T05:00:00.000-04:00">
      <contributor organization="The MITRE Corporation">Andrew
Buttner</contributor>
    </submitted>
   -<modified comment="modified wrt-222 - changed pattern match"
date="2005-03-07T05:00:00.000-04:00">
     <contributor organization="The MITRE Corporation">Christine
Walzer</contributor>
    </modified>
    <status_change date="2005-03-09T05:00:00.000-
04:00">INTERIM</status_change>
    <status_change date="2005-03-29T05:00:00.000-
04:00">ACCEPTED</status_change>
   -<modified comment="Changed IE registry test to wrt-18"
date="2005-09-20T04:00:00.000-04:00">
       <contributor organization="The MITRE Corporation">Christine
Walzer</contributor>
    </modified>
    <status_change date="2005-09-21T01:27:00.000-
04:00">INTERIM</status_change>
    <status_change date="2005-10-12T05:49:00.000-
04:00">ACCEPTED</status_change>
   -<modified comment="Added negate=true attribute to criteria sub-
block to fix conversion error from OVAL 4.2 to OVAL 5.0" date="2006-
07-03T12:56:00.000-04:00">
     <contributor organization="The MITRE Corporation">Matthew
Wojcik</contributor>
   </modified>
   <status_change date="2006-07-03T12:56:00.000-
04:00">INTERIM</status_change>
   <status_change date="2006-09-27T12:29:41.221-
04:00">ACCEPTED</status_change>
```

```
  -<modified comment="Multiple corrections and update to POSIX compatibility for ste:2878" date="2010-11-29T16:13:00.904-05:00">

     <contributor organization="G2, Inc.">Shane Shaffer</contributor>

   </modified>

   <status_change date="2010-11-29T16:14:04.414-05:00">INTERIM</status_change>

    </dates><status>INTERIM</status></oval_repository></metadata><criteria comment="Software section" operator="AND"><criterion comment="the version of mshtml.dll is less than 6.0.2716.2200" negate="false" test_ref="oval:org.mitre.oval:tst:3086"/><criterion comment="the patch q321232 is installed (Installed Components key)" negate="true" test_ref="oval:org.mitre.oval:tst:3119"/><criterion comment="the patch q323759 is installed (Installed Components key)" negate="true" test_ref="oval:org.mitre.oval:tst:3118"/><criterion comment="the patch q328970 is installed (Installed Components key)" negate="true" test_ref="oval:org.mitre.oval:tst:3117"/><criterion comment="the patch q324929 is installed (Installed Components key)" negate="true" test_ref="oval:org.mitre.oval:tst:3116"/><criterion comment="the patch q810847 is installed (Installed Components key)" negate="true" test_ref="oval:org.mitre.oval:tst:3115"/><criterion comment="the patch q813489 is installed (Installed Components key)" negate="true" test_ref="oval:org.mitre.oval:tst:3114"/><criterion comment="the patch q818529 is installed (Installed Components key)" negate="true" test_ref="oval:org.mitre.oval:tst:3113"/><criterion comment="the patch q822925 is installed (Installed Components key)" negate="true" test_ref="oval:org.mitre.oval:tst:3112"/><criterion comment="the patch q828750 is installed (Installed Components key)" negate="true" test_ref="oval:org.mitre.oval:tst:3111"/><criterion comment="the patch q824145 is installed (Installed Components key)" negate="true" test_ref="oval:org.mitre.oval:tst:3110"/><criteria comment="Windows 2000 Service Pack 4 (or later) is installed" negate="true" operator="AND"><criterion comment="Windows 2000 is installed" negate="false" test_ref="oval:org.mitre.oval:tst:3085"/><criterion comment="SP4 or later Installed" negate="false" test_ref="oval:org.mitre.oval:tst:3073"/></criteria><criterion comment="Internet Explorer 6 is installed" negate="false" test_ref="oval:org.mitre.oval:tst:3090"/></criteria>

>
```


Topological Data Network Interface Description

```
<

 The reachability input is like "hacl(HOST1, HOST2, Protocol, Port)", where "hacl" means "host access control list".

>
```

Service Level SIEM

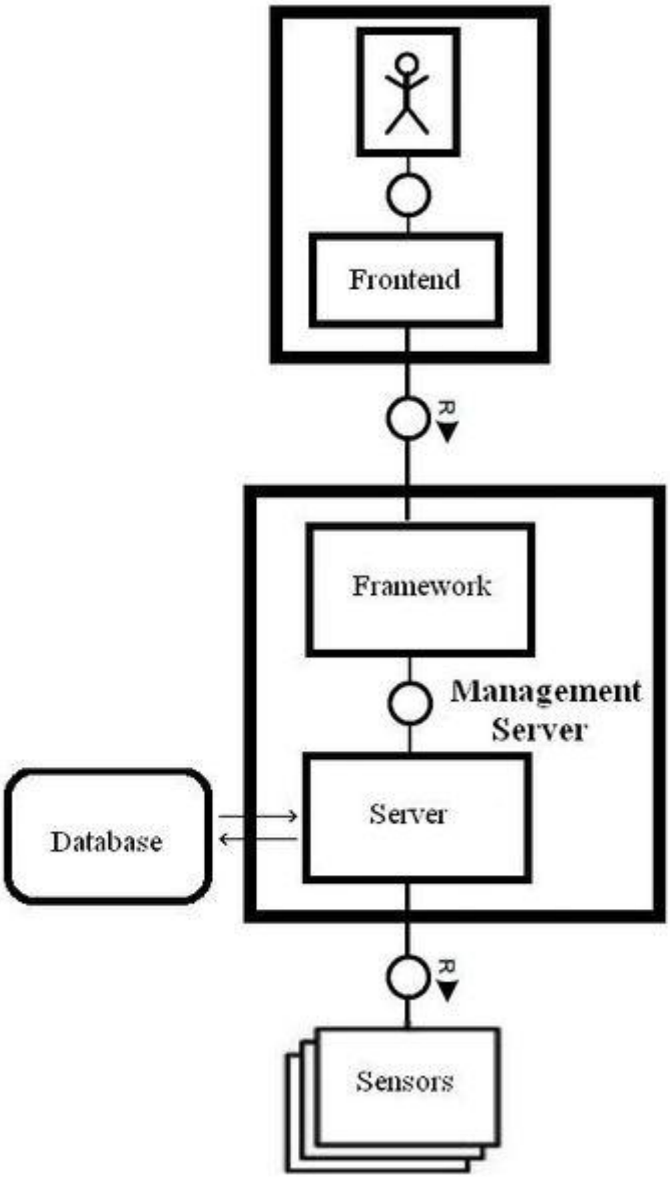
A conventional SIEM deployment is mainly composed of four elements:

1.Sensors: deployed in the networks to monitor network activity. They usually include the low level detectors and monitors that passively collect data looking for patterns but also, they can include active scanners that try to compile information about node vulnerabilities or agents which could receive data from other hosts of this network.

2. Management Server: this component is in charge of the main processing activities such as normalizing, prioritizing, collecting, risk assessment and correlating engines.

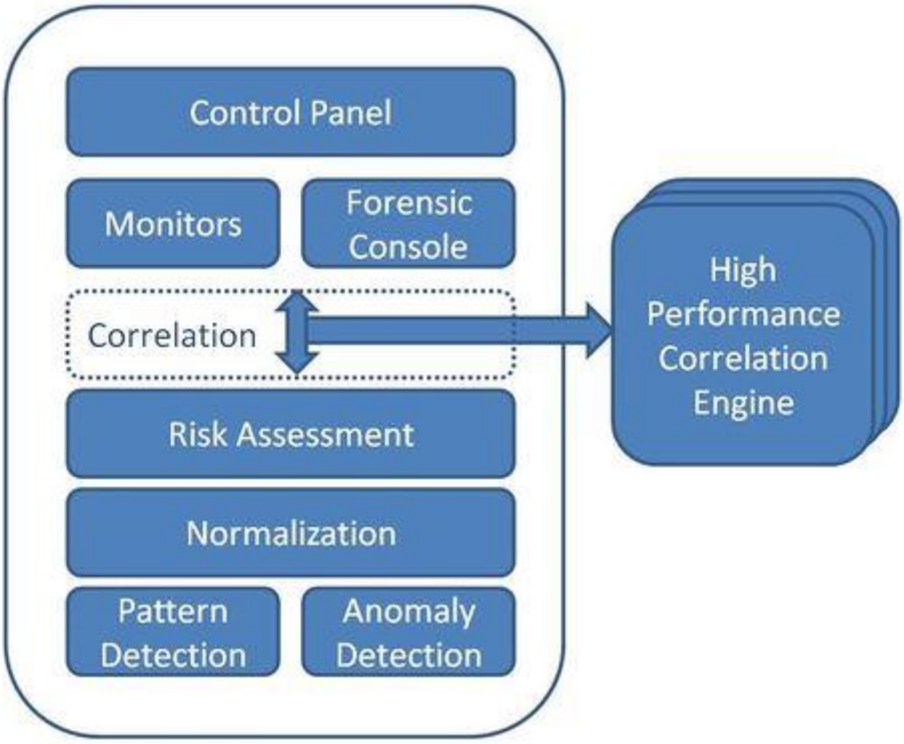
3. Database: where all events and information configuration for the management of the system is stored.

4. Fronted: where the operator can visualize the status of the system and configure the SIEM.



SIEM main elements

From a functional point of view, the OSSIM SIEM stack could be illustrated as showed in the next figure, where also the bypass of the OSSIM correlation engine is depicted.



OSSIM SIEM functional view

Botnet Tracking System

Before the analysis we need to anonymize the IP addresses of the clients in order to preserve their privacy using a reversible hash function. Because some errors can be directly associated to miscon figured softwares, a first step is to filter the error traffic using the following criteria:

- Only the DNS domain names longer than 6 characters are proceeded, as short domain names have been exhausted since a while by generic web sites and cannot therefore be used for

domain flux;

- All the requests made on non-existing Top Level Domain (TLD) like '.home' and '.local' (mostly linked to Apple Bonjour protocol) and '.arpa' (reverse lookup which is rarely implemented) are discard ;

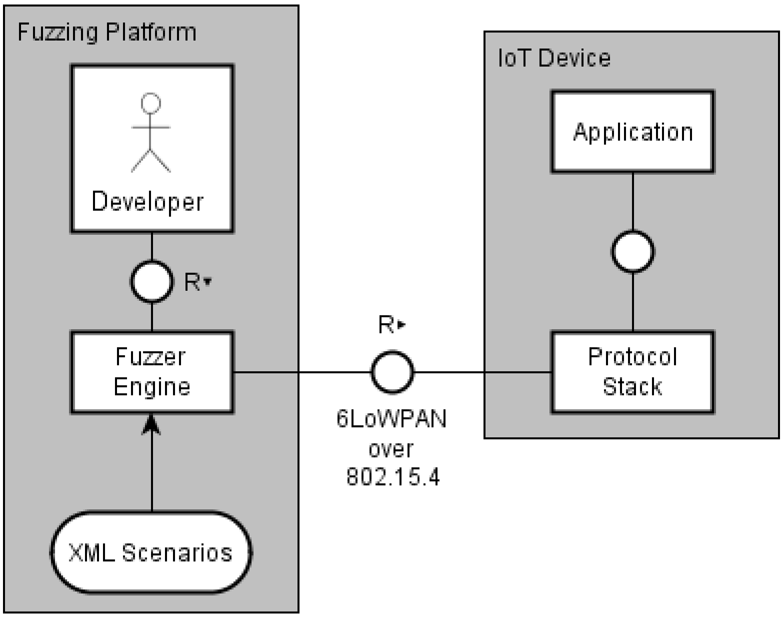
representing the 3rd most popular TLD on the L root server. Such filters are therefore useful more for performance reasons than for algorithm issues. Once the NX error traffic is expurgated from those generic errors, we build up a bipartite graph establishing the relations between failed queries of non existing domains and clients. Such graph allows us to identify communities of users with strong connectivity, i.e. doing similar errors in a short time frame. A cyclic analysis (every 60 seconds) is then made on the identified sub-graphs in order to compute a Malware Probability Factor (MPF) for each erroneous domain.

Fuzzer

The fuzzer communicates with IoT devices, through a 802.15.4 network interface on the fuzzing platform which must be in range of the devices, and must be capable of relaying raw Link Layer frames.

The fuzzer is driven by XML scenarios that define the sequence of packets to be sent, and how the fuzzed system should reply to these packets.

The developer interacting with the fuzzer can then use the provided fuzzing policies, or create new ones, and define how to apply them to the selected scenario, and start fuzzing the device.



Countermeasures

The decision making support will provide the following different interfaces:

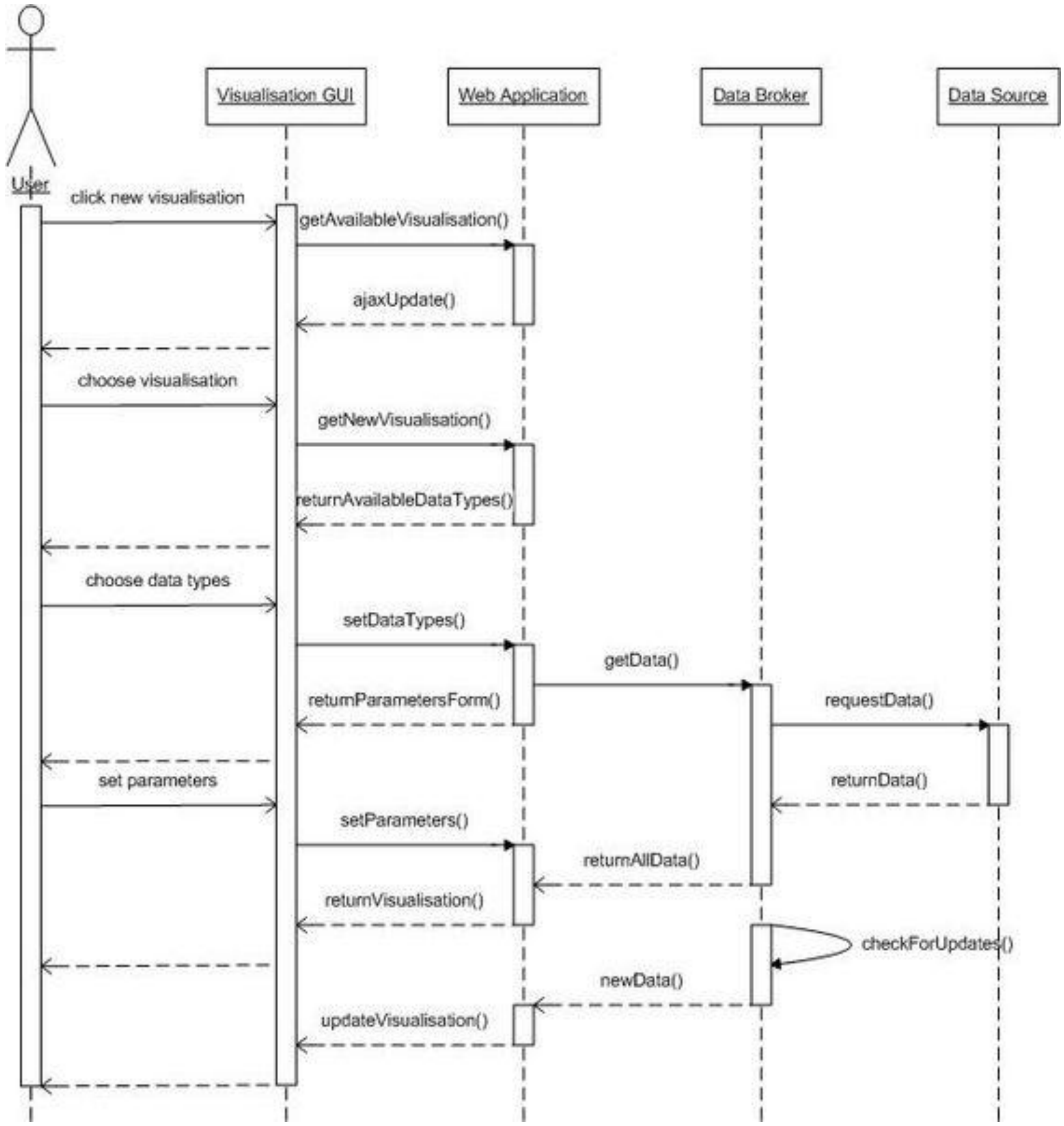
- An internal interface with the ‘Attack Path Engine’ to receive the attack path to be reduced.
- An external interface to send the countermeasure selected by the security operator to the monitored system / services.
- An external interface to collect the Security Advisories
- A GUI for the previously mentioned selection.

Visualization framework

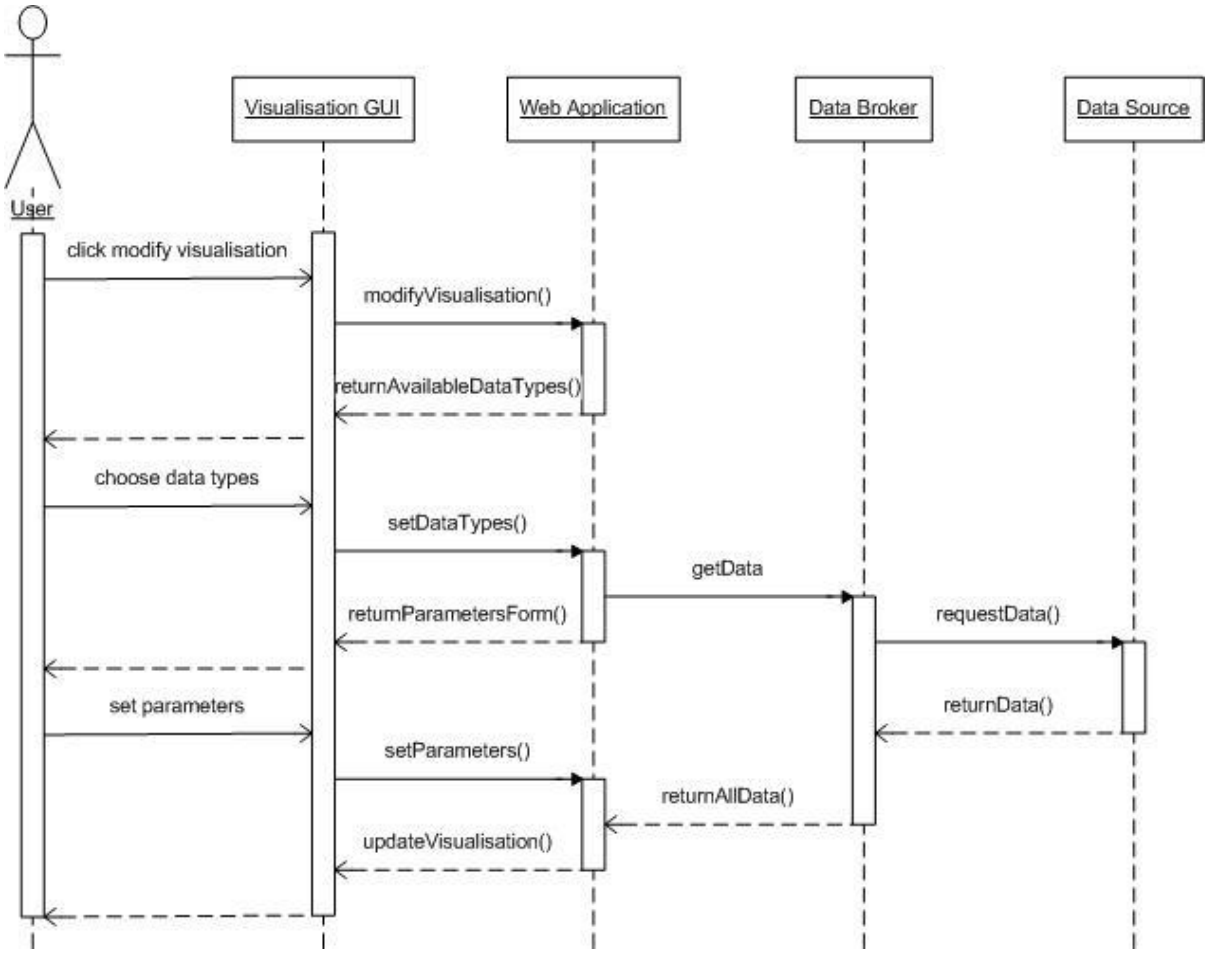
The Visualisation Framework offers a visualisation service that allows users to visualise data from multiple network components. The user accesses the visualisation service through a standard web-browser connected to the web-application server using some network connection (such as the Internet). The user will experience a single integrated application showing multiple visualisations. Behind the scenes, the browser will compliment the information from the visualisation server with data and functionality directly from the Internet.

Users of the framework will follow a similar pattern of creating, interacting with, modifying and eventually removing visualisations. There are therefore the three main interactions between users and the Visualisation Framework: adding a new visualisation; modifying an existing visualisation; removing a visualisation.

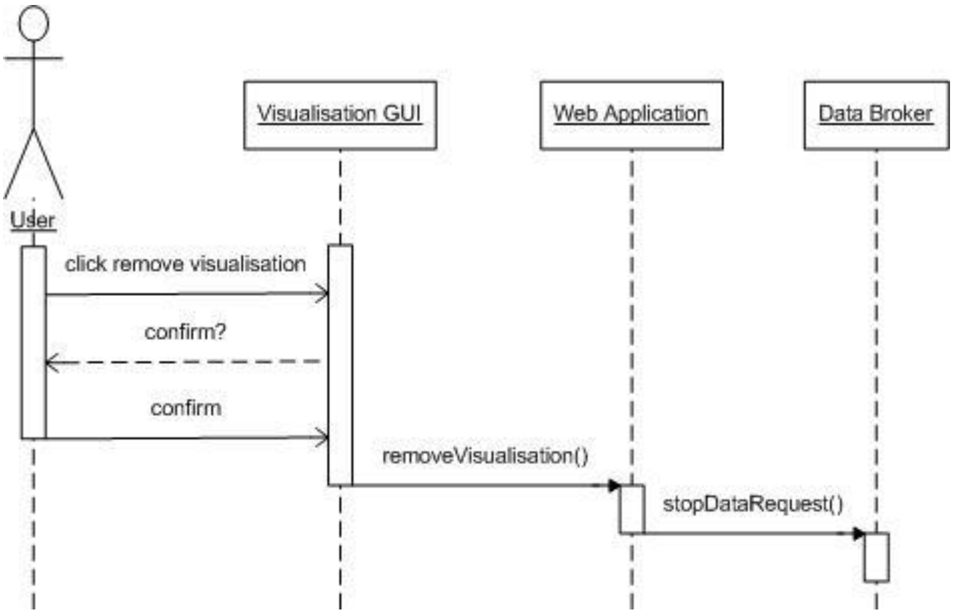
Add new visualisation enables a user to view a new visualisation. The user selects the visualisation and data type from a list of available options. External visualisations that support the existing data formats can also be added. The user can customise the visualisation, e.g. by choosing the size of the window. A sequence diagram for the interaction is shown in Figure 2.



Modify visualisation enables a user to modify an existing visualisation. The user can change the type of data displayed, the size of the window, how often the visualisation is updated. The interactions for modifying a visualisation are shown in the sequence diagram in Figure 3.



Remove visualisation enables a user to remove a window containing a visualisation from the display. A sequence diagram for the interaction is shown in Figure 4.



36.1.4 Basic Design Principles

Attack Path Engine

To determine the security impact software vulnerabilities have on a FIWARE architecture instantiation, one must consider interactions among multiple network components. The model used in the vulnerability analysis is able to automatically integrate formal vulnerability specifications from the bug-reporting community. But also from various vulnerability databases, specific to the cloud hosting, the internet of things, I2N.. Also, the analysis is able to scale to networks with thousands of machines.

To achieve these two goals, the Attack path Engine, composed of an end-to-end framework and a reasoning system, conducts multihost, multistage vulnerability analysis on a FIWARE architecture. The Attack Path Engine adopts Datalog as the modeling language for the elements in the analysis (bug specification, configuration description, reasoning rules, operating-system permission and privilege model, etc.). It easily leverage existing vulnerability-database and scanning tools by expressing their output in Datalog and feeding it to the Attack Path reasoning engine.

The reasoning engine consists of a collection of Datalog rules that captures the operating system behavior and the interaction of various components in the network. Thus integrating information from the bug-reporting community and off-the-shelf scanning tools in the reasoning model is straightforward. Reasoning rules specify semantics of different kinds of exploits, compromise propagation, and multihop network access. The rules are carefully designed so that information about specific vulnerabilities are factored out into the data generated from OVAL (Open Vulnerability and Assessment Language-MITRE)and ICAT (Categorization of Attacks Toolkit-NIST). The interaction rules characterize general attack methodologies (such as “Trojan Horse client program”), not specific vulnerabilities. Thus the rules do not need to be changed frequently, even if new vulnerabilities are reported frequently.

The Attack path Engine uses an exploit dependency graph to represent the pre and post conditions for exploits. Then a graph search algorithm can “string” individual exploits and find attack paths involves multiple vulnerabilities. This algorithm is adopted in Topological Vulnerability Analysis (TVA), a framework that combines an exploit knowledge base with a remote network vulnerability scanner to analyze exploit sequences leading to attack goals. Compared with a graph data structure, Datalog provides a declarative specification for the reasoning logic, making it easier to review and augment the reasoning engine when necessary.

The reasoning engine scales well with the size of the network. Once all the information is collected, the analysis can be performed in seconds for networks with thousands of machines.

Service Level SIEM

The OSSIM agent will receives normalized event.

The fields of which the standardised event consists are:

- type: Type of event, Detector or Monitor.
- date: date on which the event is received from the device.
- sensor: IP address of the sensor generating the event
- plugin_id: Identifier of the type of event generated
- plugin_sid: Class of event within the type specified in plugin_id

- priority: Possible deprecated (agent can't decide priority, just server)
- protocol: Three types of protocol are permitted in these events. Should a different one reach the server, the event will be rejected:: TCP, UDP or ICMP
- src_ip: IP which the device generating the original event identifies as the source of this event
- src_port: Source port
- dst_ip: IP which the device generating the original event identifies as the destination of this event
- dst_port: Destination port
- log: Event data that the specific plugin considers as part of the log and which is not accommodated in the other fields. Due to the Userdata* fields, it is used increasingly less.
- data: Normally stores the event payload, although the plugin may use this field for anything else
- username: User who has generated the event or user with whom it is identifying, mainly used in HIDS events
- password: Password used in an event
- filename: File used in an event, mainly used in HIDS events
- userdata1: These fields can be defined by the user from the plugin. They can contain any alphanumeric information, and on choosing one or another, the type of display they have in the event viewer will change. Up to 9 fields can be defined for each plugin.
- userdata2
- …
- userdata9

Fuzzer

The fuzzer engine itself is built around the Scapy packet manipulation framework, to which we have added a module that supports the assembly, disassembly and fuzzing of the 6LoWPAN protocol.

To be able to inject crafted 6LowPAN packets onto the network, we use an Atmel RZUSBstick, a 802.15.4 USB dongle, running a modified version of the Contiki OS, that simply relays packets without altering them.

The fuzzer engine is then driven by a scenario written in XML that defines a "normal" sequence of sent and received packets, as well as callback functions written in python that define the fuzzing process — which messages and message fields to alter, and how.

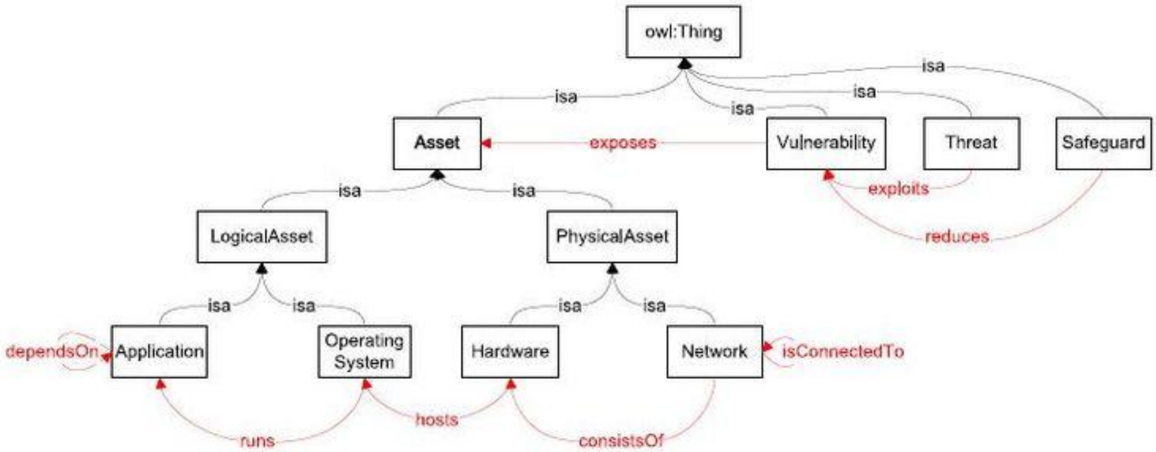
Countermeasures

The design of the decision making support module will be based on the Ontology Handler asset which has already the capacity to handle remediation (called ‘Safeguard’ in the currently defined ontology). It will coupled with the following capacities:

- a remediation extractor that will find the remediation information in the Security Advisories and will populate the ontology with it

- a defence path engine that will analyze the submitted attack path and will build the solution / countermeasure

The Ontology Handler is in charge of providing the Decision making support with knowledge relative to information systems infrastructure and security information, which are retrieved from the vulnerabilities, by means of an ontology. These two components can be replaced by other components that respect the interfaces of the Ontology Handler. In addition, the instantiated ontologies provided by the Ontology Handler can be directly used by the numerous inference engines to infer knowledge and security concerns.



The ontology repository exposes interfaces to create, access, manipulate and store instances of the Security Ontology. A. Security Ontology in computer science and information science, an ontology is a formal representation of a set of hierarchical concepts within a domain and the relationships between those concepts. It provides a vocabulary of classes and relationships to describe a domain. Ontologies are used to reason about the properties of their domain and to infer knowledge applying rules.

The proposed Security Ontology has been designed to describe the interdependencies between information systems' assets and their identified security issues according to the definitions of the ISO/IEC 13335-1:2004 standard. In this standard, vulnerabilities are considered as a property of a network security system. Assets and components have weak points named vulnerabilities. These vulnerabilities can be exploited by threats, that is, potential causes of unwanted incident which may result in harm to a system or organization, leading to attacks.

The proposed Security Ontology has been partially conceived with these concepts. We used OWL, the Web Ontology Language, endorsed by the W3C. Well-known in Semantic Web, this knowledge representation language utilizes a semantic model intended to provide compatibility with RDF (Resource Description Framework) Schema. OWL ontologies are most commonly serialized using RDF/XML (Extensible Markup Language) syntax. In addition, we integrated the Semantic Web Rule Language (SWRL), which extends OWL with inference rule extensions, within the OWL INSPIRE Security Ontology. An example of an inference rule would be to classify the assets according to their threats: AttackPatternClassification :

- Asset(?x) ^ exposes(?y; ?x) ^ exploits(?z; ?y) □! related to(?x; ?z)

Such a rule can be applied on every instance of the Security Ontology available in the Ontology repository.

The Ontology repository is the key component of the Ontology Handler. It serves as a knowledge base to store information relative to systems infrastructure and their associated security data using the Security Ontology. Each ontology instance corresponds to a version of an information system and its associated vulnerabilities, threats, and safeguards. These security data are retrieved from the vulnerability DB. Finally, the Ontology repository embeds mechanisms to create, save and retrieve instances of the Security Ontology and mechanisms to add, delete, and rename individuals and to set links between the individuals with predefined relationships. These functionalities are exposed in an interface in order for other components to remotely manipulate ontologies. The Ontology repository has been implemented in Java with the Jena library. The APIs of this component were defined in accordance with the principles of the OSS/J specification: they are available remotely by means of a Web Service.

Visualization framework

- Decoupling of data from visualisations. A fundamental principle is that the data is stored in common formats to allow any visualisation to work with the data and to allow each visualisation to work with multiple different data sets. This requires input data to be formatted in a suitable way to allow this (e.g. identification and location of common fields with other data). New data types can be accommodated, perhaps requiring some minor modifications to the Service. However, highly structured data, and particularly XML formats are preferred.
- Real-time input. The Service is designed to receive data in real time, and is particularly suited for publish-subscribe architectures.

36.1.5 Open specifications

Following is a list of Open Specifications linked to this Generic Enabler. Specifications labeled as "PRELIMINARY" are considered stable but subject to minor changes derived from lessons learned during last interactions of the development of a first reference implementation planned for the current Major Release of FI-WARE. Specifications labeled as "DRAFT" are planned for future Major Releases of FI-WARE but they are provided for the sake of future users.

36.1.5.1 *Open API Specifications*

Security Monitoring Generic Enabler APIs are under construction. The following initial functionalities will be available in September 2012:

- Attack graph engine conducting multi-host, multistage vulnerability analysis on a network and showing what attacks are possible. This functionality is included in the Risk Analysis Service
    - Security-Monitoring Mulval Attack Path Engine API Specification (PRELIMINARY)
- Security Information and Event Management tool based on a set of defined assets such as hosts, networks, groups and services.This functionality is included in the Event Correlation Service.
    - Security.Security-Monitoring Service Level SIEM API Specification (PRELIMINARY)

37 FIWARE ArchitectureDescription Security Context-based security & compliance

You can find the content of this chapter as well in the wiki of fi-ware.

***[DISCLAIMER:]** The first version of the generic enabler described below is planned to be delivered on second FI-WARE release.*

That is the reason because some of its specifications are still under discussion and has not been fixed yet.

Anyway the majority of the sections bellow are expected to have either minor or no changes at all.

The sections that are still no not mature enough are clearly identified and will be more detailed on release two.

37.1.1 Overview

37.1.1.1 *Introduction to the Generic Enabler*

The role of this Generic Enabler is support additional security requirements requested by a specific subset of applications as a result of the application of very specific regulatory constraints.

This feature provides the security layer of FI-WARE context-aware capabilities that allows any of its instances to deal with dynamic and unpredictable context changes.

End User applications will send to the GE their requirements and the context description to obtain the security solution that best fulfill them.

Once the GE has gotten the user request and the applicable rules it will query FI-WARE security marketplace to get the Optional Generic Enabler that best fulfill the security requirements defined and will deploy it in the End-User context.

At the same time that the security solution is deployed into the end-user environment the GE also instantiates a runtime monitor with the responsibility of detecting anomalous behavior or non-conformance.

In case of a non-conformance detected the framework will take compensation actions for the automated adaptation of the deployed security mechanism to the changing context conditions

The security requirement sent by the end-user applications to the GE could be expressed both by single security specifications and by references to a predefined rule, law or agreement stored in the rule repository offered by the GE.

Security services must be registered in FI-WARE market place and must describe their features by using USDL-SEC language to be successfully discovered by this GE.

37.1.1.2 *Example Scenario*

A firm has implemented some communication links between its subsidiaries by using FI-WARE capabilities.

As employees private data are shared and each of the subsidiaries is located in a different country; the communications links must be compliant with different Data Protection regulations.

Context-based security & compliance Generic Enabler is requested to deploy a security enabler that will guarantee the privacy and make the communication link compliance with the Data Protection Law of each country.

In the case of one of the affected countries would change its applicable regulations; the associated metrics will be updated in the GE rules repository by the firm.

Then the GE monitors will verify if the communication links with that specific country are still compliant with the new regulations and will triggered to the GE framework manager if needed.

Finally the optional security enabler will be either reconfigured or redeployed by the Context-based security & compliance Generic Enabler.

37.1.2 Basic Concepts

37.1.2.1 ***USDL-SEC***

USDL-SEC language is being developed as a security extension to the latest version of USDL language(Linked-USDL), currently maintained by the Applications/Services Ecosystem and Delivery Framework Work Package. More information on USDL can be found in its Open Specification page.

The available security service features as well as the rules to be fulfilled and the interfaces specification are described by using USDL-SEC language.

Furthermore the combination USDL/USDL-SEC describes a service along with functional and non-functional properties in a single and complete description file.

This characteristic provides means to compare and select services according to consumer needs.

The security extension allows:

- any application for expressing its security features in its associated properties file;
- consumers and providers to rely on a security protocol, through expressions of concrete mechanisms and links to existing standard such as WS-SecurityPolicy, XACML, P3P, etc.

37.1.2.2 ***Security Rules & specifications***

In the context of the GE we define these two concepts as follow

- **Security specification**: Any single security requirement that can be supported by a security service. Some examples could be encryption, authentication, and accountability
- **Rule**: A set or security specifications that describes a complex security agreement that must be fulfilled commonly by two (or more) entities. This rules will be depicted into our, to be defined, template with the support of the rule repository dashboard. Some examples could be a *Data Protection* law from a

country or a *security service level agreement* between two different companies

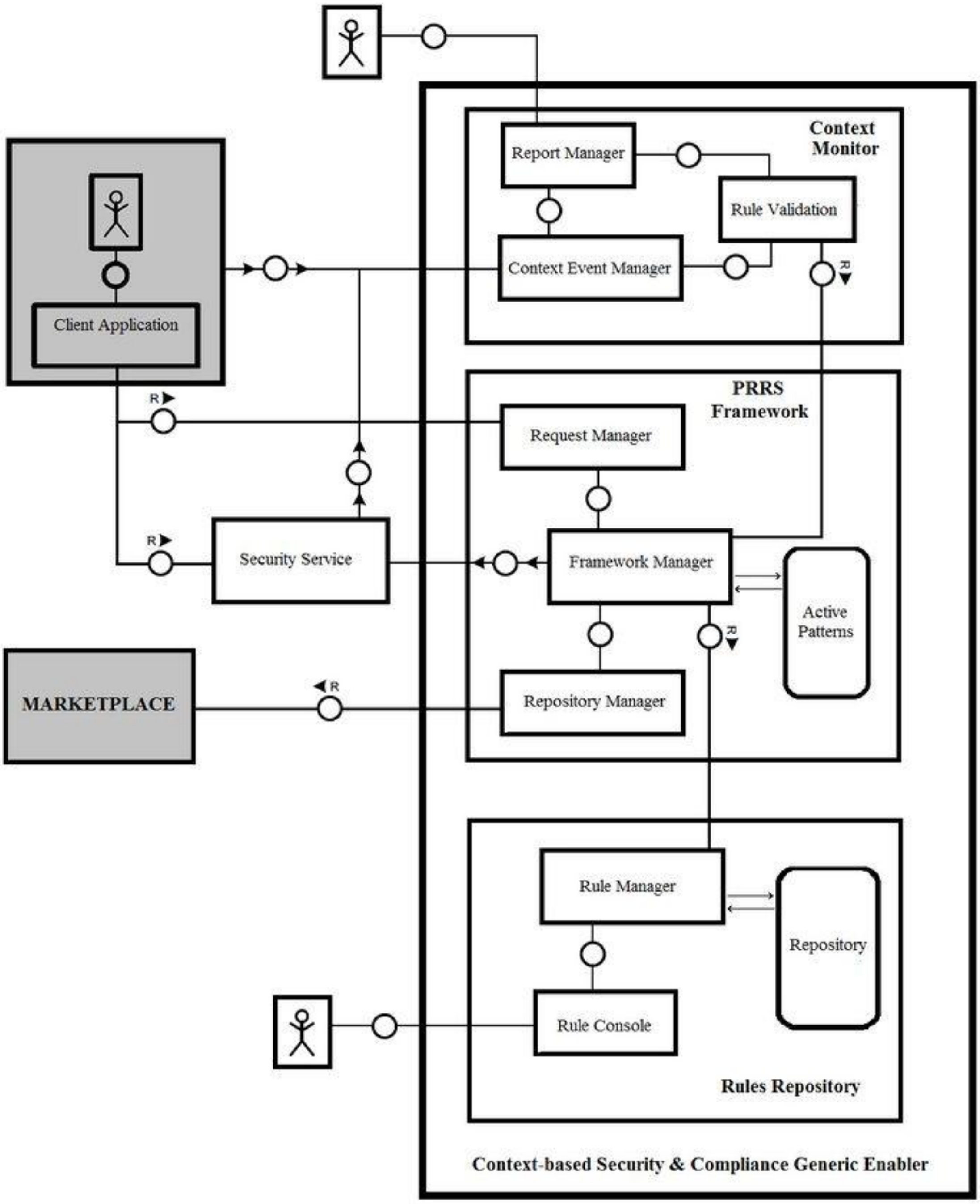
37.1.3 Context-based security & compliance architecture

This section describes software Context-based security & compliance GE modules. The overall architecture will be highlighted as well as the description of its main components.

Next chapter will briefly describe the interfaces that will be offered and implemented on second release.

The architecture of the proposed GE is detailed in the figure below.

The figure shows both the main component of the GE and the interfaces to be implemented between then and to be offered to the external applications.



Context-based Security and Compliance Generic Enabler Architecture

37.1.3.1 *PRRS Framework*

This component provides run-time support to applications performing Dynamic selection & deployment of security enablers.

PRRS framework is the core of the Generic Enabler. It is in charge of controlling the rest of the components of the GE, processing requests from end-user applications and orchestrating the instantiation of the Security Enabler selected.

End-user applications send requests in order to fulfill their security requirements to the PRRS Request Manager sub-module either as a specification of security requirements or as a reference to already existing rule.

PRRS Request Manager is also going to be in charge to send any notification from the GE to the End-User applications

PRRS Repository Manager is going to deal with the communication between PRRS component and Rule repository on one site and the FI-WARE Marketplace on the other

From Rule repository it retrieves the set of rules or security specification to be applied on each User Request and will be triggered by the repository in case of a rule change situation

From FI-WARE Market place it gets the best security solution to fulfill End-User request

Active patterns data base stores useful information related to the Applications request, monitoring systems that oversee them, the rules that are being used and the additional security enablers deployed for framework manager decision making support.

Finally PRRS Framework manager is the decision making engine. It compares rules to be applied, user requirement and security enabler features to select the most suitable solution to fulfill End-User requirements.

PRRS Framework manager also provides external monitoring components with the rules to be checked against the performance of the already deployed enabler and take the necessary recovery actions by the reactivation, reconfiguration, deactivation and/or substitution of the deployed enabler if required.

37.1.3.2 *Rule repository*

This component will allow the generic enabler to store and manage compliance requirements and relevant specifics at various abstractions levels and also check end-to-end business processes for compliance against the set of applicable constraints during design-time.

The rules to be stored could come from various sources, including laws and regulations, public and internal policies, standards, customer preferences, partner agreements and jurisdictional provisions.

In order to manage compliance throughout all phases of business process lifecycle, it must be possible to define and subsequently integrate compliance specifics into business processes and enterprise applications, and assure compliance starting from the process analysis and design phase.

Furthermore the component will be able to reuse fragments of already stored formal rules specifications to build a new formal specification form a new law or rule to be stored

Reuse of rules specification fragments will make the task of compile new laws or rules into formal language easier

Each high-level rule or specification will be compiled into a formal pattern following USDL-SEC specifications that can be applied and referenced in many scenarios either by end-user applications as a security requirement or any security enabler to describe its characteristics.

Finally Security requirements repository will be able to trigger PRRS framework when some rule will be modified so that the framework could take the necessary actions in case of the modification must be taken into account on compliance measurements

37.1.3.3 *System monitoring*

Runtime context monitors are the components in charge of detect anomalous behavior or non-conformances in End-user context environments.

Each Monitor component will get context and status events from the end user and the security enablers it is overseeing.

Then it will compare the information obtained with the rules provided by PRRS Framework.

In case of non-compliance detection the assigned event will be send to PRRS framework by the appointed monitor so that the framework could take the necessary recovering actions

Additionally Context monitor will provide a Dashboard that gives the system reporting capabilities.

Reports of system performance will be generated once the information from data context has been compared with rules received from PRRS

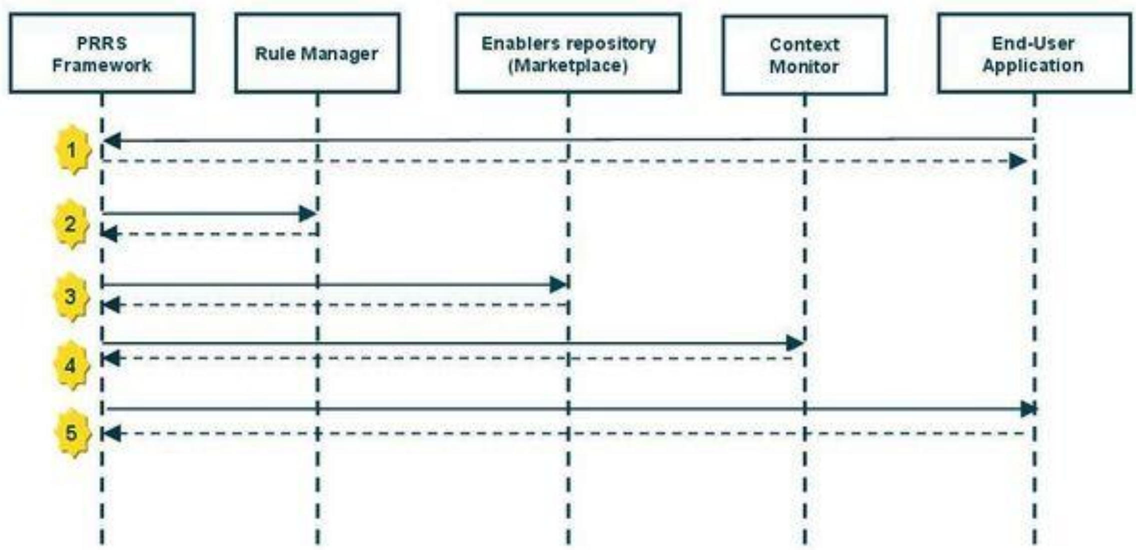
These visual reports will provide useful information about the levels of compliance, performance of the solutions dynamically deployed by PRRS and make the task of identification of root-causes for non-compliant situations easy.

37.1.4 Main Interactions

We describe in this section the first approach of the interface that is going to be offered by the GE and will be implemented on release two

37.1.4.1 *User Request*

This section describes the steps to be followed in a communication between an End-User application and the Generic Enable from user request of an additional security application to its deployment by the GE.



User Request Sequence Diagram

1: An End-User applications send a security request to the GE providing information about its security requirements and context. The GE stores the information into its internal data base and replies the applicant with the assigned service-ID.

End-User serviceRequest

Elements	Description
requestServiceName	Allows users to send a direct request to PRRS by service name if known
requestServiceType	User will specify type of service required
requestSpecFile	USDL-SEC compliant file rule or specifications description
requestContextDesc	XML file wih ditional and useful information about User context details

End-User serviceRequestResponse

Elements	Description
requestRespStatus	It will indicate the error level of the response. Where 0 equals to OK and any other value means an error
requestRespDetails	It will contain the error description if the previous parameter indicates an error or the asigned applicaion ID otherwise

2:The GE gets from the rule repository either the list of security specifications associated to the rule End-User application likes to fulfill or a set of rules that could satisfice the security specifications detailed by the End-User Application.

ruleRequest

Elements	Description
requestRuleName	Rule Identificator name if known
requestRuleFile	USDL-SEC compliant file rule or specifications description

ruleRequestResponse

Elements	Description
requestRespStatus	It will indicate the error level of the response. Where 0 equals to OK and any other value means an error
requestRuleName	Rule Identificator name
requestRuleFile	USDL-SEC compliant file rule or specifications description

3: The GE will select from FI-WARE marketplace the security enabler that best fulfill user requirements to be deployed into End-user context.

The communication to be implemented with the Marketplace will be compliant with the API description provided by Application and Services Ecosystem and Delivery Framework Work Package (WP3). Check for additional details.

4: A system monitor is instantiated and the validation rules are sent to it .

serviceMonitor

Elements	Description
requestUserMonitorID	User applicaion ID to be monitored
requestEventURL	Will be the URL where even to be monitored are located
requestRuleFile	Will content the deployed USDL-SEC compliant file with the rules to check

serviceMonitorResponse

Elements	Description
requestRespStatus	It will indicate the error level of the response. Where 0 equals to OK and any other value means an error
requestRespDetails	It will contain the error description if the previous parameter indicates an error

5: Selected Security service is instantiated and deployed. End-user applications gets the details to interact with it

PRRS serviceDeploy

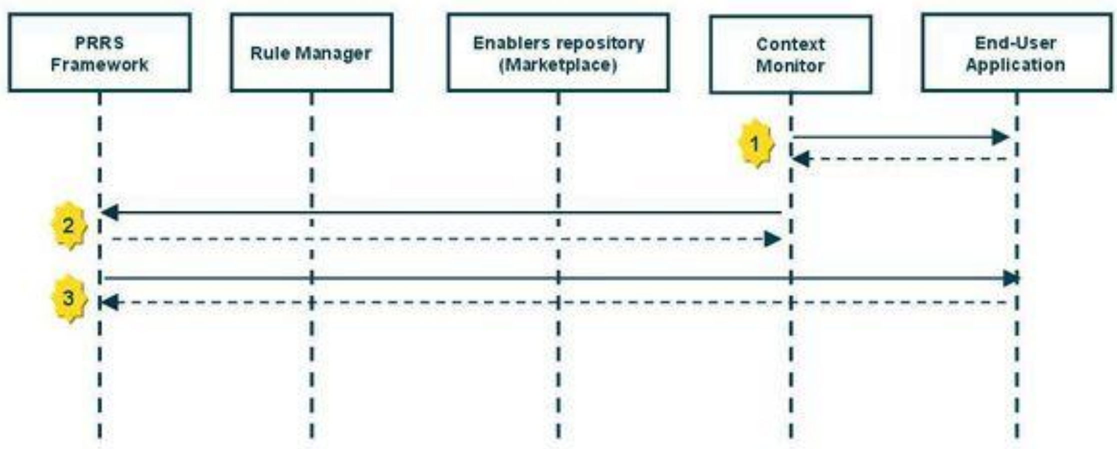
Elements	Description
requestUserID	Asigned applicaion ID
requestServiceURL	Will be the URL of the service assigned
requestRespFile	Will content the deployed USDL-SEC compliant file service description

PRRSserviceDeployResponse

Elements	Description
requestRespStatus	It will indicate the error level of the response. Where 0 equals to OK and any other value means an error

37.1.4.2 *Non Compliance Detection*

This section briefly summarizes the steps to be followed in an internal communication between a Monitor system & PRRS component to notify PRRS a non-compliance situation. **More details** about the interface to be implemented **will be available on release two**



Non Compliance Detection Sequence Diagram

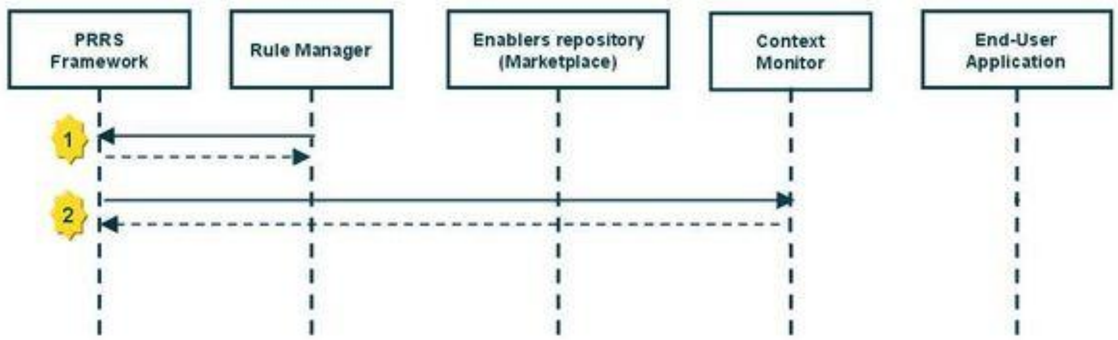
1: A System monitor retrieves information from End-User context both by receiving service even information and by accessing deployed security service logs.

2: The information retrieved on step one is compared with the validation rules and PRRS Framework is triggered in case of a non-compliance situation.

3: PRRS framework manager takes the most suitable recovery action by the reactivation, reconfiguration, deactivation and/or substitution of the deployed enabler.

37.1.4.3 *Rule Change*

This section briefly summarizes the steps to be followed in an internal communication between Rule repository & PRRS component to notify PRRS a rule change situation. **More details** about the interface to be implemented **will be available on release two**



Rule Change Sequence Diagram

1: Rule Manager triggers PRRS framework every time a stored rule is modified.

2: PRRS framework manager gets from its internal data base the system monitors that are overseen a context where the rule is applicable and updates them the associated validation parameters.

37.1.5 Basic Design principles

- Provide run-time security support for applications by dynamically deploying and monitoring the End-User applications environment
- Enhanced security and dependability by supporting automated integration, configuration, monitoring and adaptation
- Dynamic compliance of software services to business regulations and user requirements than can be easily modeled trough the rule repository dashboard
- As an extension (USDL-SEC) of the standard USDL language is going to be defined we share their main principles with it:
    - Uniform Service Descriptions
    - Modular Design
    - Extensibility Principle

38 FIWARE ArchitectureDescription Identity Management Generic Enabler

You can find the content of this chapter as well in the wiki of fi-ware.

38.1.1 Overview

On the one hand the ever-growing tsunami of today's shore-bound technologies can often overwhelm the user, significantly affecting his daily life. On a daily basis, he is forced to depend on his technological competence. The smooth running of his affairs depends on user's ability to handle a whole raft of often transient technologies. On account of very intensive, at times forced usage of the Internet and diverse services, the user encounters the need to transfer his "network-duties" to the networks as much as possible.

In other words, he seeks to find a convenient problem solver, which will allow him to cope easily and securely with services. Thus, the need arises for a clever composed Identity Management system, which will address the users' requirements.

Identity Management (IdM) encompasses a number of aspects involved with users' access to networks, services and applications, including secure and private authentication from users to devices, networks and services, Authorization & Trust management, User Profile management, Single Sign-On (SSO) to service domains and Identity Federation towards applications.

An IdM system is intended to undertake the complex task of handling, communicating with and coordinating between the slew of today's diverse technologies. Provide user-friendly technologies, putting the end user and his needs squarely at centre of the architecture (user-centric approach) whilst protecting the users' privacy.

On the other hand the computing resources are being actively exploited by the Enterprises lately through the use of cloudification and virtualization technologies. Nevertheless, with regard to such an evolution on the Web, the Enterprises still have to keep in mind the Identity Management issues and should be able to deliver such technologies to their customers. Thus, the Identity Management Enabler could also deliver a multi-tenant user and profile management solution that allows Enterprises to manage consumers of their (Web based) services in the Cloud securely. Instead of developing and operating the user and profile management by themselves, it can be hosted in the Cloud as a tenant instance and will be delivered on demand.

38.1.2 Basic Concepts

Identity Management encompasses a number of aspects involved with users' access to networks, services and applications, including secure and private authentication from users to devices, networks and services, Authorisation & Trust management, User Profile management, Single Sign-On (SSO) to service domains and Identity Federation towards applications. The Identity Manager is the central component that provides a bridge between IdM systems at connectivity-level and application-level.

Identity Management is used in multiple scenarios spanning from Operator oriented scenarios towards Internet Service Providers (ISP). End users benefit from having simplified and easy access to services (User Centric Identity Management). In the following basic concepts supporting the above mentioned features are described:

38.1.2.1 *User Life-Cycle Management*

The IdM offers tools for administrators to support the handling of user life-cycle functions. It reduces the effort for account creation and management, as it supports the enforcement of policies and procedures for user registration, user profile management and the modification of user accounts. Administrators can quickly configure customized pages for the inclusion of different authentication providers, registration of tenant applications with access to user profile data and the handling of error notifications. For end users, the IdM provides a convenient solution for registering with applications since it gives them a means to re-use attributes like address, email or others, thus allowing an easy and convenient management of profile information. Users and administrators can rely on standardized solutions to allow user self-service features like:

- User registration and login resp. logout,
- Checks for password strength,
- Password reset or renewal procedures or
- Secured storage of user data.

38.1.2.2 *Flexible Authentication Providers*

In addition to providing a native login, the Identity Provider (IdP) supports the integration of multiple 3rd party authentication providers. Foremost, it supports in a first step the configuration of preferred identity providers to lower entry barriers for a native user registration to administrators and on user side to link a preferred 3rd party IdP as alternative authentication provider to a native account.

38.1.2.3 *3rd Party Login*

3rd party login supports customers of the IdM to enhance the reach of their websites by means of attracting users without forcing them to register new user accounts on their sites manually. 3rd party login allows users to register to the customers' sites with already existing digital identities from their favourite 3rd party identity providers, such as e.g. Google, Facebook or Yahoo. Thus, 3rd party login lowers the obstacles of registration processes and increases the number of successful business flows on the customers' sites.

38.1.2.4 *Web Single Sign-On*

As it is possible to configure several applications that shall be linked to his IdM, the main benefit for users is a single sign-on (SSO) to all these applications.

38.1.2.5 *Hosted User Profile Management*

The IdM offers hosted user profile storage with specific user profile attributes. Applications do not have to run and manage their own persistent user data storages, but instead can use the IdM user profile storage as a SaaS offering.

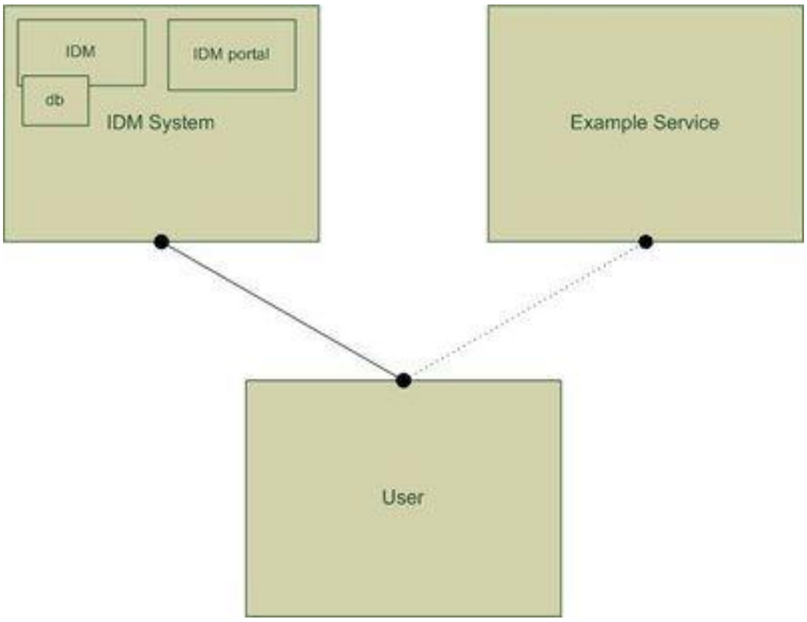
38.1.2.6 *Multi-Tenancy*

A multi-tenancy architecture refers to a principle in software architecture where a single software instance runs on a server, serving multiple client organizations/customers (tenants).

Multi-tenancy is contrasted with a multi-instance architecture where separate software instances (or hardware systems) are set up for different client organizations. With a multi-tenant architecture, a software application is designed to virtually partition its data and configuration, and each client organization works with a customized virtual application instance. In a multi-tenancy environment, multiple customers share the same application, running on the same operating system, on the same virtualized hardware, with the same data storage mechanism. The distinction between the customers is achieved during application design, thus customers do not share or see each other's data. The concept allows each tenant to apply their own branding to login or registration UIs or for user self-services to create a user experience that is aligned with the one offered in a tenant application.

38.1.3 Main Interactions

38.1.3.1 *Example Architecture*



Identity Generic Enabler - High Level Architecture

38.1.3.2 *Modules and Interfaces*

The Identity Management System consists of the following building blocks:

- IDM Portal

Providing the interface to the user / application. The functionality includes user profile management and modification of user accounts (e.g. password settings, secured storage of user data).

- IDM system

The core component handling the authentication requests of the users, by providing e.g. federated IDM for Web-SSO or basic authentication features for devices and services.

- Authentication framework

The authentication framework consists of the Extractor and the Authentication Pipeline. The Extractor extracts the authentication data from different sources. Each one of them is specialized in extracting a special kind of data. There exists a pipeline of authentication data extractors.

- Database

Central repository that stores the user data, profiles and preferences as well as the service provider preferences. This could be implemented as a distributed storage system, depending on the usage scenario.

- Supported Authentication Methods

- OAuth stack

Open Authorization Protocol is an open standard for authorization.

- SAML stack

Security Assertion Markup Language is an XML-based standard for exchanging authentication and authorization data between security domains

- OpenID stack

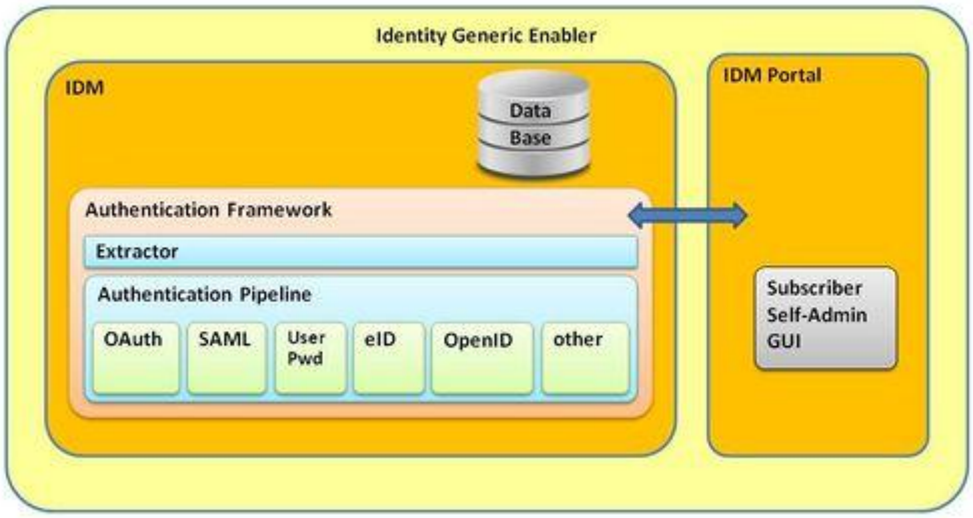
Open standard that describes how users can be authenticated in a decentralized manner

- eID support

The Generic enabler will support European Identity Cards of multiple memebr states.

- user name / password

As well basic authentication mechanisms like user name / password are provided.



Identity Generic Enabler - Core Components

Beyond that there may be some additional network security components necessary (e.g. Firewall, router, …). Next to the Public Key Infrastructer (PKI) this is a prerequisite for the Generic Identity Enabler.It supports the following authentication methods:

1. *SAML (in a later version)*
2. *OAuth*
3. *OpenID*
4. *Username / Password*
5. *eID - cards*

38.1.3.3 *Interface Descriptions*

Different authentication mechanisms are offered by the Identity Generic Enabler. It supports standardized interfaces as well as proprietary once. In the following the interfaces are described with the help of message flows and reference code examples, by thus offering an easy implementation and usage of the Generic Enabler.

SAML

The Identity Management GE makes use of SAML (Security Assertion Markup Language) 2.0 for authenticating federated relying parties and, after authenticating the Users on behalf of the federated relying parties in a second step, for informing them that these Users are authorized to access their services.

The advantages of SAML 2.0

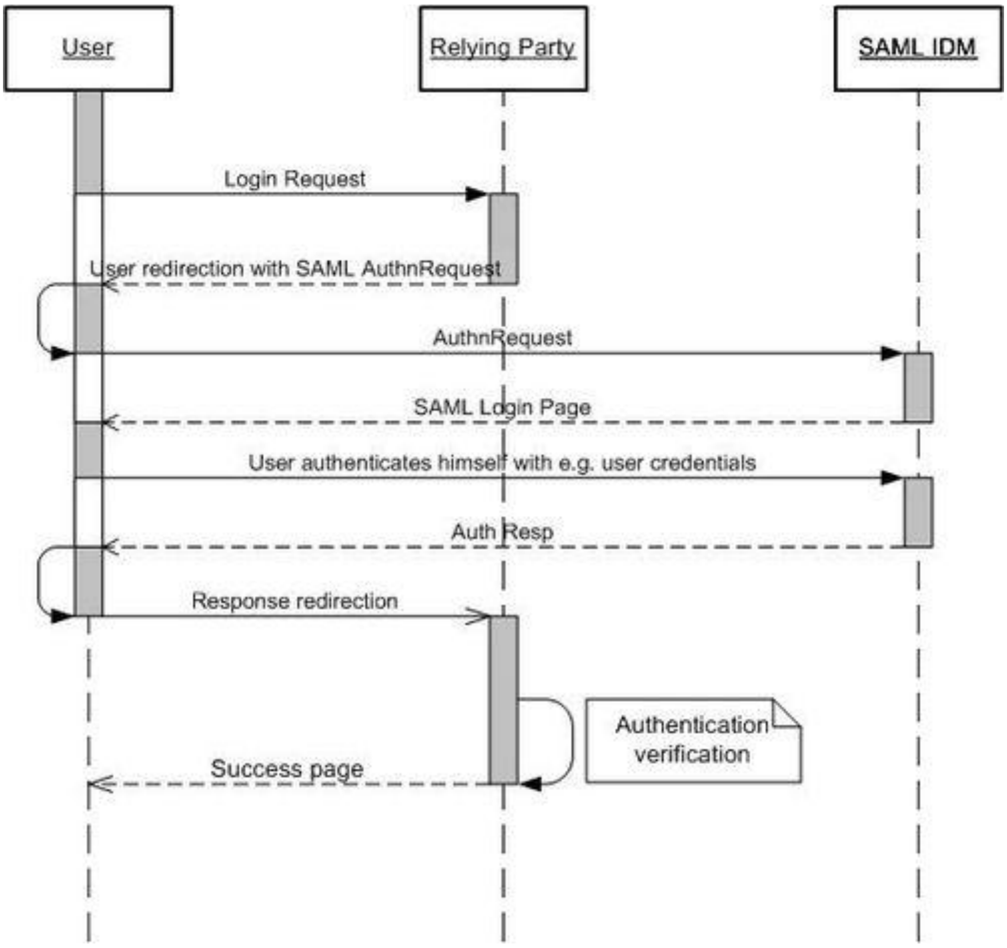
• Provides a means of exchanging data between security domains (i.e. the Identity Management GE and its federated service providers (relying parties))

• Provides the SSO feature for the federated service providers to the Users

• Service providers do not need to authenticate users themselves

• Provides security features such as digital signatures to certify the integrity of the exchanged data (and certified attributes)

• Standardized, non-proprietary protocol (e.g. also supported by Google)



Identity Generic Enabler - SAML Authentication Flow

Authentication Request

```
<samlp:AuthnRequest
  xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
  ID="aaf23196-1773-2113-474a-fe114412ab72"
  Version="2.0"
  IssueInstant="2004-12-05T09:21:59Z"
```

```
  AssertionConsumerServiceIndex="0"
  AttributeConsumingServiceIndex="0">
  <saml:Issuer>https://sp.example.com/SAML2</saml:Issuer>
  <samlp:NameIDPolicy
    AllowCreate="true"
    Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient"/>
</samlp:AuthnRequest>
```

Authentication Response

```
  <samlp:Response
xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
ID="identifier_2"
InResponseTo="identifier_1"
Version="2.0"
IssueInstant="2004-12-05T09:22:05Z"
    Destination="https://sp.example.com/SAML2/SSO/POST">
    <saml:Issuer>https://idp.example.org/SAML2</saml:Issuer>
    <samlp:Status>
      <samlp:StatusCode
        Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
    </samlp:Status>
    <saml:Assertion
      xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
      ID="identifier_3"
      Version="2.0"
      IssueInstant="2004-12-05T09:22:05Z">
      <saml:Issuer>https://idp.example.org/SAML2</saml:Issuer>
      <ds:Signature

xmlns:ds="http://www.w3.org/2000/09/xmldsig#">...</ds:Signature>
      <saml:Subject>
        <saml:NameID
          Format="urn:oasis:names:tc:SAML:2.0:nameid-
format:transient">
          3f7b3dcf-1674-4ecd-92c8-1544f346baf8
        </saml:NameID>
        <saml:SubjectConfirmation
          Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
          <saml:SubjectConfirmationData
            InResponseTo="identifier_1"
            Recipient="https://sp.example.com/SAML2/SSO/POST"
            NotOnOrAfter="2004-12-05T09:27:05Z"/>
        </saml:SubjectConfirmation>
      </saml:Subject>
      <saml:Conditions
        NotBefore="2004-12-05T09:17:05Z"
        NotOnOrAfter="2004-12-05T09:27:05Z">
        <saml:AudienceRestriction>
```

```
<saml:Audience>https://sp.example.com/SAML2</saml:Audience>
        </saml:AudienceRestriction>
      </saml:Conditions>
      <saml:AuthnStatement
        AuthnInstant="2004-12-05T09:22:00Z"
        SessionIndex="identifier_3">
        <saml:AuthnContext>
          <saml:AuthnContextClassRef>

urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
         </saml:AuthnContextClassRef>
        </saml:AuthnContext>
      </saml:AuthnStatement>
    </saml:Assertion>
  </samlp:Response>
```

OAuth

Contrary to SAML, OAuth has nothing to do with SSO. Main focus is not on attributes, but ‘resources’ of the user which a consumer wishes to access.

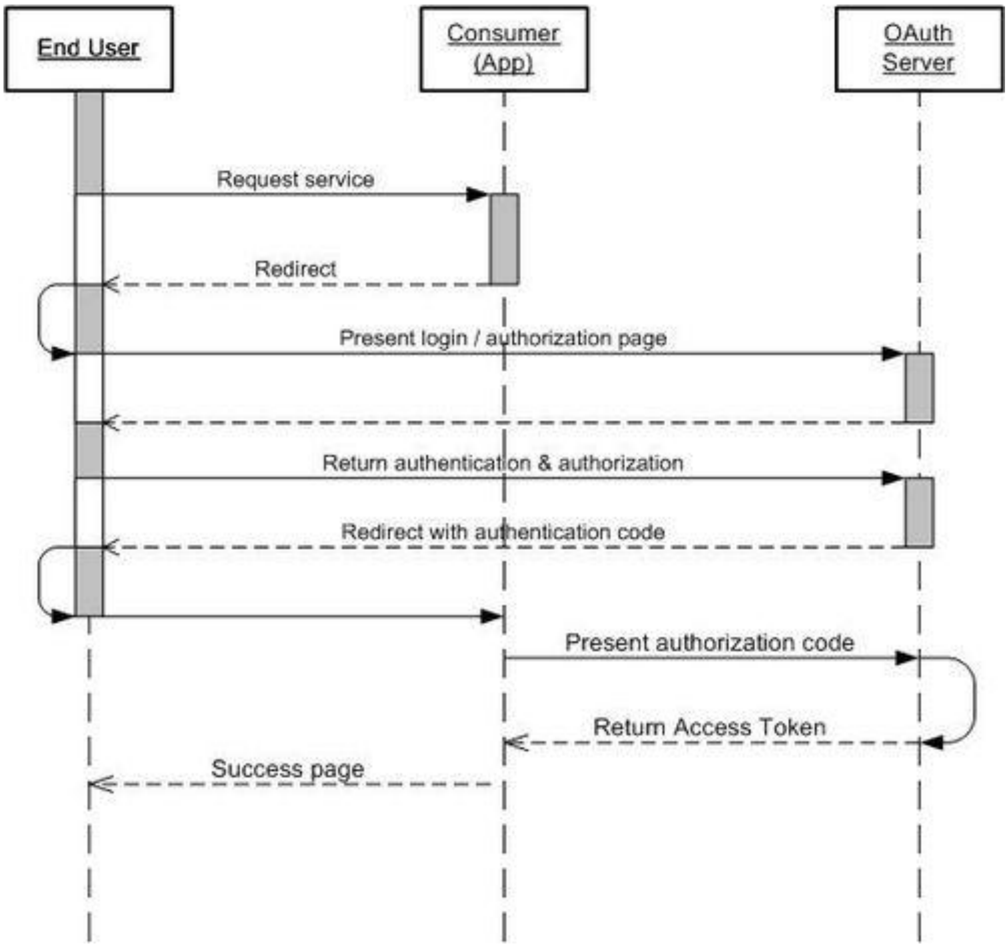
The advantages of OAuth 2.0

• A standardized protocol supported by a wider set of Service Providers (Facebook, Google,LinkedIn, …)

• The user grants access for a consumer to a specific resource by providing an access token to the consumer. The user need not to be online, when the consumer accesses the resource, i.e. actually makes use of the access token.

• The consumer always requires the consent of the user for receiving the access token.

• The user is in full control of who can access his resources.



Identity Generic Enabler - OAuth 2.0 Authentication Flow

Get Request Token

```
https://api.login.<xyz.com>/oauth/v2/
   get_request_token?oauth_nonce=ce2130523f788f313f76314ed3965ea6
   &oauth_timestamp=1202956957
   &oauth_consumer_key=123456891011121314151617181920
   &oauth_signature_method=plaintext
   &oauth_signature=abcdef
   &oauth_version=1.0
   &xoauth_lang_pref="en-us"
   &oauth_callback="http://yoursite.com/callback"
```

Get Request Token Response

```
oauth_token=z4ezdgj
&oauth_token_secret=47ba47e0048b7f2105db67df18ffd24bd072688a
&oauth_expires_in=3600
```

```
&xoauth_request_auth_url=https%3A%2F%2Fapi.login.<xyz.com>%2Foauth%2
Fv2%2Frequest_auth%3Foauth_token%3Dz4ezdgj
&oauth_callback_confirmed=true
```

Get Access Token

```
https://api.login.<xyz.com>/oauth/v2/request_auth?oauth_token=j5nyp6
```

Get Access Token Response

```
http://yoursite.com/callback?oauth_verifierer=svmhhd
```

Exchange Token Request

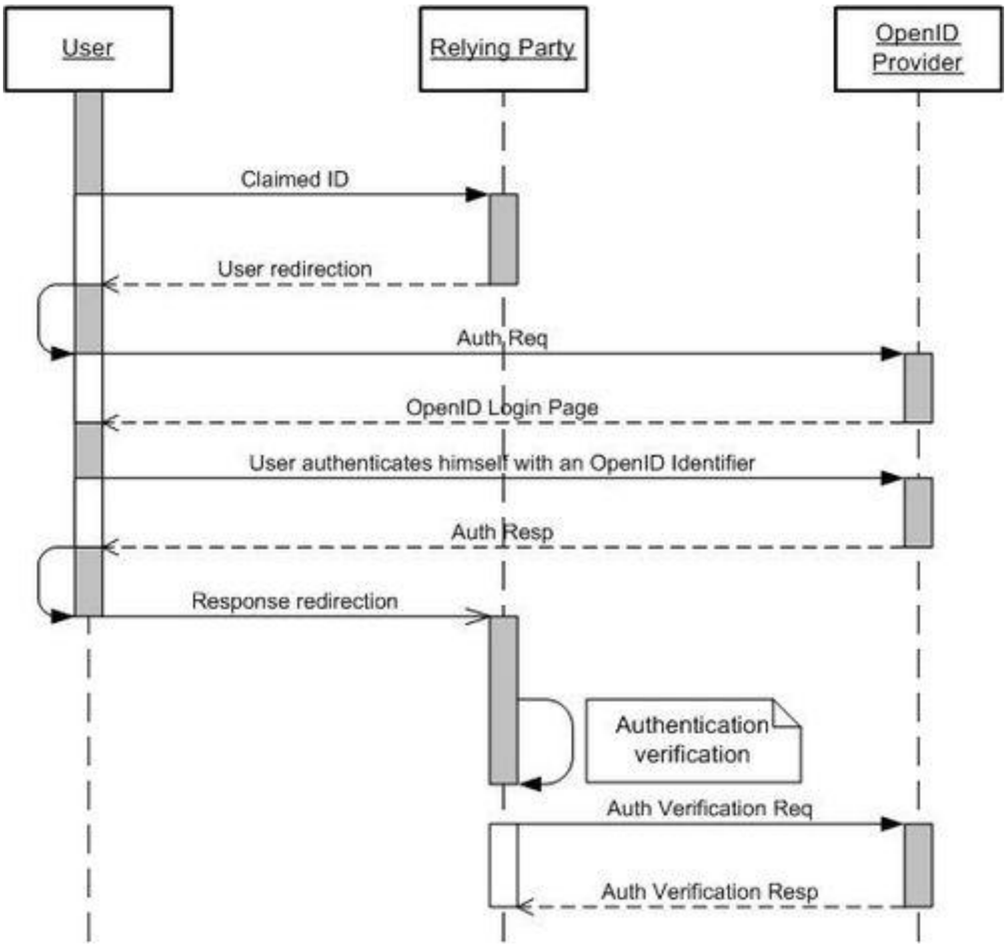
```
https://api.login.<xyz.com>/oauth/v2/get_token?oauth_consumer_key=dj
0yJmk9NG5USlVvTlZsZEpnJmQ9WVdrOVQwa

zFPRUozTkc4bWNHozlNVE13TXprM01UUTBNZy0tJnM9Y29uc3VtZXJzZWNyZXQmeD1kN
g--
&oauth_signature_method=PLAINTEXT
&oauth_version=1.0
&oauth_verifier=svmhhd
&oauth_token=gugucz&oauth_timestamp=1228169662
&oauth_nonce=8B9SpF

&oauth_signature=5f78507cf0acc38890cf5aa697210822e90c8b1c%261fa61b46
4613d0d32de80089fe099caf34c9dac5
```

Exchange Token Response

```
oauth_token=A%3DqVDHXBngo1tEtzox.JMhzd91Rk99.39Al7hos3J80mm1j_3nGP4B
iilL777vUj2rsPLj1cZw.srbisvw.cz42Lzmlxt

H0Kk9mkXilvS1ll5lNoMKXO5zy5YG4vO3fbGKewp7IESYMIdEi4Md7SroYiv6kBCEjqB
4jXr0.8XsMvO1QgZ.aKNKXwc2sv3n4BOZxs

54tzXV6rGNpEHZUaj9CovPUo44isTgs9FnLIKpXFCU4Jq1BB3_IOTFBNf1vtf5vSxaxe
_L5dUhr.i15Hx0LTZ2tlsWeDcActSGGBWVc

vytPF3cK9mDWy44baBgCVI3AEbGCqg.NGhDPqOh1ZHfKFtYlBZfG4xf2n..CdxcM5x4I
NxnVz2.biMkfhfkw8haJuR0RaUY37lBxZ9z
```

OpenID

In case if a user wants to login to the service of a Service Provider (SP) [referred to as "Relying Party" (RP) in the OpenId protocol] a "Claimed Identifier" is being presented by the user . The SP analyses the Identifier and redirects the user to his OpenId Provider (OP) with an OpenID Authentication request, in consequence of which the user authenticates himself against this OP. Nevertheless it is important to mention that the authentication process between the user and the OpenID server is not part of the OpenId protocol. After successful authentication procedure the OP redirects the user back to the RP with an assertion, indicating that the authentication was successful (Positive Assertion), else with the information that authentication failed (Negative Assertions). In the positive case the RP verifies the received information by sending a return URL, the discovered information, the nonce and the signature within a request to the OP. Alternatively the OP and the RP can establish an association after the RP received the Claimed Identifier by using the key exchange protocol standardized by the IETF.



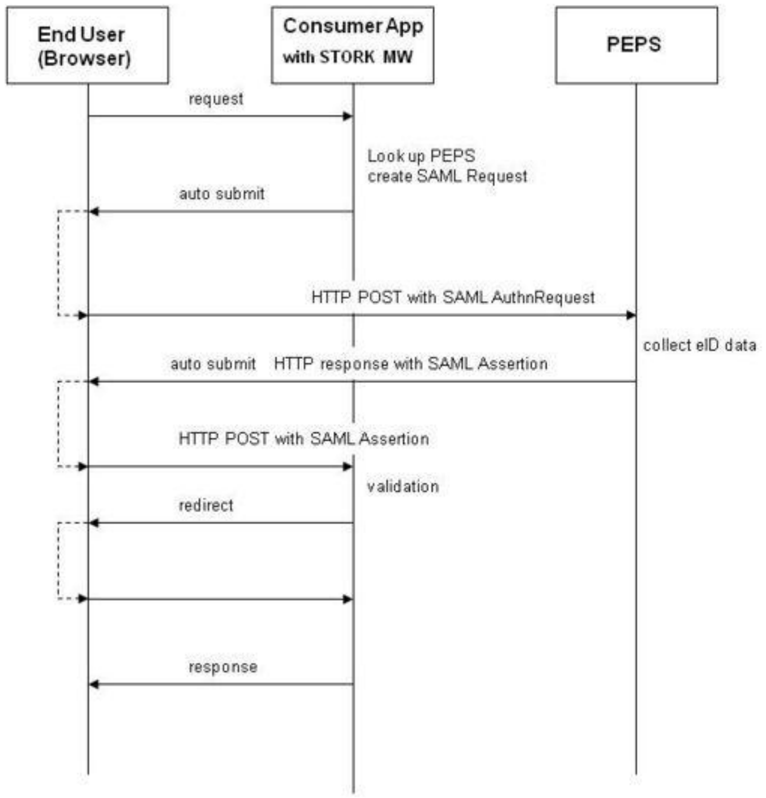
Identity Generic Enabler - OpenID Authentication Flow

Username / Password

A user can be identified by providing a <user name> and a <pass word>, which will be verified against data base.

eID – card

Alternatively to <user name> and <pass word> a user can identify himself/herself with an European eID (electronic identity). This is not possible with all eIDs of all European countries, but with those, supported by the STORK project. European countries have a variety of different and incompatible eID solutions. STORK has developed an integrative wrapper, allowing a uniform, country-independent access. In the Generic Identity Enabler Scenario the IDM will delegate the user authentication to an external eID service. The information exchange with the eID Service is SAML based.



Identity Generic Enabler - Access Portal with eID Authentication

38.1.3.4 *Open Specification*

- [Identity Management Generic Enabler API Specification]

38.1.4 References

[SAML]	http://code.google.com/intl/de-DE/googleapps/domain/sso/saml_reference_implementation.html http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security#samlv20

[OAuth]	http://oauth.net http://en.wikipedia.org/wiki/OAuth
[OpenID]	http://openid.net/specs/openid-authentication-2_0.html current version http://openid.net/connect/ next version based on oauth2
[e-ID]	http://www.eid-stork.eu

39 FIWARE ArchitectureDescription Security Privacy Generic Enabler

You can find the content of this chapter as well in the wiki of fi-ware.

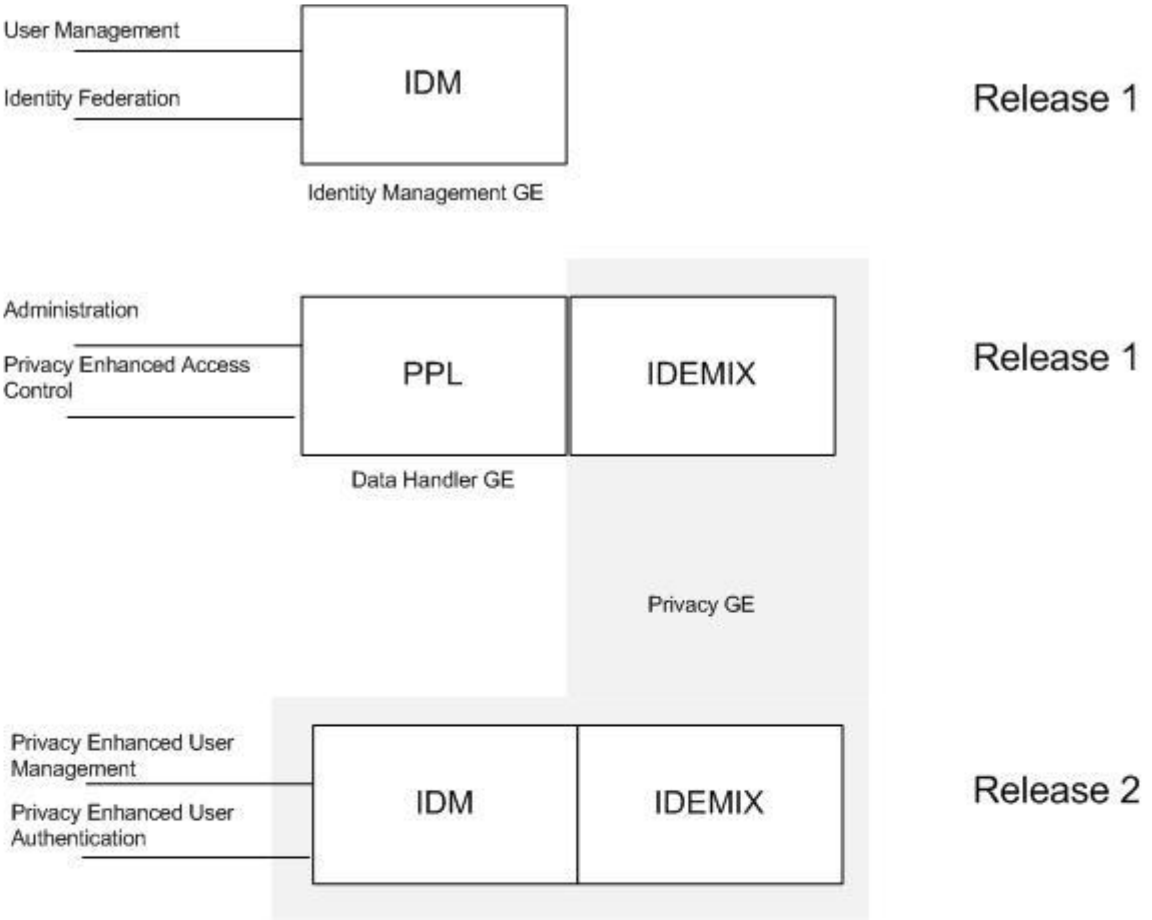
39.1.1 Privacy Generic Enabler

The Privacy Generic Enabler will not be part of the first FI-WARE release and as such the documentation in this section is rather preliminary.

The Privacy GE does not expose its own external interfaces directly; rather it enhances the functionality provided by the other generic enablers:

- Identity Management GE,
- Data Handling.GE

These generic enablers will provide a series of interfaces for release 1 of FI-WARE.



The Data Handling GE in release 1 will expose all privacy related interfaces to consuming applications. These interfaces are described in the Data Handler GE. The Identity Management GE interfaces will be enhanced with privacy functionality to create the Privacy Generic Enabler in Release 2 of FI-WARE.

The Privacy GE offers trustworthy, yet privacy-friendly authentication. The privacy features provided by the underlying privacy-enhancing technology will be made accessible through an adapted version of the IDM system (which is also the core of the Identity Management GE). Thus, the interfaces and main interactions that will be exposed by the Privacy GE heavily depend on the specification of the adapted IDM system and their integration with the underlying technology.

Since this integration is ongoing work, and the Privacy GE is scheduled for the second release of FI-Ware only, we only briefly sketch the main functionality provided by the Privacy GE and refer for the full specification to the second version of this deliverable.

In a nutshell, the Privacy GE allows the User to first obtain credentials, which are certified attribute-value pairs, from an Issuer who vouches for the correctness of the certified attributes. The User can subsequently authenticate towards a Verifier by sending a presentation token which is derived from her credentials. A single presentation token can selectively reveal attribute values from one or more credentials. It can also prove that a given predicate over one or more attributes holds without revealing the full attribute values, e.g., that the birthdate is before January 1st, 1994, or that the name on the User’s credit card matches that on her driver’s license.

40 FIWARE ArchitectureDescription Security Data Handling Generic Enabler

You can find the content of this chapter as well in the wiki of fi-ware.

40.1.1 Copyright

40.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

SAP strives to make the specifications of this Generic Enabler available under IPR rules that allow for a exploitation and sustainable usage both in Open Source as well as proprietary, closed source products to maximize adoption.

40.3 Overview

The Data Handling GE is a privacy-friendly attribute-based access control system, that targets mainly sensitive data. It permits to store information together with an attached privacy policy, that regulates its usage. Thus, Data Handling GE can reveal certain attributes, according to specific supplied prove conditions. Data Handling GE supports integrated data handling (two-sided detailed data handling), that takes into account specific preferences/policies expressed using the PPL language (Privacy Policy Language)[PPL]. PPL is based on the XACML standard [XACML]. Data usage purpose must always be declared, as it is a relevant part of the policy that must be expressed, as well as downstream usage, i.e., whether one can disclose collected data with third parties. The PPL language supports the enforcement of a number of obligations, that are bound tightly to data. For instance, one can impose a specific retention period, as well as the production of user's notifications and/or logging under certain conditions.

40.3.1 Target usage

The Data Handling GE provides a mechanism for controlling the usage of attributes and data (more precisely, of Personal Identifiable Information or PII) based on the concept of ‘sticking’ a data usage policy to the data to which it applies. When the data is accessed by an application, an access control technology is then used to verify that the intended use of the data matches the scope defined in the usage policy. Therefore, the Data Handling GE can be used by any application or service that would offer a transparent data handling policy to users and third parties. In the example scenario later proposed, a

40.4 Basic Concepts

40.4.1 Relevant Concepts and Ideas

In this section, the more important concepts shall be presented. The used terminology is coherent with definitions contained in the European Parliament Directive 95/46/EC, "on the protection of individuals with regard to the processing of personal data and on the free movement of such data". More detailed information is provided in the Terminology section.

Data Controller

"Data Controller" indicates the entity which (alone or jointly with others) determines the purposes and means of the processing of personal data.

Data Subject

The Data Subject is the person whose personal data are collected, held or processed by the Data Controller.

The Data Subject has the right to access his data and to require the Controller to rectify without delay any inaccurate or incomplete personal data. The Data Subject has the right to require the Controller to erase data if the processing is unlawful.

Personal Data (Personal Identifiable Information, or PII)

Personal data means any information relating to an identified or identifiable natural person or "Data Subject".

An identifiable person is someone who can be identified, directly or indirectly, in particular by reference to an identification number or to one or more factors specific to his or her physical, physiological, mental, economic, cultural or social identity.

User Agent

A software system (such as a web browser) acting on behalf of a user. The user agent acts on user preferences when dealing with a server acting on behalf of a Data Controller.

40.4.2 PII and PPL

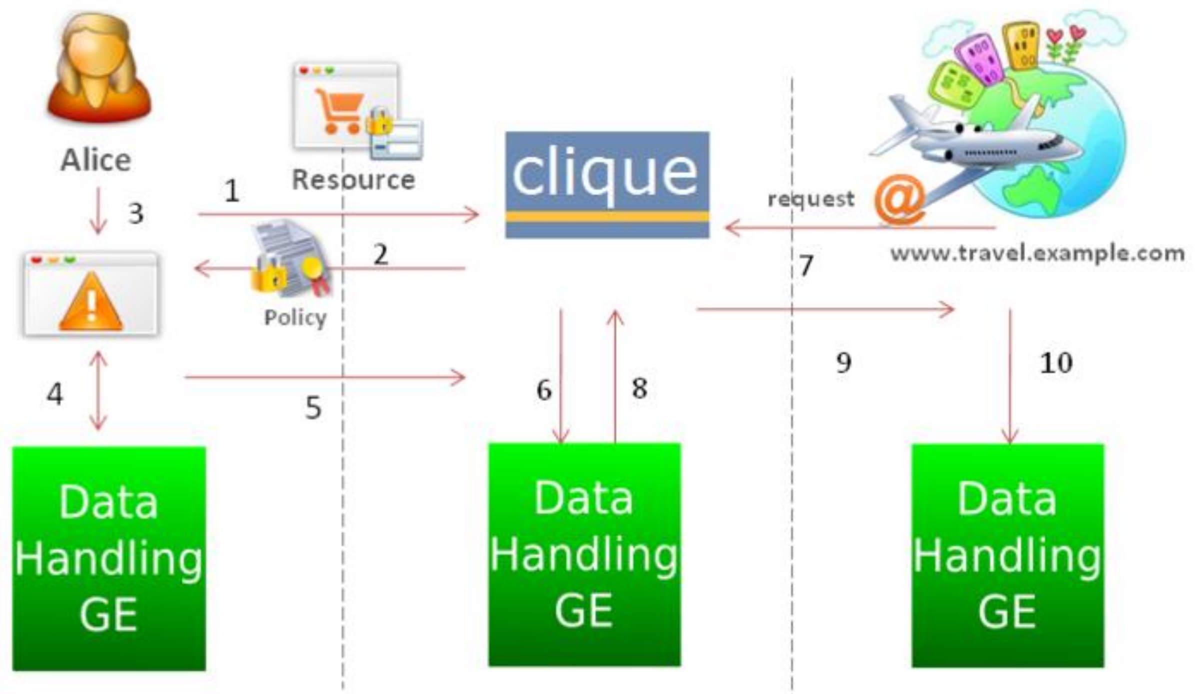
The Data Handling GE regulate the access to sensitive data, collected from users. This can be achieved through the association of a set of preferences/policies to each PII; privacy policies are expressed using the PPL. PPL is used:

1. at PII collection time, as each information that enters the Data Handling GE must come along with a PPL policy;
2. at each data usage, that is regulated according to the associated PPL policy.

In fact, for each data access, the Data Handling GE evaluates its purpose, which must always be declared. Access purpose is a relevant part of the policy that must be expressed, and if and only if there is a compatibility between the data policy and the request policy, the information are provided to the requester. The Data Handling GE is also responsible for fulfilling obligations contained in PPL policies, like for instance, sending an email to the data owner at each access.

40.4.3 Example Scenarios

40.4.3.1 *Use Case: Connection to a Social Network Website*



In this scenario, it is described a subscription to a social network shopping website. The scenario is depicted in the previous picture, and it underlines the different information exchanged between the different parties.

Prerequisite: a different Data Handling GE is available for each peer, and interacts with each subject only using the public interface, so no direct access to stored data is possible.

- *Step 1*: The Data Subject Alice is a privacy-aware user who is quite active in the Web 2.0 websites, but who is concerned about what happens to the data that she provides about herself. Before starting her social networking activity, she has to create an account at an online social network, Clique, an experimental social network. Clique is playing the role of Data Controller.
- *Step 2*: In order to validate her subscription, the website will need to collect some personal information, like her name, her birth date, her e-mail address and her street address... This information is contained in an access control policy on the website side (i.e., In order to create an account the user has to provide a list of credentials). In order to explain the conditions of usage of such collected information the website sends a privacy policy (written in PPL language).
- *Step 3*: Previously, Alice created privacy preferences related to her personal data.
- *Step 4*: After receiving the website’s privacy policy, the Data Handling GE engine of Alice automatically check if the private data requested by the server is stored on Alice's machine. If it is the case, the engine will enforce the access control rules related to the requested data; in other words, Alice's Data Handling GE will check whether Alice's policy and website's policy are compatible (ex. does the domain fi-ware.eu can access my e-mail address?). If the domain is allowed to access this data the engine creates a new privacy policy, combining the privacy policies of both

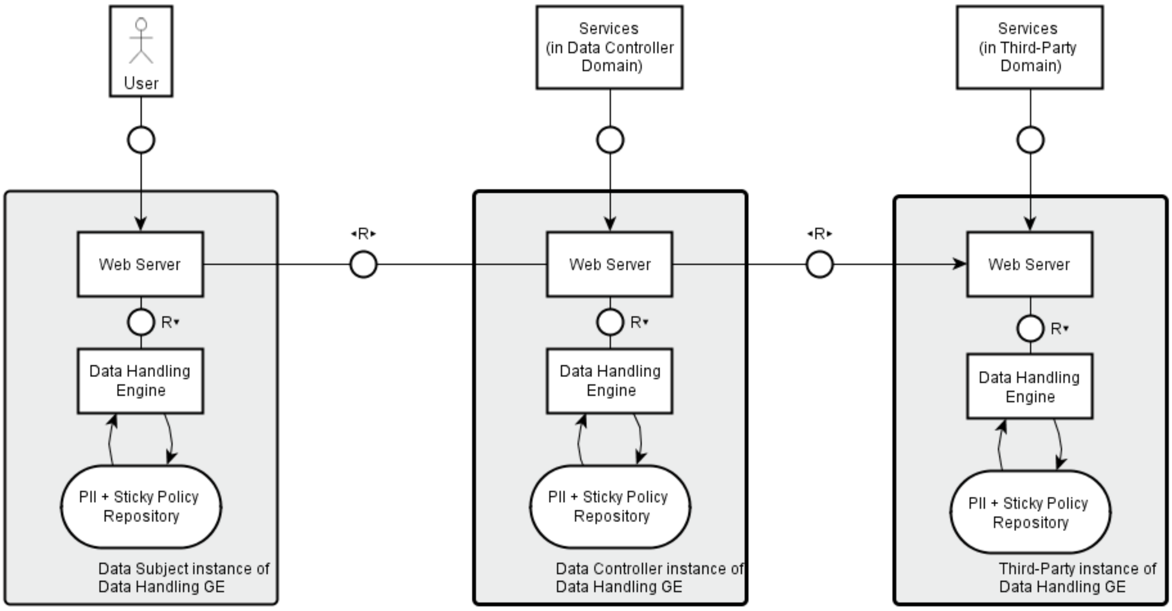
website and Alice. This new policy will comprise the data handling conditions expressed by the two parties.

- *Step 5*: Alice has the possibility to decide if she accepts or refuses to send her data, according to the newly generated privacy policy. If Alice accepts, the new policy gets associated to the requested data, and it becomes a "*sticky policy*". Private Alice's data and sticky policy will be sent at the same time to the Data Handling GE of the website (the Data Controller).
- *Step 6*: The website's Data Handling GE will store the received information, and the website will be able to use this data according to the obligations stated in the sticky policy.
- *Step 7*: The online travel agency www.travel.example.com (third party Data Controller) decided to start an e-mail advertising campaign. In order to target a wide scope of persons, the www.travel.example.com admin asked his partner Clique to provide him with valid e-mail addresses for marketing and statistics purposes. The request will contain a resource query for e-mail and a privacy policy.
- *Step 8*: The policy engine of Clique will compare the privacy policy of travel.example.com with the sticky policy related to Alice's data (and in particular, the obligations on the e-mail address), to verify that the sticky policy allows to forward the protected information for the purpose of statistics. In general, a full matching is performed between travel.example.com's policy and Alice's sticky policy.
- *Step 9 and 10*: The travel.example.com website receives the e-mail address of Alice with a sticky policy (Step 9), that is derived from the original one (see Step 5). The travel agency Data Handling GE configures the actions and the triggers related to the sticky policy obligations, and it stores the e-mail and the sticky policy in the same way as was done by the Clique Data Handling GE (Step 10). In this way, no misuse of data is allowed, as data access is again protected and regulated by the use of privacy policies, as already explained.

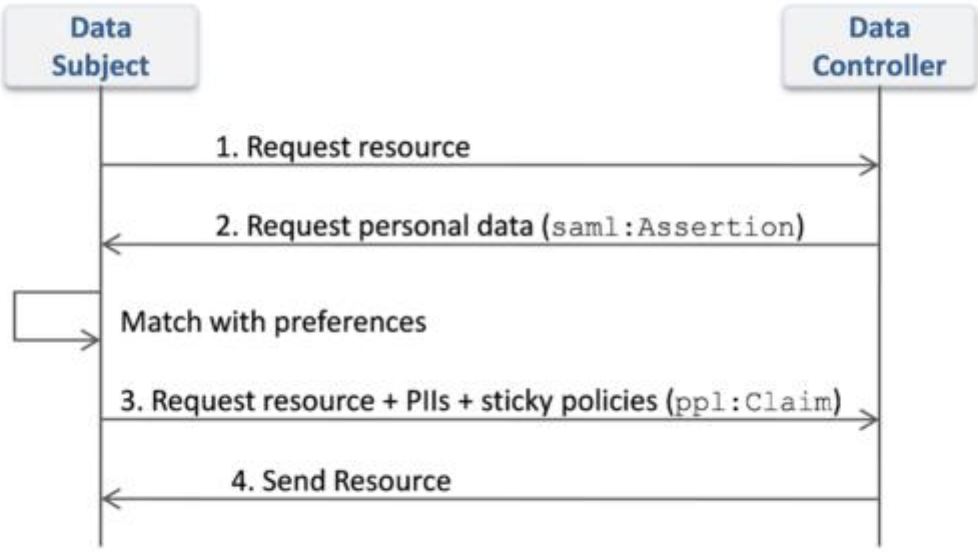
40.5 Main Interactions

40.5.1 Architecture

40.5.1.1 *Block Diagram*



40.5.1.2 *Sequence Diagram*



Description:

This sequence diagram reflects the use case proposed in Use Case. This explanation keeps the same concepts, focusing on detailing Steps 1 to 6, however the numeration of the sequence diagram does not reflect exactly the one of Use Case. *Data Subject* and *Data Controller* are distinguished instances of the Data Handling GE.

1. **Request Resource**: this refers to the access to a resource, that is regulated by the Data Controller. In the Use Case, it refers to Step 1
2. **Request Personal Data**: as the requested resource requires the transmission of PIIs from the Data Subject, the Data Controller invokes the API "Get a PII" of the Data Subject, together with a privacy policy that describes how the PII requested will be used (in a SAML envelope). This reflects Step 2 in the Use Case.
    - **Match With Preferences**: the Data Subject checks whether there is a compatibility (a "match") between user's privacy policy and Data Controller's privacy policy. This is Step 4 in the Use Case.
3. **Request resource + PIIs + sticky policies**: if the two privacy policies match, PIIs are sent to Data Controller, together with the initial request and a number of sticky policies (one for each PII). This is Step 5 in the Use Case.
4. **Send Resource**: Eventually, the resource is sent to the Data Subject. Step 6 of the Use Case.

40.6 Basic Design Principles

The Data Handling GE permits to protect information according to a specific privacy policy. The Data Handling GE safeguards data, storing them together with the respective privacy policies. Any access to the protected resources can happen only declaring explicitly its purposes (using again a description encoded in a privacy policy). The Data Handling GE evaluates the two policies (data and access requests), and transmits the requested information if and only if the policies match.

40.6.1 Detailed Specifications

Following is a list of Open Specifications linked to this Generic Enabler. Specifications labeled as "PRELIMINARY" are considered stable but subject to minor changes derived from lessons learned during last interactions of the development of a first reference implementation planned for the current Major Release of FI-WARE.

40.6.1.1 *Open API Specifications*

- FIWARE.OpenSpecification.Security.DataHandlingGE.Open RESTful API Specification

40.7 Appendix

40.7.1 References

EC Data	http://eur-

Protection directive	lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:EN:HTML
PPL Language	http://www.primelife.eu/results/documents/153-534d
XACML	http://xml.coverpages.org/xacml.html

41 FIWARE ArchitectureDescription Security Optional Security Enablers

You can find the content of this chapter as well in the wiki of fi-ware.

The Optional Security Enablers are services that can complement applications with security-specific vertical functionalities, that are not covered by Core Generic Enablers.

In FI-WARE Major Release 1, the following Enablers have been released:

41.1 DBAnonymizer

FIWARE.ArchitectureDescription.Security.Optional Security Enablers.DBAnonymizer

41.2 Secure Storage Service

FIWARE.ArchitectureDescription.Security.Optional Security Enablers.SecureStorageService

In FI-WARE Major Release 2, the following Enablers will be released:

41.3 Morphus antivirus

FIWARE.ArchitectureDescription.Security.Optional Security Enablers.Morphus

New Optional Security Enablers will be added in FI-WARE Releases 2 and 3, mainly as result from requirements collected in interactions with the Use Case FI-PPP projects and the other FI-WARE chapters.

42 FIWARE ArchitectureDescription Security Optional Security Enablers DBAnonymizer

You can find the content of this chapter as well in the wiki of fi-ware.

42.1 Copyright

- Copyright © 2012 by SAP

42.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

SAP strives to make the specifications of this Generic Enabler available under IPR rules that allow for a exploitation and sustainable usage both in Open Source as well as proprietary, closed source products to maximize adoption.

42.3 Overview

Large Organizations hold thousands of terabytes of datasets about their customers or their activities. They often have to release data files containing private information to third parties for data analysis, application testing or support. To preserve individuals' privacy and comply with privacy regulations, part of released datasets have to be hidden or anonymized using various anonymization techniques, before data release.

However, two different problems may arise: first to decide if a piece of data has to be considered private or not, second, to assess whether the exposure of non-private data could be used by correlation algorithms to infer hidden private data. The second task is particularly challenging, and cannot be handled manually for large datasets, where the potential number of combinations of different fields is extremely large. In fact, disclosure policies are typically described by human users (security experts or not) that are not able to predict all the possible combinations of the data that could ease the guess of private data contained in the dataset. In some other cases, policy writers are not necessarily security experts and could expose sensitive data without being aware about the impact of such exposure.

DB Anonymizer is a database risk evaluation and anonymization service; it can be used as a support tool in case of dataset disclosure. In fact, DB Anonymizer analyses whether a shared database is not vulnerable to the re-identification of the non-shared part of the database. In other words, the service allows understanding if a certain policy to be used to anonymize a dataset should be considered safe or not. In particular, the service exposes a function that calculates a value, that represents the likelihood (0->impossibility, 1->certainty) that an attacker can reconstruct exactly a table's content, that is anonymized using a certain obfuscation policy.

42.3.1 Target usage

The service can be used to support these DB administrators to evaluate the disclosure risk for all their types of data; by recommending the safest configurations using a smart bootstrapping system. The service provides the user with an estimation of the re-identification risk when disclosing certain information, and proposes safe combinations in order to help him during the information disclosure. Albeit privacy risk estimators have already been developed in some specific contexts (statistical databases), they have had limited impact, since they are often too specific for a given context, and do not provide the user with the necessary feedback to mitigate the risk. In addition, they can be computationally expensive on large datasets. DB Anonymizer is specifically designed to address all these issues, exposing a simple ReSTful API that can be easily integrated in any application.

42.4 Basic Concepts

The service needs to receive a dump of a MySQL table, containing all data, together with a disclosure policy. Both inputs are mandatory to let the service's algorithm to be able to evaluate the effectiveness of the disclosure policy. Once the policy is evaluated, the table is dropped from the DB and the file dump is erased. The application server encapsulation model permits a complete isolation of each user's data, and any intermediate result created during the algorithm's execution is deleted immediately at the end of the computation.

42.4.1 Input Format

The main function has two parameters:

1. an SQL table dump (for MySQL), containing all information to be disclosed: this file shall contain only a table definition and a set of elements to populate it;
2. a policy file in XML, which describes which information of the previously specified table are going to be disclosed or not: the policy file has the following syntax:

```
<Policy>
  <Column>
    <Name>Gender</Name>
    <Type>identifier</Type>
    <Hide>false</Hide>
  </Column>
  <Column>
    <Name>Wine</Name>
    <Type>sensitive</Type>
    <Hide>false</Hide>
  </Column>
</Policy>
```

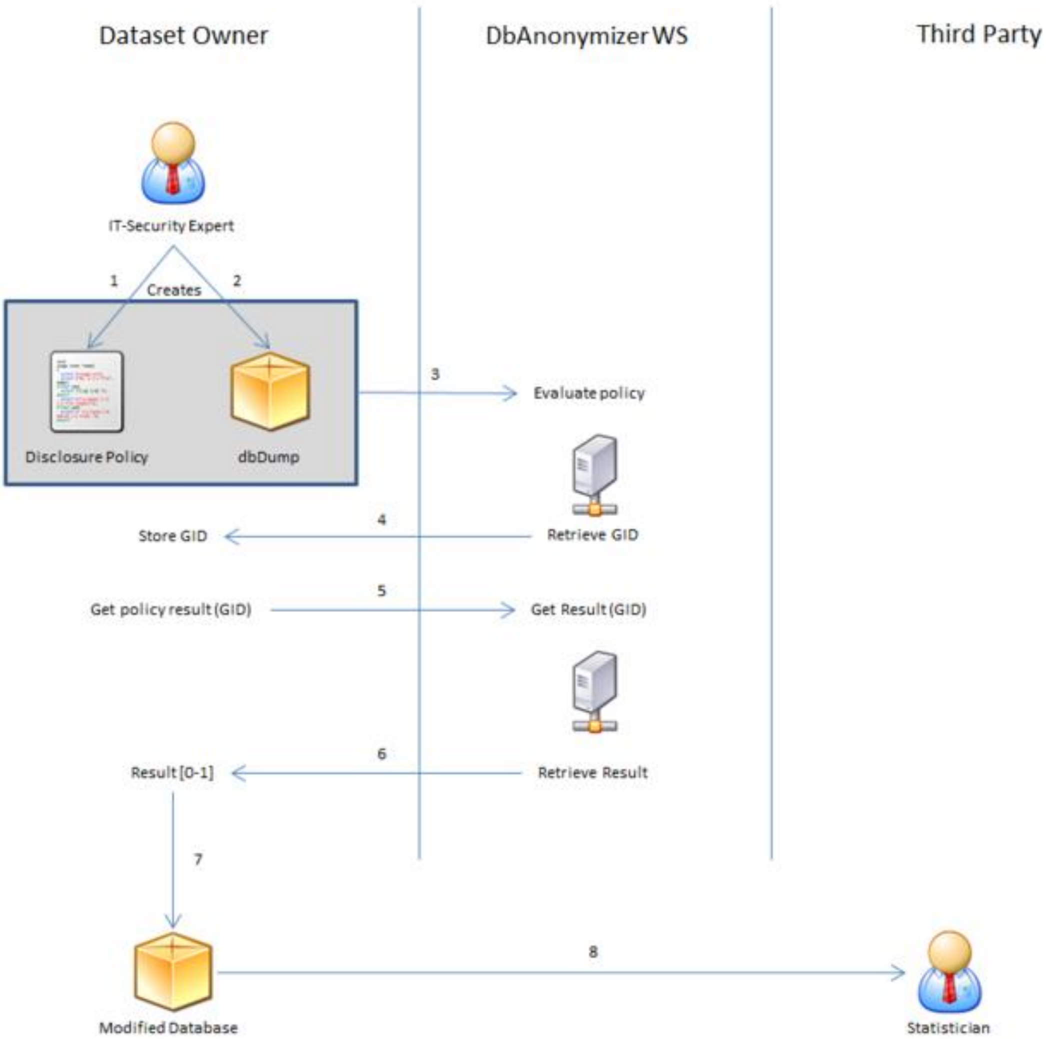
The “Type” information shall be “identifier” or “sensitive”, in order to allow the service algorithm to distinguish them. Please refer to the Glossary (at this link: FIWARE.Glossary.Security.Optional Security Enablers.DBAnonymizer) for an explanation of the two terms.

The service supports MTOM, a W3C standard which allows services to transfer binary data efficiently and conveniently (see http://www.w3.org/TR/soap12-mtom/).

42.4.2 Use Case

A company holds information about people structured in dataset records. Each record has many attributes, such as birthday, address, marital status and occupation, that are useful for company's purposes, but usually are not sensitive, if considered in isolation. Other attributes related to the connection between an individual and the company, such as customer purchases, debts, and credit rating, may be sensitive. Suppose a dataset is released with obvious identifiers, such as social security number, name and address, omitted, some other attributes such as occupation and marital status left intact, and other key and sensitive attributes modified to preserve confidentiality. For example, salaries might be truncated, ages grouped more coarsely, and zip codes swapped on pairs of records. Furthermore, some attributes on some records might be missing or intentionally removed. If the anonymization process is not carefully designed, it could be possible to use techniques to reconstruct the original dataset, as a whole or in parts, also by cross-comparing it with other datasets (e.g., a similar dataset of a competitor). The DB Anonymizer allows to evaluate an anonymization policy, in order to measure its robustness to dataset reconstruction techniques.

Let us consider the following example.

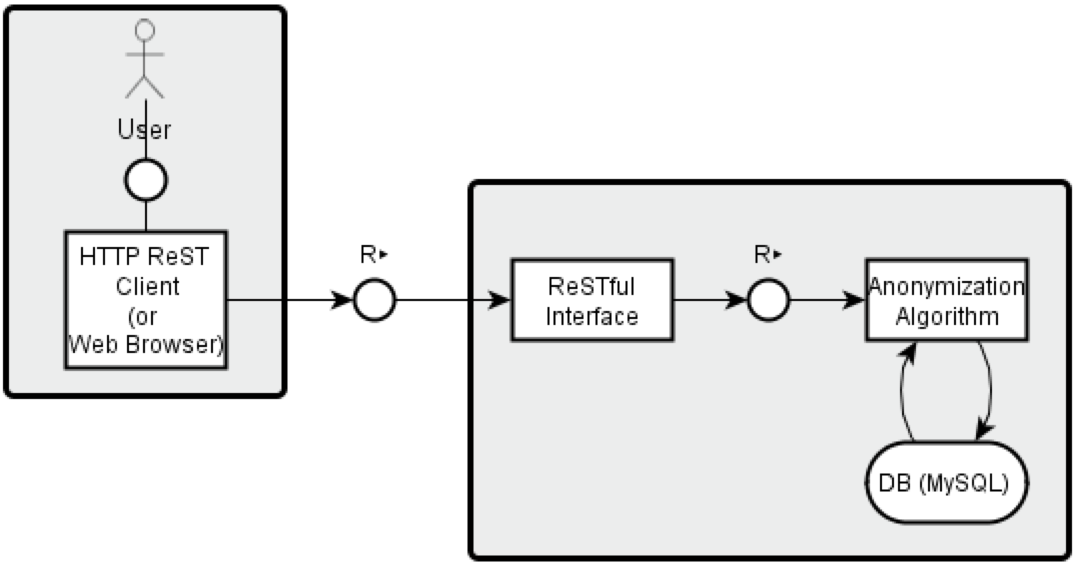


1. The IT-Security Expert creates the Disclosure Policy.
2. The IT-Security Expert creates the DB Dump.
3. DB Dump and Disclosure Policy are sent to the DbAnonymizer WS using the evaluate policy.
4. The DbAnonymizer WS sends back the Result Identifier (GID).
5. The IT-Security Expert ask for the evaluation result.
6. The DbAnonymizer WS sends back the evaluation result.
7. The IT-Security Expert modifies the DB data, according to the accepted policy.
8. The modified DB dump is sent to the Consulting Company.

42.5 Main Interactions

42.5.1 Architecture

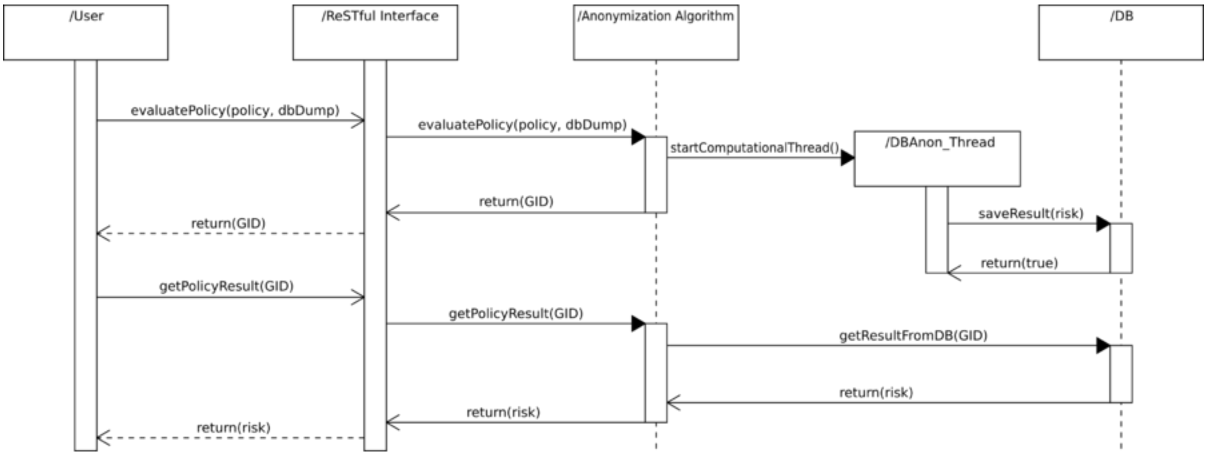
42.5.1.1 *Block Diagram*



The previous block diagram shows the different elements that compose the DB Anonymizer service. Starting from the DB Anonymizer block (on the right side of the diagram), the core of DB Anonymizer is the Anonymization Algorithm component, that interact closely with an internal MySQL database. The Anonymization Algorithm interacts with users through another component, that exposes a ReSTful interface. More precisely, the ReSTful interface component is responsible for invoking the Anonymization Algorighm operations, and to provision them with user inputs.

In the left part of the block diagram, a user is depicted together with a ReSTful client component, with which it is possible to interact with DB Anonymizer ReSTful interface. The ReSTful client can also be implemented by a traditional web browser.

42.5.1.2 *Sequence Diagram*



The previous block diagram shows the order with which DB Anonymizer operations should be invoked; the entities depicted are the same as for the previous block diagram.

The DB Anonymizer API is composed by two methods, to be invoked by users in the following order:

1. evaluatePolicy;
2. getPolicyResult.

The first method allows for starting the analysis of an anonymization policy together with the associated dataset. The ReSTful interface component exposes this method, and any incoming request get routed and served by the Anonymization Algorigthm component, that creates a new computing process. The Anonymization Algorithm component returns immediately a request identifier (GID) to the ReSTful component and thus to the user, which can be used to retrieve the analysis result. Each computation process performs its analysis on the received policy and dataset, and then writes a result to the DB. At that point, the process terminates, deleting any data it used. The second method can be invoked by users to retrieve the result of a computation, identified by a GID. The result of getPolicyResult can be the analysis result when available, otherwise an error code (result is not ready, error in receiving parameters and so on; please refer to the ReSTful API documentation for a detailed error code list and explanation).

42.6 Basic Design Principles

The service manipulates user data in a secure way; dataset and policies are deleted from the application just after their use, to keep confidential any information transmitted. A temporary MySQL table is created at the beginning of the operations, and destroyed just before returning the final result to the invoker. A new process is created for each user request, to ensure data isolation during computation phases.

42.7 Detailed Specifications

Following is a list of Open Specifications linked to this Generic Enabler. Specifications labeled as "PRELIMINARY" are considered stable but subject to minor changes derived from lessons learned during last interactions of the development of a first reference implementation planned for the current Major Release of FI-WARE.

42.7.1 Open API Specifications

- FIWARE.OpenSpecification.Security.DBAnonymizer.Open RESTful API Specification

43 FIWARE ArchitectureDescription Security Optional Security Enablers SecureStorageService

You can find the content of this chapter as well in the wiki of fi-ware.

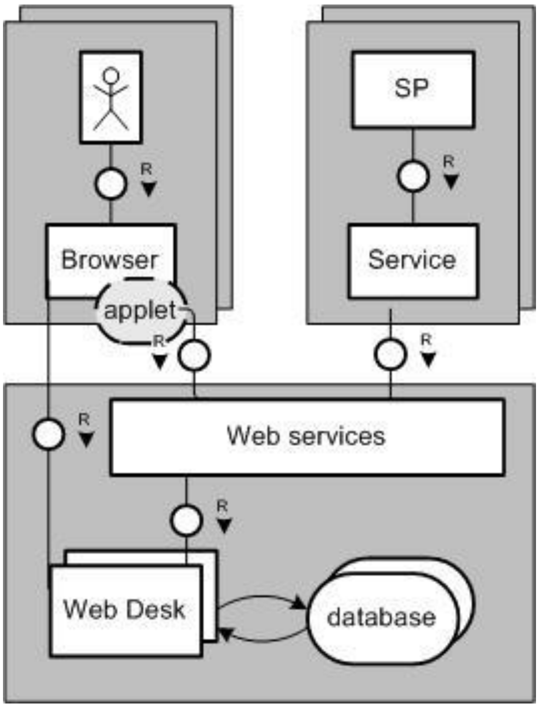
43.1 Copyright

43.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

43.3 Overview

The Secure Storage Service provides a storage for labelled (i.e. XML DSIG protected) data. It comes with an application-level filter which authorizes read access in function of the identity of the authenticated requester (for example, a service provider) and in function of the sensitivity of the data.



Glossary

SSS Secure Storage Service

SP Service Provider

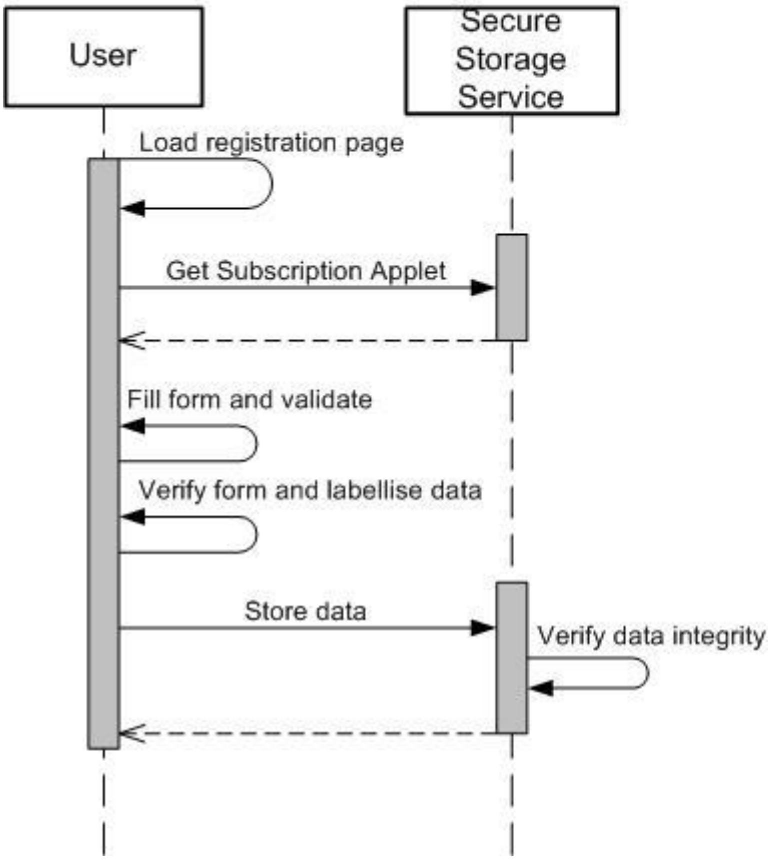
43.4 Basic Concepts

The data is labelled before being stored, i.e. it is previously protected by its owner. Moreover, the owner himself has initialised the sensitivity level of the different fields of his data (for example : mail address > private, main interest > public, job > public, etc...). Once the data are stored by SSS, the public fields (i.e. the fields that have been tagged 'public') can be read by anyone. The private one can be read by trusted service providers (SP) only. A trusted service is a service which is authenticated by a certificate which has been delivered by a dedicated Certification Authority.

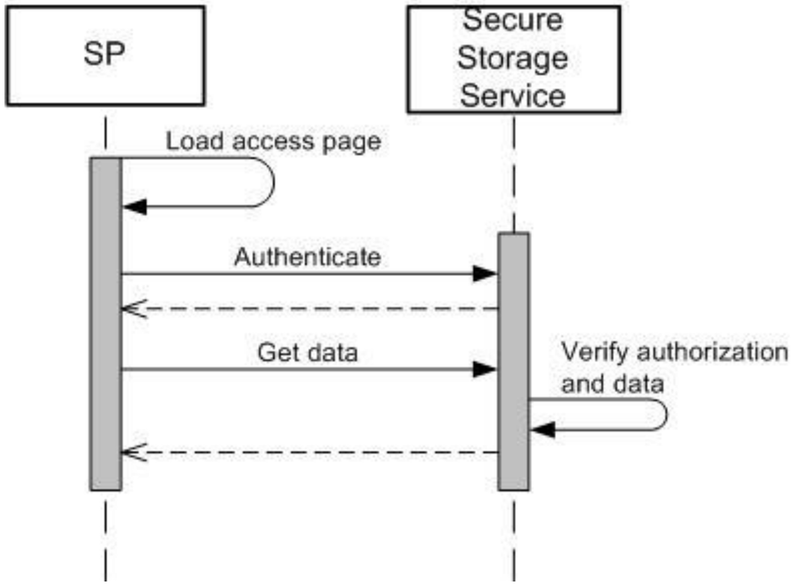
43.5 Main Interactions

The interactions between the SSS and a user or a SP are made through Web Services developed with the SOAP/WSDL technology. The user needs to connect through a web page which downloads an signed applet on the secured Web Desk. This applet labellizes the data according to the level of sensitivity and sends the data to the SSS via the WS available. The SP, authenticated and authorized, accesses the data in the SSS via the WS available.

Sequence diagram : Saving data



Sequence diagram : Retreiving data



43.6 Basic Design Principles

The functions (WS) available for the data owners are:

- Data injection
- Data update
- Data deactivation
- Get the list of SPs who had access to the profile and when, and what usage they did of the data

The functions (WS) available for the service providers:

- Data request
- Security Attribute request

NB :

- These functions are defined according to European rules : 95/46/CE, 2002/58/CE, COM(2010) 609
- WSDL available

44 FIWARE ArchitectureDescription Security Optional Security Enablers Morphus

You can find the content of this chapter as well in the wiki of fi-ware.

44.1.1 Overview

44.1.1.1 *Description*

A malware is a program designed to have an unwanted behavior seen from the legitimate user's side. It may be used to disrupt services, organise valuable/hidden data leaks, or give access to some non-authorized security levels on a system. Malware are a very common threat, more than one 70 million of such programs are known (cf. any report by an anti-virus software company e.g. McAfee [1]). Furthermore, malware use some more and more sophisticated techniques leading to a more difficult detection. A virus such as Duqu has been revealed only six month after it has been developed (see Symantec's report on Duqu [2]).

One of the main issues in malware detection is that there is no way to characterise definitely a program by its behaviour, neither syntactically nor semantically. A malware may hide its code by means of many techniques such as encryption, self-rewriting, and so on.

Morphus is a detector of malware which takes into account some parts of the structure of programs. Given a database of malware and some input binary code, "Morphus" verifies that the binary code is not infected by one of the malware of the database. The software can be used within many scenarios. The main GE offers an access to "Morphus" through a webservice.

44.1.1.2 *Use Case*

This is a typical scenario. The user has a suspicious binary file **F** whom he would like to estimate how dangerous it is. He sends a copy to "Morphus" which provides a distance matrix to known malware. The report describes which malware occur in **F** and an estimate of the level of the infection. Morphus provides two options of analysis, static or dynamic. A static analysis is fast but not very precise, a dynamic analysis is finer but less efficient.

44.1.2 Basic Concepts

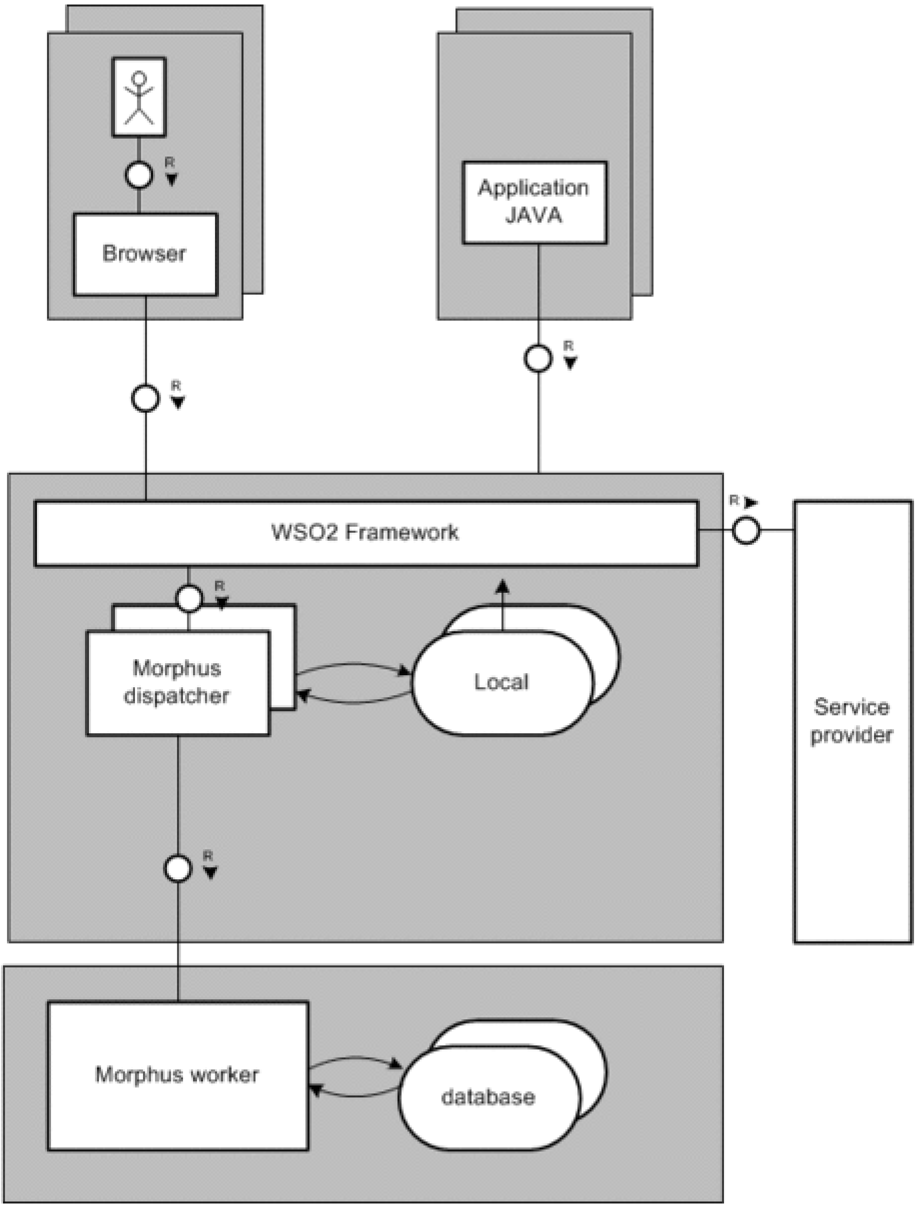
Morphus reads input binary files and extract from them signatures. Signatures are composed of sites, that is abstract descriptions of the behaviour of the input program. Binary files are only compared according to their signatures through their sites. The signatures depend on some theoretical options such as dynamic/static analysis, or some security thresholds. The system provides currently only the static/dynamic choice. We let for further release the other options. Morphus takes into account a white list database, that is a list of known and safe signatures. Usually, it contains basic operating system services.

Answers are given either as SANE/INFECTED or by a distance matrix to malware database. The second option offers a finer analysis of suspicious cases.

44.1.3 Main Interactions

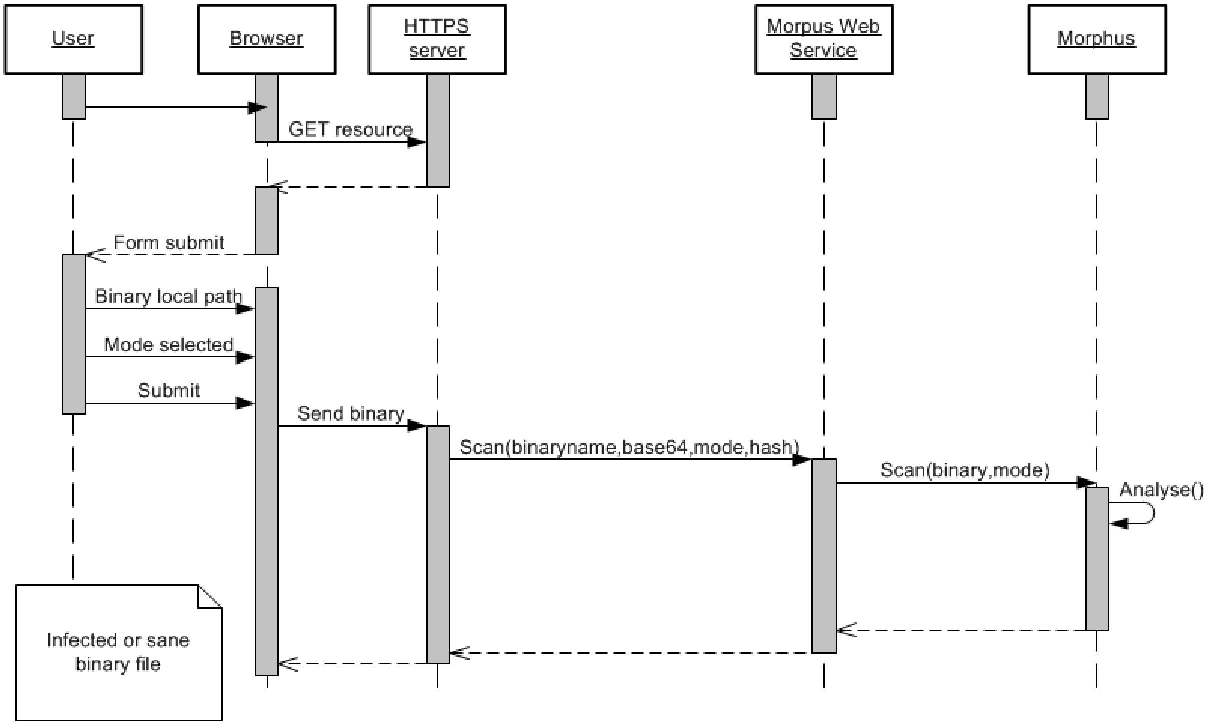
The Morphus software is available either as a web-service or through a direct connection to the website.

44.1.3.1 *Architecture*

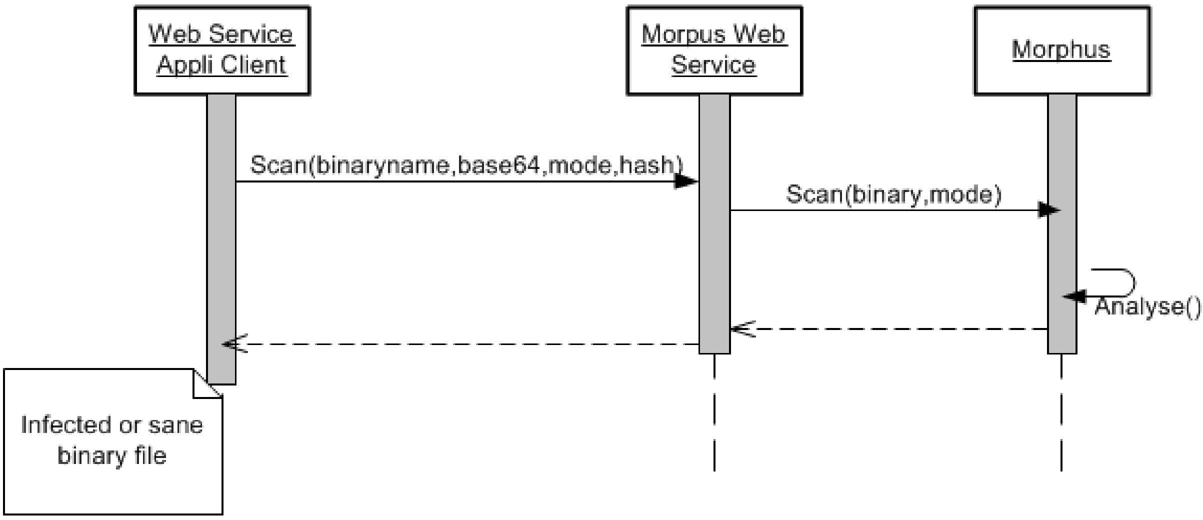


44.1.3.2 *Sequence diagrams*

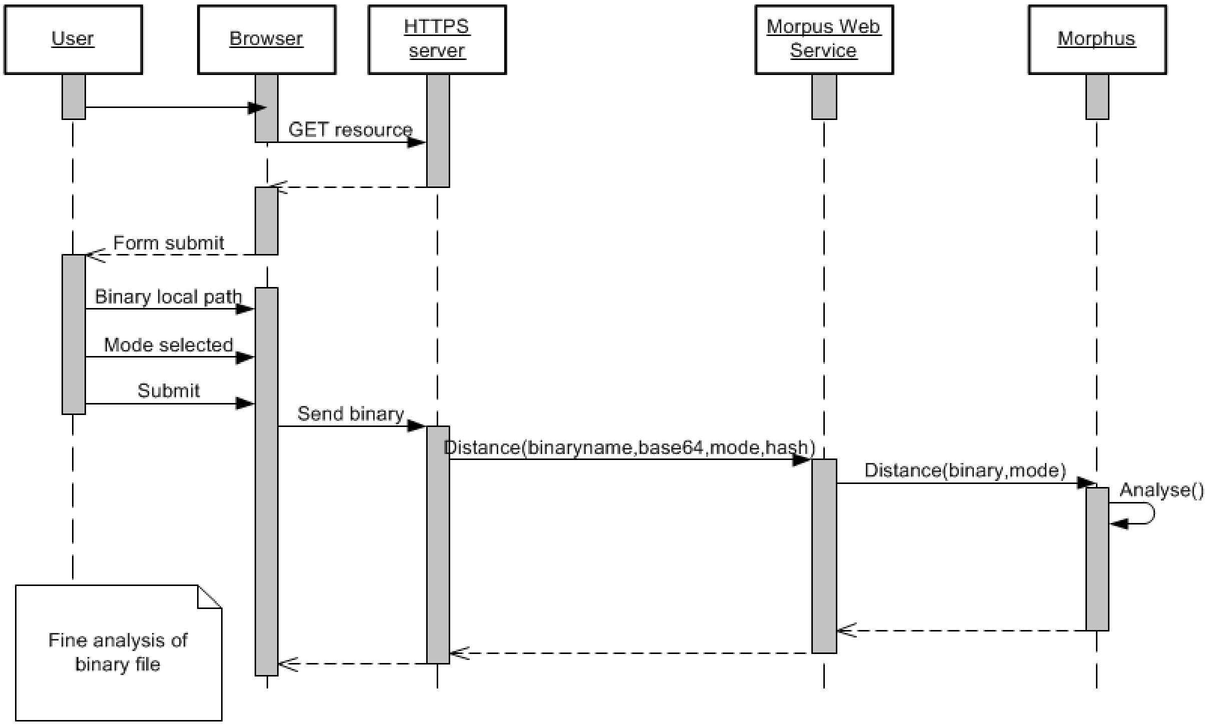
Sequence diagram : binary scan from browser



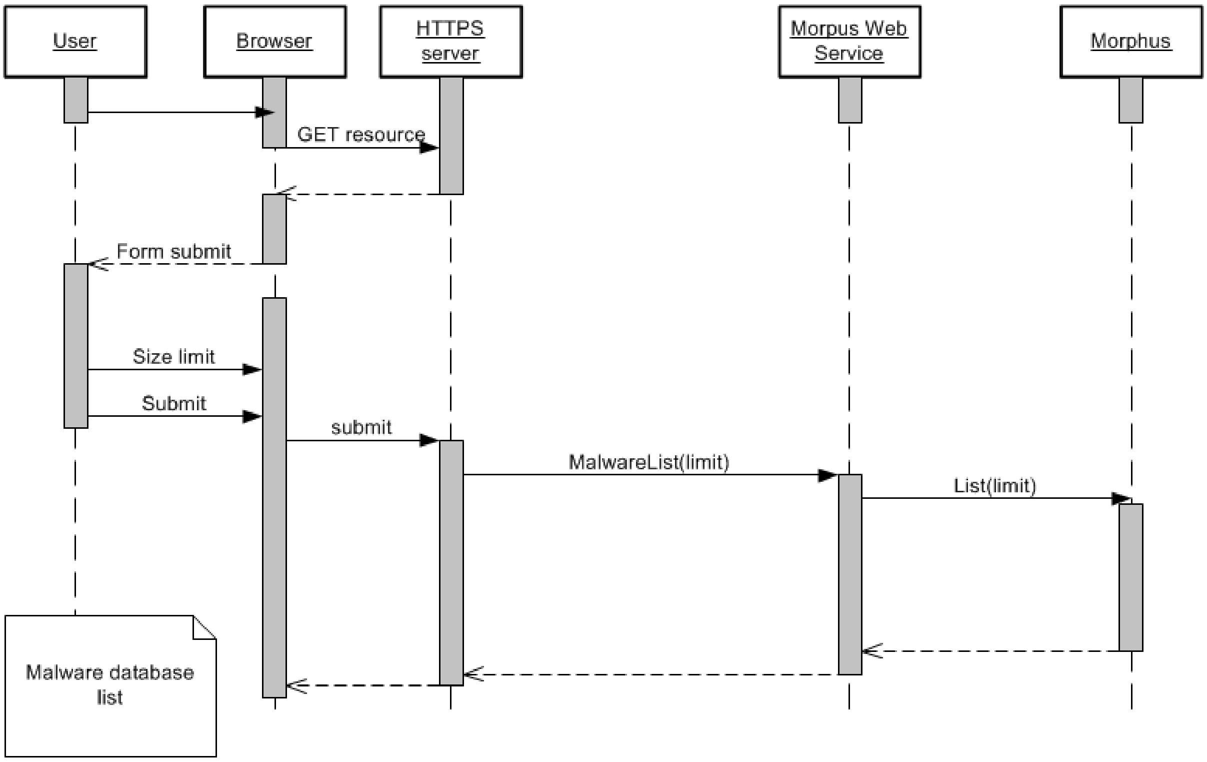
Sequence diagram : binary scan from web service application client



Sequence diagram : binary distance evaluation from browser



Sequence diagram : consult malwares list from browser



44.1.3.3 *Public API*

Description

End-user applications send requests in order to submit a binary file for evaluation to determine if it is sane or infected. Additionally, it is possible to list all recognised malwares contained in the database.

Operations

Scan a binary file

- SOAP action: urn:Scan
- Operation type: *Request-response*
- Input type: *ScanWrapper*

Data type *ScanWrapper* is composed by a complex type (*DataRequest*) that have itself 4 parameters:

1- filename: name of the binary file to scan

2- binaryData: binary file body transformed into base64 format

3- mode: scan mode switch between static (default) and dynamic

4- sha256: binary file hash in SHA-256 format

```
  <xs:element name="ScanWrapper">
    <xs:complexType>
       <xs:sequence>
          <xs:element minOccurs="0" name="Datas"
type="ns1:DataRequest"/>
       </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="DataRequest">
    <xs:sequence>
       <xs:element minOccurs="0" name="filename" nillable="true"
type="xs:string"/>
       <xs:element maxOccurs="unbounded" minOccurs="0"
name="binaryData" nillable="true" type="xmime:base64Binary"/>
       <xs:element minOccurs="0" name="mode" nillable="true"
type="xs:string"/>
       <xs:element minOccurs="0" name="sha256" nillable="true"
type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
```

- Output type: *ResponseWrapper*

Data type *ResponseWrapper* is composed by 1 parameter:

1- result: Will indicate INFECTED for an infected binary file, SANE otherwise

```
  <xs:element name="ResponseWrapper">
      <xs:complexType>
        <xs:sequence>
           <xs:element minOccurs="0" name="result" nillable="true"
type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
  </xs:element>
```

This action allows to submit a binary file to be scanned by Morphus, which will respond with either INFECTED for an infected binary file, or SANE otherwise.

Distance of a binary file

- SOAP action: urn:Distance
- Operation type: *Request-response*
- Input type: *DistanceWrapper*

Data type *DistanceWrapper* is composed by a complex type (*DataRequest*) that have itself 4 parameters:

1- filename: name of the binary file to scan

2- binaryData: binary file body transformed into base64 format

3- mode: scan mode switch between static (default) and dynamic

4- sha256: binary file hash in SHA-256 format

```
  <xs:element name="DistanceWrapper">
    <xs:complexType>
       <xs:sequence>
          <xs:element minOccurs="0" name="Datas"
type="ns1:DataRequest"/>
       </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="DataRequest">
    <xs:sequence>
       <xs:element minOccurs="0" name="filename" nillable="true"
type="xs:string"/>
       <xs:element maxOccurs="unbounded" minOccurs="0"
name="binaryData" nillable="true" type="xmime:base64Binary"/>
       <xs:element minOccurs="0" name="mode" nillable="true"
type="xs:string"/>
       <xs:element minOccurs="0" name="sha256" nillable="true"
type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
```

- Output type: *ResponseWrapper*

Data type *ResponseWrapper* is composed by 1 parameter:

1- result: Will indicate the distance from malwares that are already in database

```
  <xs:element name="ResponseWrapper">
      <xs:complexType>
        <xs:sequence>
           <xs:element minOccurs="0" name="result" nillable="true"
type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
  </xs:element>
```

This action also submits a binary to the scanner, but in this case, Morphus will reply with the distance from malwares that are already in database.

(example: Backdoor.Win32.Hupigon.bto: 100.00% Backdoor.Win32.Hupigon.bto, 59.33% Backdoor.Win32.Hupigon.bhes, 6.57% Packed.Win32.CPEX-based.e)

List malware database

- SOAP action: urn:MalwareList
- Operation type: *Request-response*
- Input type: *MalwareListRequest*

Data type *MalwareListRequest* is composed by 1 parameter:

1- limit: maximum malware names in the list (0 for unlimited)

```
    <xs:element name="MalwareListRequest">
      <xs:complexType>
         <xs:sequence>
            <xs:element minOccurs="0" name="limit" nillable="true"
type="xs:string"/>
         </xs:sequence>
      </xs:complexType>
    </xs:element>
```

- Output type: *ResponseWrapper*

Data type ResponseWrapper is composed by 1 parameter:

1- result: Will indicate the malwares list

```
  <xs:element name="ResponseWrapper">
      <xs:complexType>
        <xs:sequence>
           <xs:element minOccurs="0" name="result" nillable="true"
type="xs:string"/>
        </xs:sequence>
```

```
      </xs:complexType>
  </xs:element>
```

This action allows to list the names of malwares that are in the database.

44.1.4 Basic Design Principles

44.1.4.1 *Direct submission through a browser*

- Once authenticated, a normal user can submit a binary file by filling a form (local binary path, scan or distance action, mode between static and dynamic). The result is directly displayed in the browser.

44.1.4.2 *Web Services client application*

- Client application submits a binary file through scan or distance web service and waits until morphus-engine returns the result (a boolean or a distance vector).
- Local binary file is transmitted to server via MTOM (Message Transmission Optimization Mechanism)

44.1.5 Reference

[1] Mc Afee, Threat Quarterly report. http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q3-2011.pdf

[2] Symantec, The Precursor to the Next Stuxnet http://www.symantec.com/connect/w32_duqu_precursor_next_stuxnet

[3] Isabelle Gnaedig, Matthieu Kaczmarek, Daniel Reynaud, Stéphane Wloka. Unconditional self-modifying code elimination with dynamic compiler optimizations, 5th International Malicious and Unwanted Software Conference (Malware 2010)

[4] Guillaume Bonfante, Jean-Yves Marion, Daniel Reynaud. A Computability Perspective on Self-Modifying Programs,7th IEEE International Conference on Software Engineering and Formal Methods.

[5] Jean-Yves Marion, Daniel Reynaud. Dynamic Binary Instrumentation for Deobfuscation and Unpacking, presentation at the In-Depth Security Conference Europe 2009inria-00330022, version 1

[6] Guillaume Bonfante, Matthieu Kaczmarek, Jean-Yves Marion. Architecture of a Morphological Malware Detector, Journal in Computer Virology 5, 3 (2009).

[7] Guillaume Bonfante, Matthieu Kaczmarek, Jean-Yves Marion. Morphological Detection of Malware, 3rd International Malicious and Unwanted Software Conference (Malware 2008)

45 Interface to Networks and Devices (I2ND) Architecture

You can find the content of this chapter as well in the wiki of fi-ware.

45.1.1 Introduction

I2ND will define an enabler space for providing Generic Enablers (GEs) to run an open and standardised network infrastructure. The infrastructure will have to deal with highly sophisticated terminals, as well as with highly sophisticated proxies, on one side, and with the network operator infrastructure on the other side. This latter will be implemented by physical network nodes, which typically will be under direct control of an operator, or the node functionality will be virtualised – in this case the I2ND functionality can be accessed by further potential providers, like virtual operators.

45.1.2 Architecture Overview

The I2ND architecture covers four Generic Enablers (GEs). The four GEs will have interfaces and APIs according to their underlying technologies:

- CDI (Connected Device Interface) towards the Connected Devices. These devices include, but are not limited to, mobile terminals, tablets, set top boxes and media phones, and will have features such as remote access from a control environment, exposure of own functionality (device status, sensors, etc).
- CE (Cloud Edge) towards the Cloud Proxies. Cloud Proxies are gateways, which will connect and control a set-up of nodes towards the Internet or/and an operator network. The nodes might be either accessible or not accessible from the outside networks.
- NETIC (NETwork Information and Control) towards Open Networks. Open Networks are following the idea of flow based controlled networks, and can be used for virtualisation of networks.
- S3C (Service Capability, Connectivity and Control) towards Underlying Networks. The underlying networks are following standards such as Next Generation Networks (NGNs) or Next Generation Mobile Networks (NGMNs). In the case of the S3C specified in I2ND the baseline underlying network will be the Evolved Packet Core (EPC) by 3GPP.

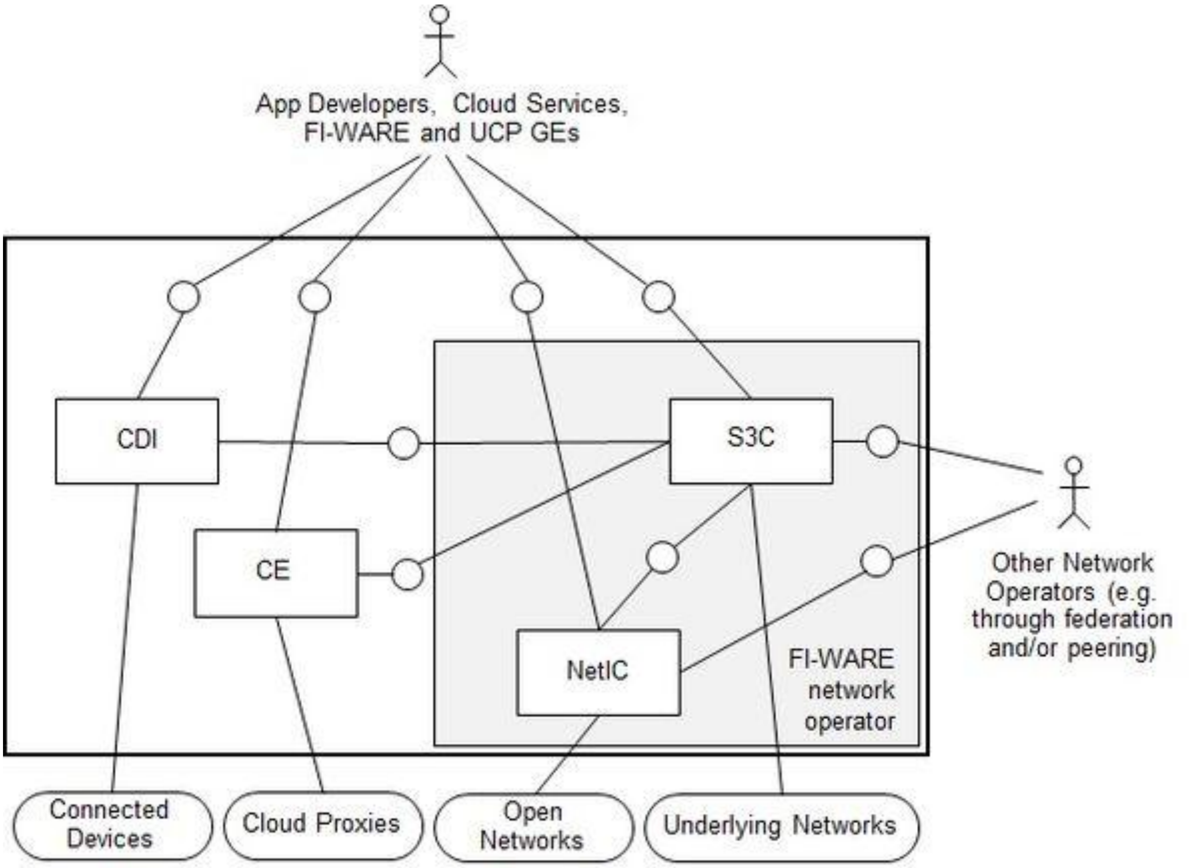


Figure 7.general.1: I2ND Architecture

Each of the GEs of I2ND will have specific interfaces towards Application Developers, Cloud Services, FI-WARE and other Use Case GEs and projects. The respective interfaces will be described in the GE sections.

The GE S3C is the central point of the I2ND architecture. I2ND develops an enabling environment which can be used by network operators. Together with NetIC, both GEs will build the environment of an operator, which might even be a virtual operator. S3C can be seen as the GE to run and steer the network infrastructure.

A special role in I2ND is devoted to the interface towards other operators. Since the Future Internet infrastructure will follow the Internet philosophy of a multi domain/end-to-end path through different network operators, it is necessary to interface with other operators. In I2ND the interface description provided by the ETICS project ([1] [2]) is adopted. The interface between NetIC and other operators promotes to the future of network interface virtualisation down to the network layer. This interface might be restricted by the owner of the network infrastructure, thus offering a sub-set of whole interactions.

The interfacing between S3C and CDI will provide status and control information exchange of the device and remote control capabilities. A respective interface template will be used.

Cloud proxies might be part of an operator infrastructure. Therefore it is necessary to have access to these network nodes through a standardised interface.

More details about the interfacing mechanisms among the GEs in the I2ND chapter are given in the wiki pages describing the respective GE architecture.

45.1.3 Architecture description of GEs

The following pages describe the architecture of the Generic Enablers defined in the I2ND chapter. **Please note that such architectures, in particular those related to the network-orientes GEs (i.e. NetIC and S3C) are still preliminary and might be subject to significant changes.**

- FIWARE.ArchitectureDescription.I2ND.CDI
- FIWARE.ArchitectureDescription.I2ND.CE
- FIWARE.ArchitectureDescription.I2ND.NetIC
- FIWARE.ArchitectureDescription.I2ND.S3C

45.1.4 Interfaces to other Chapters

Interaction between the I2ND chapter and other chapters within FI-WARE is necessary for realization of a complete Future Internet Core Platform. The I2ND chapter interacts with four other chapters within FI-WARE, in the following the relevant chapters are highlighted.

45.1.4.1 *Cloud Hosting*

Cloud Hosting Architecture

An interaction between cloud Hosting GEs and the S3C GE of I2ND is expected to handle SLA management. This interface type is defined by Telemangement Forum and the outcomes of the EU-project ETICS will be also exploited.

45.1.4.2 *IoT*

Internet of Things (IoT) Services Enablement Architecture

An information exchange is expected to happen between S3C and IoT GEs, this interface exchanges messages in XML like format (under specification). Device Management interfaces are also expected between CDI and IoT GEs. The interface specification is currently under definition.

45.1.4.3 *Data*

Data/Context Management Architecture

An interaction to exchange information is expected between S3C and Data GEs, this interface will manage messages in XML like format (under specification).

45.1.4.4 *Security*

Security Architecture

The main interaction with Security GEs concerns the Identity Management. An interface will be provided (based on XML-like format) to interact with the S3C GE of I2ND. Support of client security is also an interaction involving CDI with Security GEs. The specification of such interface is currently subject to definition.

45.1.5 References

[1] ETICS Deliverable D4.2: Economics and Technologies for Inter-Carrier Services https://www.ict-etics.eu/fileadmin/documents/publications/deliverables/D4_2_ETICS_architecture_and_functional_entities_high_level_design.pdf

[2] ETICS Deliverable D4.3: Revision of ETICS Architecture and Functional Entities https://bscw.ict-etics.eu/pub/bscw.cgi/d37005/ETICS_D4.3_v1.0.pdf

46 FIWARE ArchitectureDescription I2ND CDI

You can find the content of this chapter as well in the wiki of fi-ware.

46.1 Copyright

46.2 Legal Notice

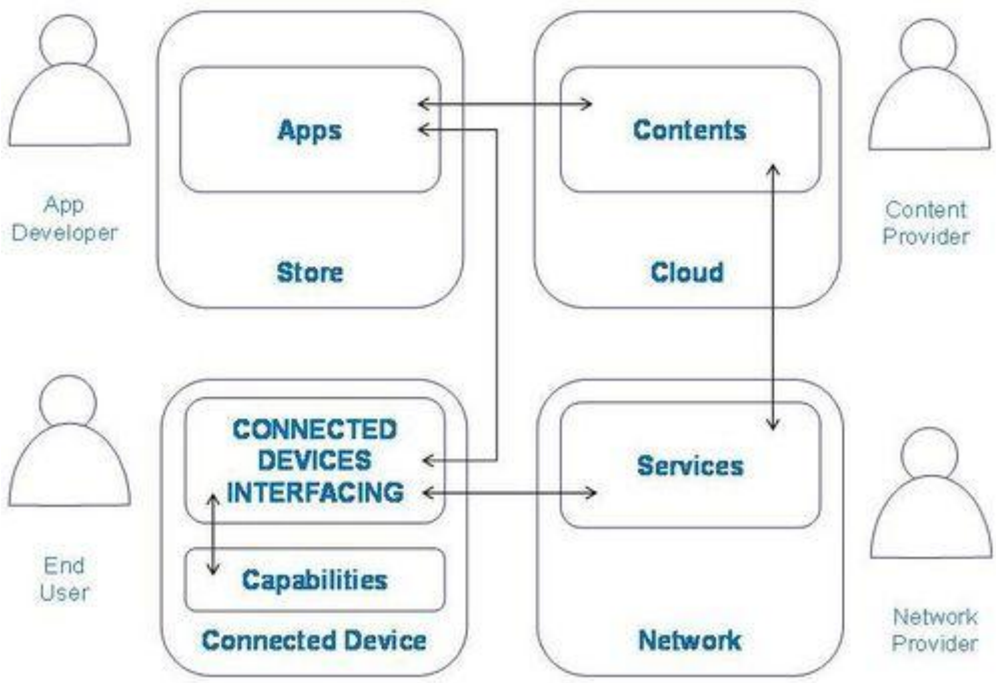
Please check the following Legal Notice to understand the rights to use these specifications.

46.3 Overview

The Connected Devices Interface (CDI) Generic Enabler (GE) provides to FI-WARE chapters and Use Case Projects applications and services, the means to detect and to optimally exploit capabilities and aspects about the status of connected devices, through the implementation of interfaces and Application Programming Interfaces (APIs) towards device features.

The term “connected devices” refers to a broad range of networked electronic components, including Handsets (cellular phones, smart phones), Tablets, Media phones, Smart TVs, Set-Top-Boxes, In-Vehicle Infotainment, Information kiosks, each being able to connect to a communication network. The Connected Devices Interface (CDI) Generic Enabler (GE) will provide, to FI-WARE chapters and Use Case Projects applications and services, the means to detect and to optimally exploit capabilities and aspects about the status of devices, through the implementation of interfaces and application program interfaces (APIs) towards device features.

The CDI is a GE located in the connected device, ad shown in the figure below, with the aim to provide to the network services and the developer’s applications common API to exploit the device capabilities, resources, contextual information and contents.



Connected Devices Interfacing (CDI) target usage

The CDI GE open specifications will expose API to access connected device features and provide them to applications which will execute on the device (On Device Access), including as an example battery power, network status, location systems, quality of service, media capabilities, phone available features and sensors, profile information and status information. Moreover, additional API will detail how connected devices can be accessed via a Remote Management service (via a REST API).

Also, the network operator can exploit the CDI API to remotely manage the parameters used by the device for establishing the connectivity to the network such as policies for access network discovery and selection, attachment and connectivity policies as well as management of the Quality of Service control and in-device routing policies in order to enable the mobile device to best match and react on its own to contextual situation (e.g. coexistence of WiFi and 3G coverage, usage of another connection over another access network for better QoS etc.). The result is that end users can enjoy their favorite contents, anywhere and anytime, using their preferred apps across heterogeneous networks and connected devices.

This means that same applications and network services can be used consistently over dissimilar connected devices, if they are equipped with the CDI. Moreover, the apps provided by FI-WARE application developers will be able to exploit the capabilities and features the specific device platforms provide. This allows cloud hosted application services for example, to render their interfaces to make best use of individual devices and their relative connectivity. To realise this, the CDI GE will offer a uniform and standardized access and easy portability across multiple device classes of the features mentioned.

To look at this in terms of practical benefits, a hypothetical scenario may involve a cloud-hosted media rich service which is accessed by many consumers across fixed or mobile terminal devices. The service provider will want to ensure that all consumers get the best possible experience, as a function of their connectivity, and the display and processing power (noting remaining battery power) of their device. By programmatically enabling the service with the ability to detect relevant details of the client device and its connectivity, decisions can be made at run-time in terms of selecting different levels of media 'richness' including sizing, resolution, 2d or 3d, etc., for individual users

Another example might relate to location-based services, where the location of the device (shopping mall), might be a useful indication of the customer's intention, and suggest useful sidebar links in the form of advertising offers. Yet another might be the availability of triaxial accelerometers on the device and a user profile indicating a preference for this as a user input modality, which could be comprehended directly into the rendered UI configuration of a service.

46.4 Basic Concepts

A major challenge to be faced when defining the CDI GE specifications, is to provide common interfaces to device functionality addressed to application developers. It is recognised that the extreme fragmentation of platforms adopted for connected devices, including a variety of different OSs and programming languages, is introducing several troubles to develop once for all the application and make it run on all such devices. While applications development can exploit different programming paradigms (native versus interpreted versus scripting applications), the CDI GE specifications attempt to find a convergence point at least on the interfaces. The basic assumption for the definition of CDI interfaces is that they will be defined as independent as possible from specific technology implementation and programming paradigm, thus creating a layer on top of the technology dependent architecture of the devices, which enables communicating with the applications and the network services.

Supporting different devices and different form factors is central to CDI, therefore the exposed interfaces provide an abundance of features and platform discovery functionalities.

46.4.1 Architecture of I2ND

The CDI is unique in FI-WARE as it creates an execution environment for applications on connected devices, and supports the inter-GE communication throughout FI-WARE. There are three main interfaces presented by the CDI Generic Enabler. These are:

1. ***On Device Interface***

   This interface supports applications (apps) running on the connected device. It presents a uniform set of functionalities across all connected devices, regardless of form factor.

2. ***Remote Management Interface***

   The remote management interface presents an externally accessible interface which supports other Generic Enablers. It provides the following functionality:

   - **Push Notification Framework**

     It provides a push notification framework where external GEs can signal connected devices with information and data

   - **Remote Device Management**

     Configuring device and network settings

   Main purposes of the remote device management interface is to provide an access to CDI to manage:

- Installation and management of configuration information (both initial and operational information) in devices.
- Update of firmware and software
- Retrieval of management information from devices
- Processing events and alarms generated by devices
- Perform diagnostic tests
- Controlling device capabilities

The managed information includes e.g. configuration settings, operating parameters, software and parameters, application settings, user preferences.

3. ***Mobility Manager Interface***

The mobility manager interface connects to the S3C GE in order to:

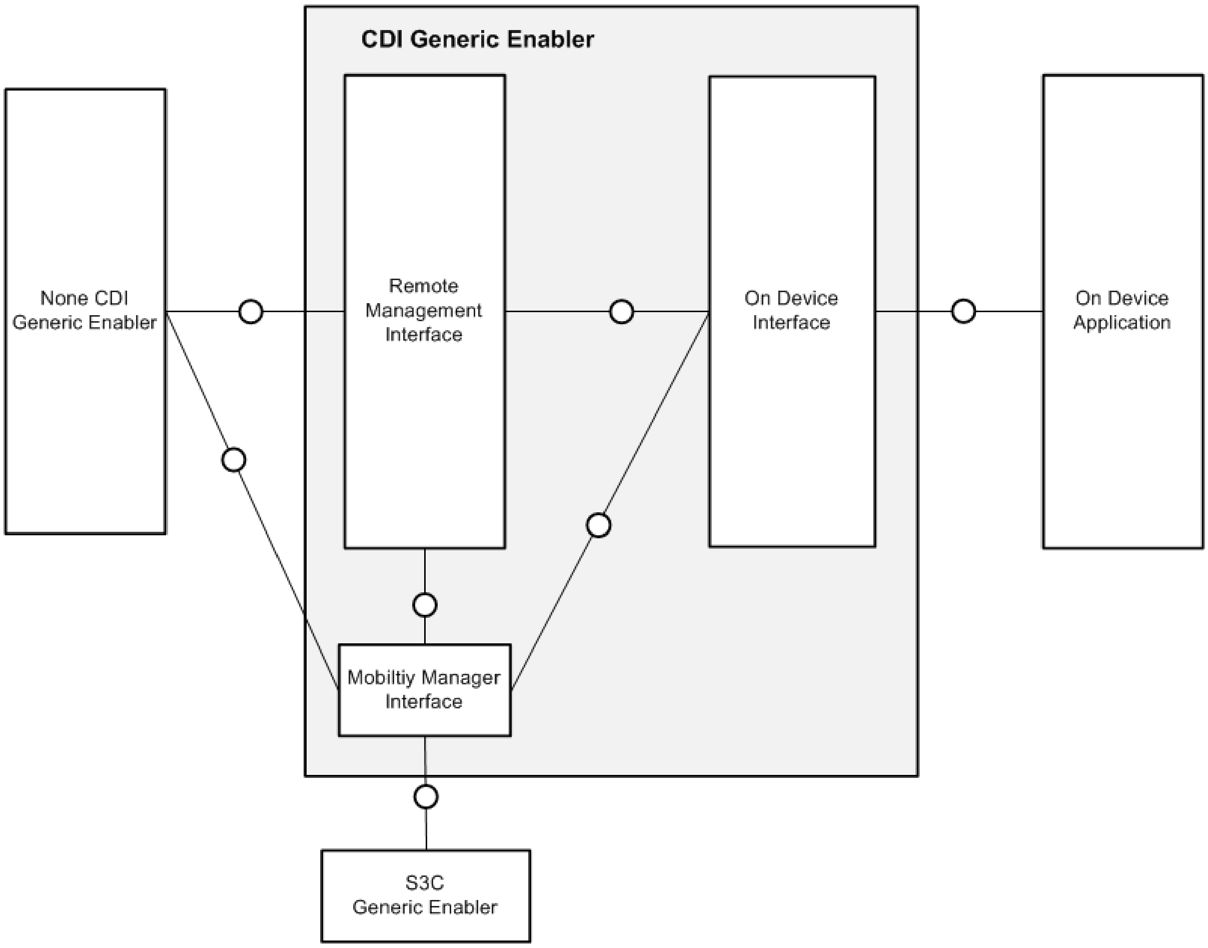
- **Receive network policies**

  Based on the available network interfaces the S3C GE can transmit to the devices policy rules describing when to connect to which network interface.

- **Request network information**

  In order to provide a specific QoS pertaining connectivity the mobility manager interface is used to request information about available network interfaces that meet certain connectivity requirements.

The high level architecture of CDI is represented by the following FMC model.



Note: The "None CDI Generic Enabler" represents any other GE (Generic Enabler) or 3rd party server or cloud hosted service which needs to access the remote interfaces provided by the CDI GE. It is included purely for illustrative purposes only.

46.5 Main Interactions

For each of the three main interfaces specified for the CDI GE, the following sections will provide insights about the interactions a client might activate to access a range of functional information.

46.5.1 On Device Interface

The following is a detailed description of the functionality supported by the interfaces accessed through the ***On Device Interface***. The functionality is grouped together based on aligned EPICs. Each group of functionality is further subdivided into Functional Blocks.

46.5.1.1 *Device Sensors*

All connected devices targeted in FI-WARE contain a range of sensors which can provide useful functionality for application developers. CDI will support the following four sensor types:

- Camera
- Microphone
- Geo-Location
- Device Orientation & Accelerometer

The interaction with the sensors are broken out into functional descriptions below. A description of the interfaces provided to the developer is also supplied. These functional blocks are available on all devices which support them. Not all devices may support these sensors, therefore the specification of a sensor discovery functionality is also supplied.

- **Camera** The camera interface allows the developer to capture images and video. Devices often have more than one camera. The interface enumerates the available cameras and allows the developer to select the camera to use. For developer convenience the interface defines a default, implicit camera for use when an explicit camera is not specified by the developer.
  User authentication: The user will be prompted for permission the first time the application requests the use of camera. If the user accepts that the application should be allowed to access the camera then all subsequent requests will complete without prompting.
- **Microphone**
  The developer will be able to write applications which can record audio. Devices can support more than one microphone. The interface enumerates the available microphones and allows the developer to select a microphone to use. For developer convenience the interface defines a default implicit microphone for use when an explicit microphone is not specified.
  User authentication: The user will be prompted for permission the first time the application attempts to record audio. If the user accepts that the application should be allowed to access to the microphone then all subsequent access attempts will complete without prompting.
- **Geo-Location**
  The application developer is able to retrieve information about the devices location. This includes the longitude, latitude, altitude, and speed. This API provides both a polling format, where the developer can simply request the current location, and a notification format where the developer can be informed when the location or speed changes. This notification framework will provide the developer with a battery friendly method of monitoring the devices location.
  User authentication: The user will be prompted for permission the first time the application requests Geo-Location. If the user accepts that the application should be allowed to access the geo-location then all subsequent requests will complete without prompting.
  Note: Geo-Location functionality is supported in the initial release (Release 1) of the CDI GE implementation.
- **Device Orientation & Accelerometer**

  The developer will be able to capture information on the devices orientation (compass) and physical orientation; rotation on the x, y and z axis. The accelerometer information provides the developer with notification on when the orientation or rotation

changes. Again this call back / notification framework provides a battery friendly method of detecting changes in the devices position.
User authentication: There is no user authentication required for capturing the device orientation & accelerometer data.

- **Sensor Discovery**

  The developer will be able to obtain information on the sensors available to the device.

  - Camera
    Information about the camera includes, the number of cameras, camera placement (front, back, unknown) camera resolution, flash support. Additionally capture capabilities of the camera are also be provided, including:
    - What image formats are supported
    - Is video capture supported
    - What video capture formats are supported
  - Microphone
    Information on the microphone includes: the number of microphones, their placement - on device, or on a headset, encoding support - what formats are supported.
  - Boolean values exist to determine if each of the following is supported:
    - Device Orientation (Compass)
    - Accelerometer
    - Geo-Location

46.5.1.2 *Quality of Experience (QoE)*

All devices equipped with a web run time, can provide useful QoE control functionality for application developers. The QoE control functionality is provided by the QoE Engine functional block. Such functional block is able to combine explicit QoE feedbacks coming from the user, with the QoS level provided by the network. Both inputs are matched against a target QoE level to be achieved, and network resources are explicitly requested in order to approximate the target as much as possible. For what concerns interfaces, a mobile application developer will be able to instantiate a graphical object (i.e., a button), logically related to an application flow (i.e., audio, video, file transfer etc.), allowing the final user of the application to express his current degree of satisfaction (QoE) while enjoying the service. The application developer can use an interface with the following functionalities:

- **Provide QoE feedback**

  To be used by the application developer to inform the underlying QoE framework, that the user is expressing a bad QoE feedback. Such functionality must be attached to a button. Ideally the user is more unsatisfied, as the number of clicks on the button increases. QoE Engine is also able to distinguish unfair and wrong calls of the functionality, performed respectively by malicious and incapable users.

- **Start/Resume QoE monitoring**

  To be used by the application developer to inform the underlying QoE framework that a new application flow has been started or that a paused application flow has been resumed. The effect is that QoE Engine starts/restarts monitoring the flow.

- **Pause QoE monitoring**

  To be used by the application developer to inform the underlying QoE framework, that the application flow is not currently under the user attention. For instance when the window passes in background, or when a pause button is pressed.

- **Stop QoE monitoring**

  To be used by the application developer to inform the underlying QoE framework that the application flow doesn't need to be monitored anymore. For instance when the window is closed, or when a stop button is pressed.

- **Express Mean Opinion Score (MOS)**

  A developer can decide to provide the user the possibility to express a final rating for his experience. There are five possible ratings (EXCELLENT, GOOD, FAIR, POOR, BAD). The MOS rating will enrich the Context data base maintained by the QoE framework running underneath.

- **Track the realtime evolution of QoE parameters**

  The application developer has the possibility to track in realtime the evolution of QoE parameters related to the users. For instance the developer can track the Click Rate of the user and show it on the screen. Other possibilities include to track the realtime evolution of the allocated bandwidth for a given network flow.

46.5.1.3 *User Profile*

This set of interfaces provides the connected device applications to access identification and authentication procedures.

- **Access to local and personal device data**

  Each of the functional blocks which expose information PII (Personally Identifiable Information) such as photos, contacts, camera, etc. have a separate User authentication step. In this separate user authentication step the user receives a prompt requesting them to grant an application access to a functional block. Prompting the user every time an application attempts to access a PII functional block would result in multiple recurring prompts. This would make applications which access these functional blocks impossible to use. To balance the need for user awareness of an application's behaviour with the need to provide a consistent and appealing user interface, the CDI will prompt the user initially and only once. Subsequent accesses of a functional block by an application will no longer result in a user prompt.

  If the user declines an application's request to access a functional block, then the application will receive an error message informing it that it does not have permission to access the functional block.

  User authentication: Please refer to the "User authentication" descriptions provided in each of the functional blocks described.

- **Security and Sign on**

  Devices may be used by more than one person. In order to offer a secure, safe, and personalised experience it is often important to know information both the device and the currently active user. To facilitate this the CDI enforces a user identification process. Users will need to identify themselves to the CDI, once identified (signed in) the CDI will assume that this same user is continually logged in, in perpetuity. Via a prompt the user may explicitly sign out (log out). If no user is currently signed in (identified) to the CDI then the client will display a sign in page. The user's account will be protected by a password. Once logged in the user will have the ability to change this password. The password will be associated with the users device, and optionally a email address. When an incorrect password is entered an error message will be displayed. The error message will invite the user to try again. If the user forgets their password they may reset the device, loosing all information stored on it and create a new user account via the initial start up scenario.

User authentication: The CDI supports an initial start up scenario where no user is associated with a device. In this instance the user is invited to created an account on the device. Once this is done the only way the user can access CDI functionality is to ensure that they are signed on. Once signed into the CDI the user's stored credentials can then also be used to grant them access to resources stored within other generic enablers. This provides them with a SSO (Single Sign On) experience.

- **Access to User Profile Information**

  To ensure a personalised experience the developer may obtain information about the currently signed on user. This includes the users data like Gender, Date of Birth, Home address, Telephone numbers (mobile, home, work etc), Email addresses, IM details, User profile photo, Full name.

  User authentication: The user will be prompted for permission the first time the application requests to the users profile. If the user grants access then all subsequent requests will complete without prompting.

46.5.1.4 *Device Features*

Application developers need to understand the limitations and opportunities presented to their applications by the devices upon which they run. Information describing the device is presented to the application to allow the application to tailor itself accordingly. Likewise any requests by the CDI device to other GEs via a RESTFUL interface also include enough information to allow the cloud hosted component to tailor its response to the device. Therefore the functional blocks below divide this task into two segments, "On Device Profile Information" and "Off Device Profile Information".

On Device Profile Information

Applications running on the device have access to the following information:

- **Device form factor.** An boolean value for each of the following indicating that the device is: a phone (phone like - supports calling), a Tablet, a PC, a Set top box, an in-car system
- **Screen size.** The screen is described using numeric values indicating the screen size (width x height), the colour depth (bit depth), the DPI of the device (pixels per inch)
- **Available Inputs.** The available input methods are described by (boolean) values indicating Touch screen support, Hardware QWERTY keyboard, On screen keyboard, Numeric keypad (T9), Stylus support.
- **Processor Type.** Details about the processor are provided to the application. This includes an enumerated value indicating the processor family (e.g. X86, ARM, Other), an enumerated value indicating machine word size (e.g. 32bit or 64bit), an integer value representing the number of cores.

- **Available Disk (Storage) Space .** During the lifetime of almost any application the need will arise to utilise disk space, either for downloading of content from the cloud / internet or the storage or locally produced content. Storage will be described by integer values representing the total size of the disk in bytes, the total number of bytes consumed, the total number of bytes available.

  This information allow any application to adapt itself to the actual disk state of the device.
- **Connectivity .** The network connectivity of the connected device is provided to the developer. This is expressed in two forms, firstly as the available connectivity options, and secondly the currently connected (if any) connectivity options.
  - *Available* - Multiple boolean values indicate the presence of the following technologies: Wi-Fi, Bluetooth, Cellular
  - *Current* - A list of the currently connected technologies is provided. For each entry in the list of connected technologies the signal strength is expressed as a percentage. Multiple triples of one boolean, one integer and one floating point value indicating if the technology is connected (boolean), signal strength (integer) and estimated bandwidth in MB/s. Where zero MB/s is used when the technology is not connected. Applies to the following technologies: WiFi, Bluetooth, Cellular
- **Battery State.** The state of the battery can be used to tailor applications and services, allowing application to throttle their behaviour when in low power situations allowing the user to continue using their device for longer. The battery information will be provided to the application as follows:
  - A (boolean) value indicating if the device is currently charging
  - A (integer) value representing the length of time remaining before the battery is completely discharged
  - A percentage value indicating the current battery charge state
- **Media Services Support** (See "Discovering Media Features Supported")
- **Device Sensor Support** (See "Sensor Discovery")
- **Personal Data Interface Availability** (see "Personal Data Interface Availability")
- **Messaging Interface Availability** (see "Messaging Interface Availability") **User authentication:** None

Off Device Profile Information

The following details will be provided to other GEs or cloud devices when requesting services. For complete details see above.

- **Current connectivity**
- **Media Services Support For Consumption**
- **Processor Type**
- **Screen size**
- **Device form factor**

  User authentication: None

46.5.1.5 *Media Services*

In order to offer the best possible media experience on the device it is important to understand what media formats and codecs the device can support. Usually devices can play back more codecs than they can record or produce. This explains the following function blocks.

- **Discovering Media Features Supported For Consumption**

  An array of mime types will be used to describe the media formats supported for media consumption. This will be similar to the standard HTTP accept header. **User authentication:** None

- **Discovering Media Features Supported For Production**

  An array of mime types will be used to describe the media formats supported for media production.

  User authentication: None

46.5.1.6 *Personal Data Services*

Getting access to personal data on a device opens up a wide range of possible applications. It also allows applications and devices to integrate seamlessly with cloud hosted services. To support this, and the epic above the following functional blocks have been created.

- **Contacts**
  For any communication devices from a PC running an email client to a mobile phone supporting voice calls an address book or storage of contacts and their details is vitally important. This functional block supports this by enabling an application to search for contacts,read a contact's details, update a contact's details (change, add and remove fields on a contact) , delete a contact entry

  User authentication: The user will be prompted for permission the first time the application requests use of this feature. If approved all subsequent request will complete without prompting to the user.

- **Calendar**
  Many devices also act as a personal organiser for the user and being able to access and interact with the daily schedule provides scope for a wide range of personalised experiences. CDI provides an interface to the developer allowing them to both read and write calendar entries. However devices can support more than one calendar, e.g. the user's calendar from work, and a calendar for home. CDI will enumerate all the available calendars on the device and provide context information for each one, allowing the develop to select the most appropriate calendar for editing.

  User authentication: The user will be prompted for permission the first time the application requests use of this feature. If approved all subsequent request will complete without prompting to the user.

- **Gallery**
  Galleries offer access to the collection of physical visual and audio media stored on the device, this includes both images, and video clips. The gallery interface allows an application to search for media based on encoding format or date produced (date range), sort media based on date produced or media format (either ascending or descending), present a UI based "picker" based on a search and sort order

description to the user, an array of URIs representing the media elements selected by the user is returned to the application.

User authentication: The user will be prompted for permission the first time the application requests use of this feature. If approved all subsequent request will complete without prompting to the user.

- **File System Access**

  All devices have a file system, or file like system access to the file system facilitates the creation of a wide range of applications and services. This functional block allows an application to read the contents of directories, search directories for files, navigate directories, rename files, read and update file attributes, read and write a file's binary data.

  User authentication: The user will be prompted for permission the first time the application requests use of this feature. If approved all subsequent request will complete without prompting to the user.

- **Personal Data Interface Availability**

  Not all devices will support all of the functionality provided by Personal Data Services. Therefore a discovery interface will be provided allowing an application to query the device for supported functional blocks. The supported functional blocks will be expressed as a collection of (boolean) values including contact support, calendar support, gallery support and File System Access.

  User authentication: N/A.

46.5.1.7 *Phone*

It is important that applications running a device can leverage the power and facilities offered by the device. An application running on a phone should know it is running on the phone and be able to access phone status and/or operations. The functionality presented in the functional blocks below is available when the boolean value indicating that the device is a phone is set to true (on). The blocks break the functionality up into two segments, monitoring device status and using phone operations.

- **Phone Status**

  Before invoking any functionality it is important that the application is able to assess the current state of the device. This can be implemented e.g. through an enumerated value, which is used to identify the phone state as either Engaged, Ringing, Dialing, InCall or Idle. The interface also allows the application to monitor the device for changes in state. If the device is currently in the Engaged, Ringing, Dialling or InCall state then it will also be able to query the device for the destination phone number of the call.

  User authentication: The user will be prompted for permission the first time the application requests use of this feature. If approved all subsequent request will complete without prompting to the user.

- **Phone Operations**

  The following functionality is supported by this function block:

  - Make Call: the application may invoke a new call
  - End Call: the application may end a call which is in progress

- Answer Call: the application may answer an incoming call
- Reject Call: the application may reject an incoming call

User authentication: The user will be prompted for permission the first time the application requests use of this feature. If approved all subsequent request will complete without prompting to the user.

46.5.1.8 *Messaging*

Increasingly devices come equipped as standard with communication technology. It makes sense to offer that functionality to application developers where possible. This group of functionality deals with Email, SMS and MSS messaging.

- **Email**
  This functional block will allow the application to compose emails on behalf of the user.

  User authentication: The application may compose and address the email, however sending the email will require direct input from the user.

- **SMS**
  This functional block will allow the application to compose SMSs on behalf of the user.

  User authentication: The application may compose and address the SMS, however sending the SMS will require direct input from the user.

- **MMS**
  This functional block will allow the application to compose MMSs on behalf of the user.

  User authentication: The application may compose and address the MMS, however sending the MMS will require direct input from the user.

- **Messaging Interface Discovery**

  Not all devices will support all of the functionality provided by the messaging interface. Therefore a discovery interface will be provided allowing an application to query the device for supported functional blocks. The supported functional blocks will be expresses as a collection of (boolean) values for Email, SMS and MMS.

  User authentication: N/A.

46.5.1.9 *Device Connectivity*

Providing access to device connectivity allows a developer to create applications which can adapt to both the available band width and available access points. The following functional blocks allow a developer to discover the types of connectivity which are available on the current device. While cellular network connectivity is usually fixed by the network operator the connection to WiFi Access points is typically at a user description. The functional blocks below provide additional functionality over WiFi interfaces to allow the creation of applications which can help assist the end user.

- **Discovering Connectivity**

  See Device Features - Connectivity (above).

- **Detecting Current Connection Settings**

See current connectivity under the device features group (above).

- **WiFi Interfaces**

  The application is able to see a list of currently available to see:

  - Wireless access points, their SSIDs and BSSIDs along with signal strength
  - The currently connected access point, its SSID, BSSID and signal strength
  - The application is able to disconnect from a currently connected access point or request connection to a new access point

  User authentication: The user will be prompted for permission the first time the application requests use of this feature. If approved all subsequent request will complete without prompting to the user.

46.5.2 Remote Management Interface

The CDI remote management interface implements a client according to to Open Mobile Alliance – Device Management (OMA DM) specification. OMA DM Working Group (http://www.openmobilealliance.org/technical/DM.aspx)specifies protocols and mechanisms to perform management of devices. The evolution to OMA DM 2.0 is based on RESTful architecture, which is adopted for CDI interface specification. This makes the CDI interface more uniform with the remaining FI-WARE architecture, and provides platform neutral protocol to allow servers to remotely manage devices, by operating over a HTTP transport and notification protocol.

This interfacing functionality is not available in the first release of the FI-WARE platform, and will be detailed in the next period of project activity.

Remote Management implements the following functions:

- **Device Configuration**:

  Provide a common mechanism to remotely configure one of more settings of the device.

- **Application Management**:

  Provide a common mechanism to remotely install and setting parameters for applications on the device.

- **Firmware Update**:

  Provide a common mechanism to remotely verify, download and install a firmware of the device.

46.5.3 Mobility Manager Interface

The main purposes of the Mobility Manager interface is to provide an access network selection function that empowers the CDI device to select network interfaces according to the connectivity requirements of CDI applications or by network operator policies. This access network selection is transparent to the applications.

- **S3C Interface**

  The Mobility Manager component uses and extends the OpenEPC platform, which is part of the S3C GE, addressing mobile devices with customized and optimized communication services including the network status information from the device itself as well as the information which is received from the network operator including the available resources in the vicinity of the mobile device. Therefore the Mobility Manager provides a network interface to the S3C GE for the following functions:

  - **Requirements on QoS**

    Communication with the core network platform for receiving extended information on the vicinity environment. This mechanism allows communicating requirements for the QoS/QoE of the connected devices.

  - **Routing Policies**

    Receive routing policies for policy based access network selection. This mechanism empowers the mobile device with the vicinity network status including access network discovery information, inter-system mobility and routing policies enabling the forwarding of the data traffic from the mobile device to the appropriate access networks.

  - **Vicinity environment information**

    Vicinity network information format to be transmitted from the network operator to the devices. This provides a mechanism for dynamically aggregating the information on the network status of the device including static and dynamic local information on the momentary active and potentially available device interfaces and a common presentation mechanism of this information towards the applications.

- **Interface to Applications**

  The Mobility Manager also exposes an internal API to CDI applications that provides the following functionality:

  - **Request QoS level**

    An application can request a specific level of QoS. This allows the application developer to adapt the connectivity of the device to the current needs of the running applications.

  - **QoS Notifications**

    An application can register as a listener for a specific kind of QoS level. The application then gets notified by the Mobility Manager if the QoS level becomes available and can react on the situation, e.g. start transferring data if more bandwidth is available.

- **Interface to QoE Component**

  Furthermore the Mobility Manager provides an interface to the QoE Engine component. The QoE Engine measures the QoE of applications and user and tries to adopt QoS parameter if the QoE is insufficient. Therefore the Mobility Manager provides an interface that allows the QoE for the following functions:

  - **Application Monitoring**
    The QoE Engine can monitor information of QoS parameter of a certain application. These information pertain the connectivity status of an application, e.g. used bandwidth.
  - **QoS Requests**
    The QoE Engine can request a certain QoS level for a specific application. According to the measured QoE of an application, the QoE engine can invoke a change of QoS parameter in order to optimise the QoE.

46.6 Basic Design Principles

Since the beginning of FI-WARE activity, it was stressed that a variety of development paradigms and technologies are available to program and interact with different types of connected devices. The attempt made for CDI Architecture specifications is to keep a detail level high enough to avoid technology dependent definitions. It is however recognised that, when moving to a reference implementation, a specific programming paradigm has to be adopted. A number of emerging solutions for interfacing to device capabilities (and also for other functionality) rely on web based technologies, which are nowadays available on most terminals. This trend seems to favour the development of applications which can run on web browsers or runtime engines.

The architecture for the initial reference implementation of CDI will be designed to be a hardware and platform agnostic runtime, supporting developers in the creation of compelling user experiences across a range of heterogeneous devices via a common development and programming environment. The CDI implementation will therefore utilise a Web Run Time to offer a common programming environment across multiple devices and platforms. This Web Run Time (WRT) environment will be HTML5 and JavaScript based. Hence, the application developer will interact with the CDI implementation through them, as the APIs will be defined for such languages.

The CDI team seeks to build upon existing Web Run Time technologies. Therefore, rather than create a new platform, CDI implementation will extend existing platforms to deliver the required functionality. There are a number of complementary technologies in this area at various stages of development. The CDI aims to utilise current web standards or emerging standards. Therefore CDI will evolve over time to encompass the most promising WRT technologies.

46.6.1 Minimal Native Code

The CDI will support many different operating systems and hardware platforms. Normally supporting this variety of target platforms would require considerable effort in porting code between operating systems and hardware platforms. To minimise this developers contributing to CDI should try to write as much of their code in a portable way. As CDI will ensure that every target platform has a JavaScript environment CDI contributing developers are encouraged to write as much of their logic as possible in JavaScript. This will ensure that a developer's contribution will be portable between the target platforms. Code not written in JavaScript should be kept to a minimum.

46.6.2 Standard Method of Supporting Native Code

A standard interface for connecting the JavaScript engine exists. This provides a standard set of interfaces to which all native code can integrate. This reduces the amount of porting required between platforms. This is required to support the components which must execute in a native environment. This includes a number of networking elements and low-level QoS / QoE metrics which can only be gathered and managed by native code.

46.6.3 Limited API Exposure

The total number of APIs exposed to other GEs and application developers should be kept to minimum, and should enable the scenarios and feature requirements provided by the Use Case projects, as well as by the FI-WARE team.

46.6.4 Interface Definition Tools

There are two main interfaces presented by the CDI GE, an on device interface and an off device interface. Within the reference implementation the on device interface will be a JavaScript interface. The off device interface, which is accessible directly by other Generic Enablers will be a REST based interface. These will be defined using the following definition languages:

46.6.4.1 *On Device Interface Definition Language: WebIDL*

WebIDL is the interface definition language of choice for JavaScript based environments. WebIDL was defined by the W3C standards body and has been used to define many key technologies. As the CDI Generic Enabler will implementing JavaScript based reference implementation all of the functional blocks defined using WebIDL.

A full definition of WebIDL is available from the W3C web site at the following location: http://www.w3.org/TR/WebIDL/

46.6.4.2 *Off Device Interface Definition Language: FMC*

Interfaces which are accessible directly from other GE's and are not based on JavaScript shall be described using FMC (Fundamental Modelling Concepts), more details on FMC can be found here: http://www.fmc-modeling.org/

47 FIWARE ArchitectureDescription I2ND CE

You can find the content of this chapter as well in the wiki of fi-ware.

47.1 Copyright

47.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

47.3 Overview

This specification describes the Cloud Edge GE, which is located beside the cloud, acting as the cloud agent in the end consumer home network. The Cloud Edge, because located at the border between the cloud and the home network can be considered from these two view points. The description from the cloud view point is given in the Cloud Hosting chapter. We here focus on the relation between the cloud edge and the home network.

The Cloud Edge consists in unique equipment called "cloud proxy". This equipment is located in the consumer home network. The cloud proxy is in charge of ensuring the link between the end user (and his device, a laptop for instance) and the cloud. It takes benefit of its position in the home network to anticipate user needs, in particular in terms of data storage. This is why it may store data, quickly available inside the home network. It also takes benefit of its position to organize access to home devices and enhance user experience.

The Cloud Proxy can have 2 main roles:

1. It can act as a small local datacenter helping supporting in-home applications like NAS (file server), video / audio streamer (DLNA/UPnP compatible, for example), home automation etc …

1. Its most advanced role is to act as a proxy to cloud based applications. It then can help the cloud-based application to overcome the traditional limitations of remotely controlling applications such as continuity of service, keeping on with QoS etc…

An illustrative example of the Cloud Proxy helping keeping on with continuity of service is the following: a sophisticated cloud-based application can be created in order to provide heating (or Airco) home automation to a user. This application could provide a very features-rich environment to the user, for example, by providing a link to weather forecast, to energy pricing, to past statistics etc… The chosen temperature value can be sent to a local small application running inside the Cloud Edge that will regulate the devices even if the link between internet and the home falls down. This minimal application may also offer the local user a way to do some simple modifications to the setup temperatures without requiring the data link to be restored thus providing a minimal continuity of service.

Another example is related to the help the Cloud Edge can bring to the user in term of QoE (Quality of Experience). For example, when someone wants to upload big quantity of data to a server (for the illustration, says, numerous pictures to a Picassa-like service), this upload is limited by the small upstream bandwidth of ubiquitous ADSL links (typically in the 500kbps range). With the increasing size of files (pictures), this can take hours. The Cloud Edge can host a small caching application that acts as a local proxy to the remote cloud-based service and that can accept the files at the speed of the LAN and subsequently uploads them (with the user being offline) at the data link speed. This sort of application could save a substantial quantity of time for the user.

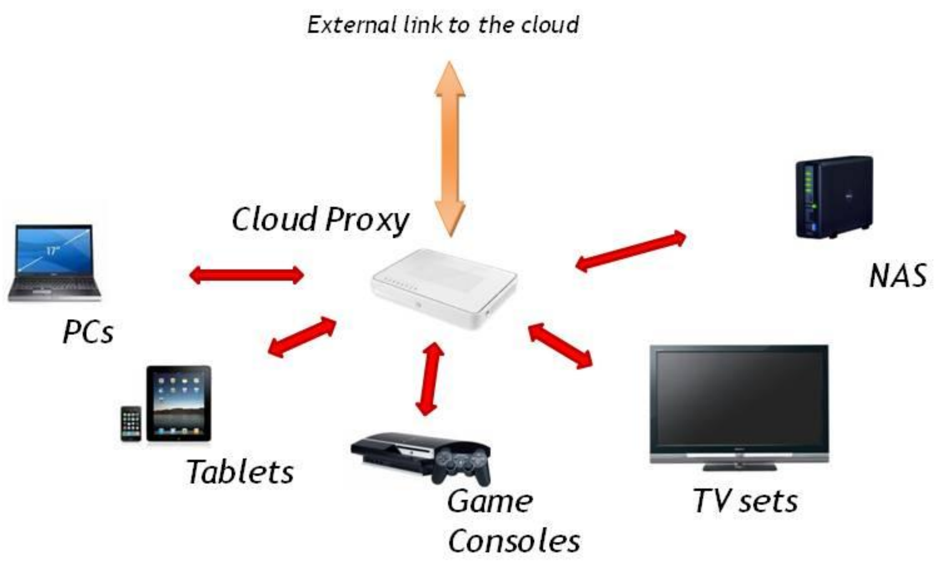
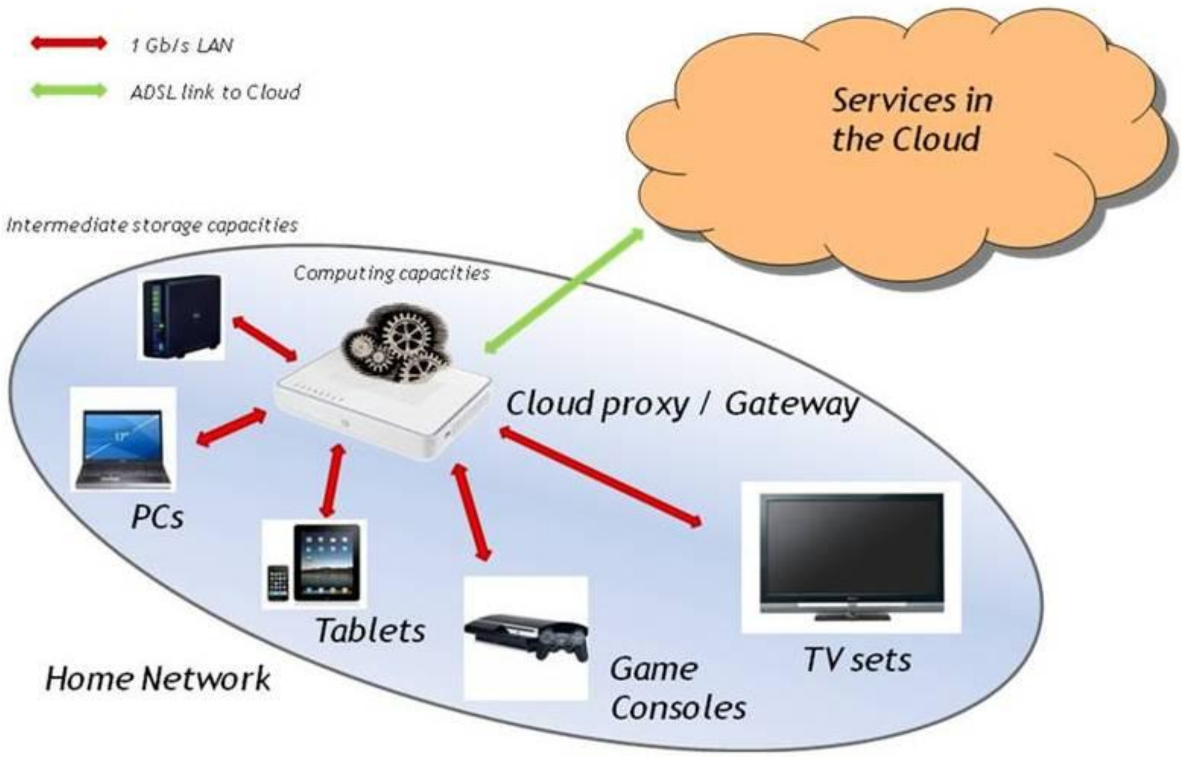


Figure 7.2_1 : The Cloud Proxy in the Home Network

In the I2ND chapter, for what regards the cloud proxy, we are focussing on the relation with the home network. The cloud proxy is acting as an organizing agent of the home network, in particular unifying the access to the data contained in the home network.

The following figure illustrates the concept of cloud proxy.



Cloud Proxy concept

The IaaS Cloud-edge Resource Management GE defined in the Cloud Hosting chapter comprises those functions that enable to allocate virtual computing, storage and communication resources in cloud proxies. However, FI-WARE will not only define and develop the software linked to the IaaS Cloud-edge Resource Management GE but also software linked to middleware technologies and common facility libraries that will be used in VM images to be deployed in cloud proxies. These middleware technologies and common facility libraries are defined and developed with the I2ND chapter and described in the following section in more detail.

47.4 Detailed architecture

47.4.1 Overview

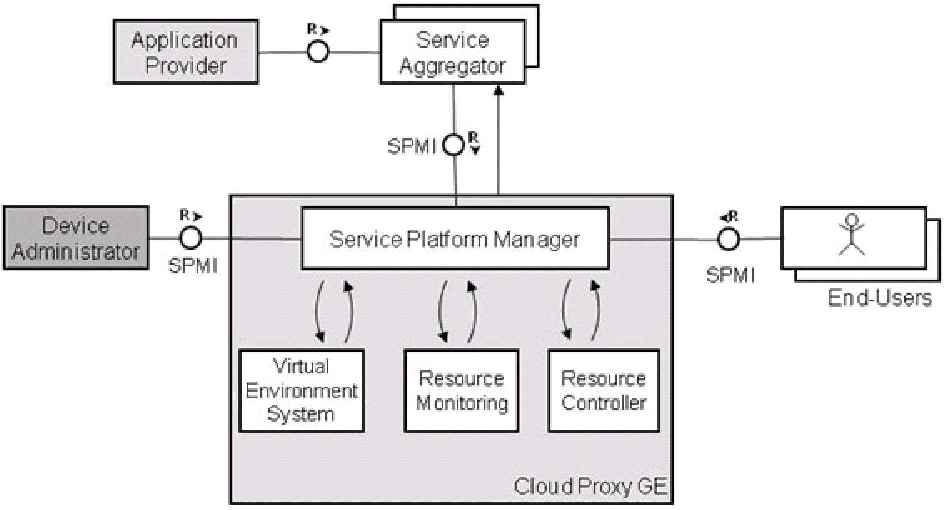
This specification describes the Cloud Edge GE, which is located beside the cloud, acting as the cloud agent in the end consumer's private network.

The Cloud Edge consists of equipment called "Cloud Proxy". Its main function is to offer local Services hosting capabilities as a complement of the standard Service hosting capabilities provided by traditional Cloud infrastructure.

By Services hosting, we mean, the ability for the Cloud Edge to host various downloadable applications. These applications will provide the "services".

Services hosted in such Cloud Proxy can benefit of this unique position inside the private network area of a consumer to provide new enhanced Services that will have a direct access to the LAN-located devices (see the previous examples of applications that can provide a better User Experience or a continuity of service) . It can leverage on the proximity between the consumer and the Service itself to offer Services that require strong connectivity. It can also offer access to private capabilities that are hosted or accessible via the cloud proxy (for example sensors or private home network storage).

The following diagram shows the main components of the Cloud Proxy and the interactions with the different potential actors.



The Cloud Proxy offers a single public interface to manage the local service hosting capabilities: the Service Platform Management Interface (SPMI).

47.4.2 Main Concepts

To use the Service Platform Management interface, you should understand the following concept:

- Virtual Appliance: this entity represents the kind of Service that the cloud proxy is able to host: it is an operating system and application package together that can to run on top of the virtualized system supported by the Cloud Proxy.
- Image: this corresponds to a set of files that compose a virtual appliance and the associate metadata that describe requirement and configuration needed for installing this virtual Appliance
- Instance: in our context, it represents the virtual machine that runs the Service. It is an instantiation of a Virtual Appliance.

47.4.2.1 *Actors and Roles*

We consider four different types of actors that can interact directly or indirectly with the Cloud Proxy.

- Application Provider (AP): entity that creates applications for any users and deploys them on the Cloud.
- Service Aggregator (SA): entity that is in charge to manage a catalogue of applications that are compatible with a set of cloud proxies. Its role is to make sure that the proposed applications are sufficiently safe and secure to be deployed in any private consumer environment. If it gets all the required agreements and authorizations, it can deploy specific applications on a set of Cloud Proxies.
- Device Administrator (DA): root administrator of the Cloud Proxy Device. The DA has complete control over the cloud proxy, this includes:
  - The management of the administration rights of all the other users that can connect to the Cloud Proxy Service Platform Management system
  - The full control of any virtual appliances hosted on the cloud proxy
- End-User (EU): any user (person) that subscribes and consumes a service that runs on the Cloud Proxy. Register EUs have the ability to selects those applications that need to be installed and executed on their local Cloud Proxy.

47.4.2.2 *Platform Components*

The Cloud Proxy GE is composed of four main components: the Service Platform Manager, the Virtual Environment System, the Resource Monitoring and the Resource Controller.

The Service Platform Manager Interface is the REST interface that supports all features offer by the Cloud Proxy. It is the single point of connection for any client (DA, SA or EU) that needs to control and manage any Virtual Appliances. This module is also in charge of managing the users that are allowed to connect and manage the set of images available on the platform.

The Virtual Environment System is the module in charge of running the system-level virtualized commands. In our case, we select LXC (Linux Container) as this virtualization system. This choice is govern by the fact that the cloud proxy is targeted to run in any hardware environment, from PCs to small embedded systems (ex broadband access Gateway). Compared to other virtualization framework (ex KVM, XEN, VMWARE) LXC fits perfectly this requirement because it is light (very low overhead in term of memory and CPU), fast (ability to start or stop any Virtual Appliance in very few seconds), and not require any specific hardware (i.e. no specific processor instructions).

47.4.3 Examples of deployment scenarios

The service hosting capabilities are managed either directly by the end user consumer or by a third party that can deploy a catalogue of application on set of cloud proxies.

47.4.3.1 *Individual on-demand deployment scenario*

Summary: While browsing from his laptop at home, an end-user decides to install an application on his local Cloud Proxy. In this example, the installed application is a small web portal accessible from the home network.

1. A user connect to a web Application and the application propose to install an application into its private cloud proxy

2. The user accepts and provides to the Web App the description of the Cloud proxy environment. In response to that information, the Web application provides the URL of the compatible virtual appliance that could run on the consumer's cloud proxy
3. The user registers this virtual appliance into the cloud proxy. The virtual appliance is downloaded on the system. If this virtual appliance is certified by a third party, the Cloud Proxy checks the validity of the certificate.
4. After reviewing the specific usage's condition of this application, the user creates a virtual machine based on this virtual appliance and starts it.
5. As soon as the local Service is started on the cloud proxy, the web browser redirects the user to the local web provided by the new running application.

47.4.3.2 *Large scale deployment scenario*

Summary: a service aggregator deploys a catalogue of virtual appliances on a set of cloud proxies.

Prior to any transactions, the DA and the SA needs to find a formal authorisation that allows the Service Manager to use the Cloud Proxy owned by the DA.

1. An AP requests from the SA to deploy a specific service on a set of Cloud Proxies.
2. The SA checks in its database what are the Cloud Proxies currently available, and among them, select only the ones that can support the application of the AP.
3. As soon as the set of compatible Cloud Proxies is selected, the SA can start to deploy this application on those cloud proxies.
4. Then, each end-user can browse the catalogue of its own cloud proxy and decide to install it or not.
5. Once an instance of Virtual Appliance is created, any authorized user can start, stop or remove the created Service.

47.4.4 Main Interactions

In this section, the SPMI operations are described. These operations are classified in the following area:

- Platform features operation: these operations are used to provide generic information about the platform itself and the resources that can be shared or offered to virtual applications.
- Images Features operation: These operations are related to the management of images that are available on the cloud proxy.
- Instances Features operation: These operations are used to manage Instances that runs on cloud proxy.
- Users Features operation: These operation are used to manage the user's authentication and authorisation.
- Monitoring Feature operation: These operations are used to provide information about the state and the behaviour of any Instance.

47.4.4.1 *Platform Features*

- Platform version: Provide the current version of the SPM.
- Platform Description: Provide the general information that describes the platform in term of product, hardware and firmware. Used for by any client that needs to provide the right image for a specified Cloud Proxy.

47.4.4.2 *User Features*

- User Create: Create an account for a user. Any user that wants to interact directly with the Services Hosting Platform (install, uninstall a Virtual Appliance) need to be registered. This is performed by the DA, or any local administrator.
- User Attributes Update: Allow the authorized client to change attributes of a user account.
- User Delete: Delete a user account.

47.4.4.3 *Images Features*

- Image Register: Registrar a particular Image into the system so that it is available into the local application's catalogue.
- Image Detail: Provide detailed description of an image and resources it needs to run on the system.
- Image List: Provide the list of all the available images for a given user.
- Image Delete: Delete all the files related to an image and remove the application from the local catalogue.

47.4.4.4 *Instances Features*

- Instance Install: create a Virtual Appliance using a registered image
- Instance Detail: provide detailed description of a specified Virtual Appliance
- Instance Uninstall: delete a Virtual Appliance and free all the resources used by this instance.
- Instance List: Provide all available Virtual Appliances created on the Cloud Proxy.
- Instance Action: Perform a set of actions (start, stop, freeze, unfreeze, reboot) on Virtual Appliances.

47.4.4.5 *Monitoring Features*

- List metrics: List all the metrics and the associated type of statistics that are available on the platform.
- Get metrics statistics: Provide the collected measured values of a set of metrics for a specified instance.

48 FIWARE ArchitectureDescription I2ND NetIC

You can find the content of this chapter as well in the wiki of fi-ware.

48.1 Copyright

48.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

48.3 Overview

Network Information & Control (NetIC) is intended to provide abstract access to heterogeneous open networking devices. It exposes network status information and it enables a certain level of programmability within the network (depending on the type of network and the applicable control interface). This programmability may also enable network virtualization, i.e., the abstraction of the physical network resources as well as their control by a virtual network provider.

Potential users of NetIC interfaces include network service providers or other components of FI-WARE, such as cloud hosting. Network operators, virtual network operators and service providers - within the constraints defined by contracts with the operators - may access, by means of specific FI-WARE services, the open networks to both retrieve information and statistics, e.g., about network utilization, and also (to a certain extent) to set control policies and optimally exploit the network capabilities.

48.3.1 General Note

In order to avoid too verbose text, in this description we typically use the term "NetIC GE" or simply "NetIC" to refer to "an implementation of the NetIC GE Open Specifications" (e.g., "an implementation of the NetIC GE Open Specifications"). Note that the notion of GE is abstract and what actually refers to tangible things are:

- "GE Open Specifications" which contain all information required in order to build components which can work as implementations of GEs.
- a "GE Implementation" which refers to components in a given product that implement a given GE Open Specification and therefore may claim that they are "compliant with the GE Open Specifications".

You may refer to the set of terms and definitions provided here.

48.3.2 Target Usage

The Network Information and Control (NetIC) Generic Enabler will provide to FI-WARE chapters as well as usage area applications and services the means to optimally exploit the network capabilities via a dedicated interface and API. NetIC will both expose related network state information to the user of the interface as well as offer a defined level of control and management of the network.

The beneficiaries of the interface include content providers, cloud hosting providers, context providers/brokers, and (virtual) network providers/operators, all of whom may need to understand and manipulate the network between them and their clients. They might want to set up flows/virtual networks to their clients and they may want to control such flows/virtual networks in order to respect pre-defined Service Level Agreements (SLAs), for example in terms of provided Quality of Service (QoS). There are several use cases for the NetIC Generic Enabler, for example the following:

- A cloud hosting provider has a couple of data centre locations. In order to distribute the allocation of virtual machines (VM) and applications to the various locations, the cloud hosting provider should know about the characteristics of the paths between the locations (e.g., delay, available capacity). To get this information, the cloud hosting provider can request from the network provider (regularly or per scheduled event) the characteristics of the paths between the data centers of the cloud hosting provider. The requested information will be provided via the NetIC interface. In addition, when dealing with migration of virtual machines and applications across data centers, the cloud hosting provider may request a temporary virtual private connection to be setup with a certain quality of service being guaranteed during the time of migration.
- To deliver a service to a client, a service provider may need a certain minimum link quality, e.g., for a high-definition live video streaming service. If the client is willing to pay for this, the service provider will request via NetIC from the network provider the setup of a virtual connection with certain quality characteristics between the server and the client. NetIC will do so if capacity is available.
- A network service provider wants to implement new business models based on the "pay-as-you-go" paradigm, setting up a specialized service for a group of clients. The specialized service is built orchestrating the network resources dynamically. A virtual network (optical or packet based) is required that connects servers, network elements and the involved clients, potentially running customized protocols. The service provider can request via NetIC from the network provider a virtual network between the involved endpoints, possibly also with some specified constraints (quality characteristics, isolation against other virtual networks, energy efficiency metrics) defined.
- A service provider wants to set up a specialized service for a group of clients. For this they need a virtual network connecting some servers and the involved clients, potentially running customized protocols. The service provider can request via NetIC from the network provider a virtual network to be setup between the involved endpoints, possibly also with some specified quality characteristics and isolation against other virtual networks.
- A cellular service provider wants to run its business on top of a virtual network which is able to “breathe” (to be re-configured as demand changes) since loads during idle and busy hours differ significantly. Benefits include reduced expenses (CAPEX is turned into pay-per-use OPEX), reduction of energy consumption and management flexibility. Today mobile traffic is typically mapped into static MPLS tunnels, and the infrastructure providing these tunnels is owned by the cellular service provider, too.

A fundamental challenge for the implementation of NetIC is that the network functionality is typically distributed over different elements potentially implemented internally in different ways (in multi-vendor environments). Also, the interfaces have to take into account the constraints of different providers (in multi-network service scenarios) as well as legal and regulatory requirements. These problems have been solved in the past by different standardized control plane solutions. This readily available functionality could be re-used by NetIC in order to provide a smooth evolution path rather than introduce a disruptive revolution. NetIC instances may be deployed by the different involved parties (e.g. virtual network providers/creators, and virtual network operators/users running a business on top). As a consequence, several instances of NetIC with different scopes may have to work together to deal with a request from e.g. a service provider or an application. Each might cover a different part of the network, for instance in the horizontal direction (i.e., type of access) or in the vertical direction (i.e., ownership, virtual network).

It should be noted that the capabilities a specific NetIC implementation can offer depend on the capabilities of the underlying network.

48.4 Basic Concepts

48.4.1 Northbound Interface

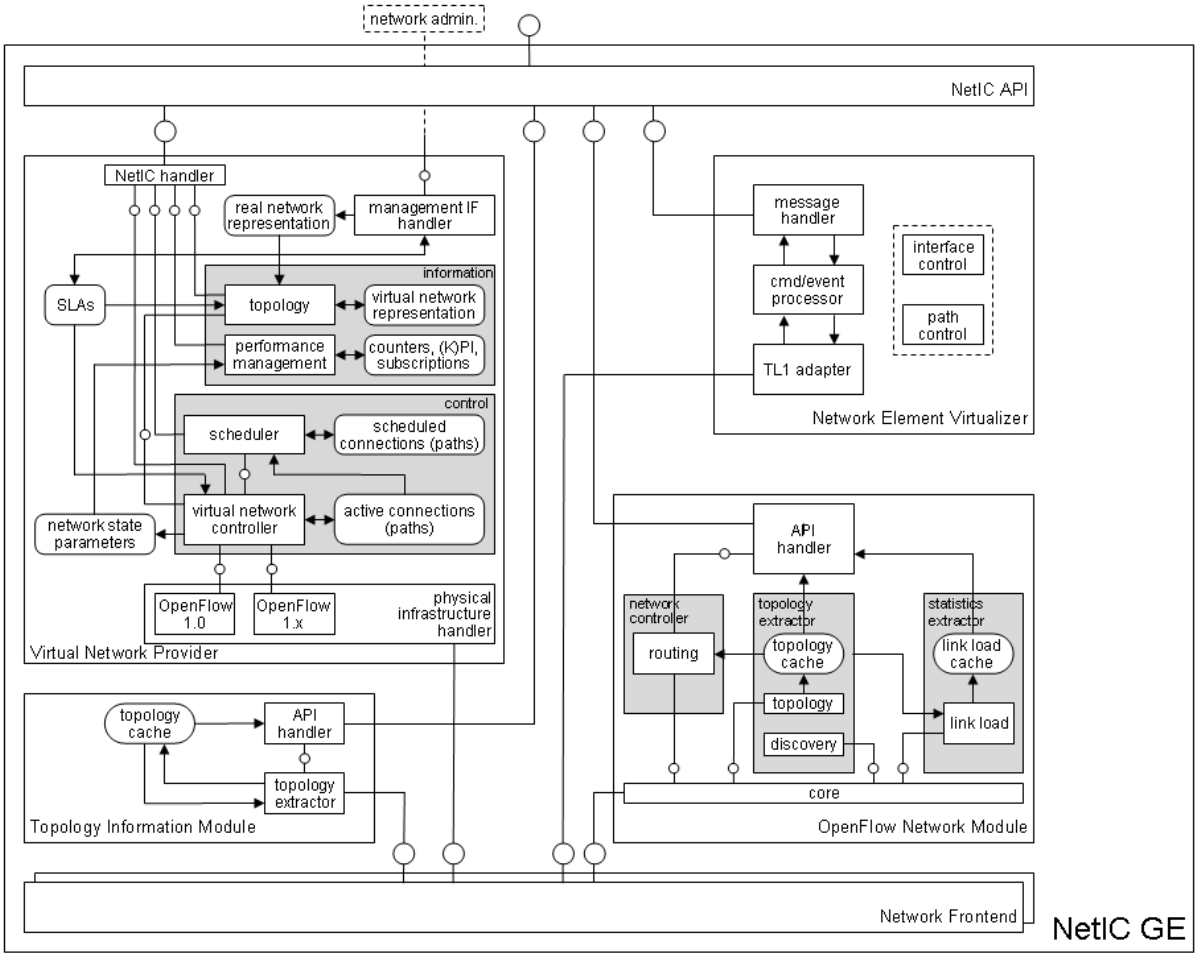
The northbound interface is intended to expose network status information and to enable a certain level of programmability within the network (depending on the type of network and the applicable control interface). This programmability may also enable network virtualization, i. e., the abstraction of the physical network resources as well as their control by a virtual network provider. Depending on the purpose (e.g. only information provision or control of network) the interface will have different characteristics.

It should be noted that the exposition of specific capabilities via the northbound interface depends on the capabilities and the technology of the underlying network.

Details of the northbound interface are currently being defined.

48.5 NetIC Architecture

The block diagram below shows the main functional modules of NetIC. It should be noted that the presence of a given module in a specific NetIC instantiation depends on the network being controlled by this instantiation.



NetIC GE Functional Block Diagram

The following sections give a brief overview on the functional modules and their interfaces.

48.5.1.1 *NetIC API*

The NetIC API is the conceptual north-bound interface of the NetIC GE. It exposes the internal module interfaces to the outside world such that applications, or other GEs, can access the modules that are present in the specific NetIC instantiation.

48.5.1.2 *Network Element Virtualizer*

The Network Element Virtualizer (NEV) represents the glue technology between an ALU physical device and the NetIC. It is part of the NetIC, acting as network element abstractor, transforming instances of physical resources into virtual resources. In other words: NEV permits to integrate the ALU network element 1850TSS-160 in the NetIC.

Message Handler

The Message Handler implements the following features:

- manages the communication over the NetIC interface
- implements the basic primitives of the NetIC interface (synchronization, provisioning, monitoring, restore)

- buffers incoming messages received via the NetIC interface before sending them to Command Processor (if needed)
- communicates with Command Processor and Event Processor modules, in particular:
  - forwarding messages received on the NetIC interface to the Command Processor depending on the message type
  - receiving messages from the Event Processor to be sent via the NetIC interface

Command Processor

The Command Processor implements the following features:

- manages the requested operation execution on the network element (physical infrastructure)
- translates abstract messages (commands) received from Message Handler to device specific commands (TL1)
- manages the requested operations execution process. In some cases there may be the need to break a received abstract command into a sequence of device-specific operations.
- communicates with TL1 Adapter and Event Processor, in particular:
  - sending equipment specific messages in the form of strings to TL1 Adapter
  - sending event notifications that need to be sent upwards to the Event Processor. E.g. information that the requested operation has been executed
  - receiving notifications from the Event Processor about actions executed on the device

Event Processor

The Event Processor implements the following features:

- manages the notification translation process from device-specific context to abstract device model-specific context.
- communicates with TL1 Adapter, Command Processor and Message Handler. In particular:
  - receives equipment specific messages / notifications from the TL1 Adapter
  - sends operation execution status to the Command Processor
  - sends event notifications (received from the TL1 Adapter) to be sent upwards to the Message Handler

TL1 Adapter

The TL1 Adapter implements the following features:

- provides an API for communication with the physical device
- manages login on the physical device (has access to login credentials)
- buffers commands to be sent to the network element (if needed)
- communicates with the Command Processor, the Event Processor and the network element

  - receives commands (in the form of strings) to be sent to the network element from the Command Processor
  - sends device specific commands to the network element
  - sends notifications received from the network element to the Event Processor (in the form of a string). It doesn't perform any operations on the received notification

48.5.1.3 *Topology Information Module*

The Topology Information Module will provide abstract information about

- the nodes in a network
- the address ranges associated with the nodes
- communication costs between the nodes

API Handler

The API Handler deals with topology information requests from applications or other GEs. If the Topology Information Module API is invoked, the API Handler will explore the Topology Cache and subsequently generates an appropriate answer. If the Topology Cache does not contain the required information, the API Handler may trigger the Topology Extractor to retrieve the required information.

Topology Cache

The Topology Cache is an internal database of the Topology Information Module. The database is generated and updated by the Topology Extractor. The API Handler has only read access to the database.

Topology Extractor

The Topology Extractor is responsible for obtaining topology information from the underlying network. It issues network specific information requests, and the received information is used to generate and update the Topology Cache.

48.5.1.4 *Virtual Network Provider (VNP)*

NetIC handler

This functionality acts as a proxy for requests/responses exchanged between VNO(s) (Virtual Network Operators) and the VNP (Virtual Network Provider). The NetIC handler functionality processes requests (and related responses) arriving via the open NetIC interface and forwards them to the appropriate functionality:

- on-demand connectivity requests are forwarded to Virtual Network Controller functionality,
- requests related to scheduled connectivity requests are forwarded to the scheduler functionality, and
- queries about network status and monitoring requests are forwarded to the performance management functionality.

This functionality also processes notifications (and related responses) sent to VNO(s) via the open NetIC interface. These include:

- notifications from topology functionality about network errors (those affecting active connections and where VNP could not recover the error),
- notifications regarding performance management.

Management interface handler

This functionality provides an interface (via a GUI or any proprietary management interface) to setup and manage the environment of the VNP.

For the controlled network it allows the physical topology to be defined, including:

- the list of nodes the VNP controls (these are OpenFlow switches),
- the list of external nodes to which the VNP provides connectivity (peers),
- the list of links the VNP controls (these include both the internal and external links of the controlled network),
- the parameters of the controlled nodes (capabilities, e.g. supported technology, physical links), and
- the parameters of the external and internal links (e.g. bandwidth, granularity).

For the served virtual network operator(s) it allows the definition of:

- the identity of the served Virtual Network Operator (ID, access point), and
- the SLAs (e.g. provided Maximum Bit Rate (MBR), Guaranteed Bit Rate (GBR)).

SLAs

This is a storage place for Service Level Agreements (SLA) with served Virtual Network Operators (VNO).

Topology

This functionality is responsible for:

- creating the virtual network representation(s). The virtual network representation is created based on the physical network representation and the SLA on a per VNO basis (the virtual network representation can be different for each VNO),
- updating virtual network representation(s) (either by regularly checking for changes in physical topology and SLAs or after receiving notification from management interface handler),
- informing Virtual Network Controller on virtual network representation changes;
- answering Virtual Network Controller queries on network topology and network representation.

Real (physical) network representation

This is a passive system that stores all parameters of the controlled network. The stored data is (over)written by management interface handler functionality and read by the topology functionality.

Virtual network representation

This is a passive system that stores virtual network representations shown to each VNO. It is managed by the topology functionality.

Performance management

This functionality manages network information related tasks. It is responsible for:

- storing counters, triggers, (Key) Performance Indicators, event subscriptions received from VNO(s),
- notifying VNO(s) about requested events, and
- monitoring network state.

Counters

This is a storage place for counters, performance indicators and event subscriptions requested by each VNO.

Virtual Network Controller (VNC)

The virtual network controller acts as a centralized controller for the underlying physical network. The virtual network controller

- receives on-demand connectivity setup / modification /removal requests from VNO(s) through the NetIC handler functionality and from the scheduler functionality,
- changes network control rules according to the connectivity requests (note that "connectivity request" can mean the setup, modification, and removal of a connection),
- sends the updated control rules to the underlying physical network through the physical infrastructure handler functionality,
- updates the active connections and network state parameters if necessary after successful control rule change,
- updates network state parameters according to the notifications received from physical network through the physical infrastructure handler functionality, and
- (if VNP Recovery use case is supported) upon error notification from the physical network, it attempts to change control rules to keep affected connections alive by changing connectivity paths.

Network state parameters

This is a storage place for values / parameters provided by the physical network to the VNP (Virtual Network Provider). It is updated by the VNC (Virtual Network Controller) functionality and monitored by the performance management functionality.

Scheduler

The scheduler functionality:

- receives scheduled connectivity requests from VNO and maintains the "scheduled connections" storage according to these requests, and
- whenever it is necessary to setup / change / remove a connection according to the stored "scheduled connections", it instructs VNC accordingly.

Scheduled connections

This is a storage place for scheduled connections (endpoints, connection parameters, timing parameters).

Active connections

This is a storage place for active connections (endpoints, connection parameters, connection ID).

Physical infrastructure handler

This functionality acts as a proxy for requests/responses exchanged between VNC and the controlled physical network.

The possible instances for an open network are OpenFlow 1.0, 1.1, 1.2.

48.5.1.5 *OpenFlow Network Module*

The OpenFlow Network Module will be an OpenFlow controller able to fulfill requests coming from the users of the NetIC API. In particular it will be able to accomplish the following types of requests (using the mechanisms provided by the network frontend protocol):

- Synchronize - used to retrieve available network resources
- Provisioning - enables configuration of a given physical resource
- Release - allows release of previously configured resources
- Monitor - provides information about current status and utilization of a given network resource

API handler

The API handler has the role to map NetIC API commands into the OpenFlow Network Module. To this aim, it exposes a RESTful web service to the NetIC API side. NetIC commands received on the HTTP interface, are sent to specific control plane blocks that effectively implement the required NetIC functionalities (synchronize, monitor, provisioning, release).

Core

This module directly provides to OpenFlow Network Module components an API to exchange OpenFlow messages with the network. It also implements an event dispatching functionality to notify components of events raised by the network, or raised by other components. OpenFlow Network Module components find events to be a powerful way of communication, as it is possible to include a certain amount of information in each event.

Network Controller

This module fulfills requests to provision and release network resources. In particular, it is able to install and remove network paths, on the basis that NetIC provides/releases commands. To achieve routing functionalities, this block needs to be aware of the network topology.

Topology Extractor

This module exposes synchronization functionality. The Discovery sub-module is in charge of discovering network links. Whenever a new link is detected it generates an event that is further dispatched to Topology by the underlying Core functions. On the basis of network nodes and network links, advertised respectively by the network, and by the Discovery component, the topology is inferred and stored in a Topology Cache.

Statistics Extractor

This module exposes monitoring functionalities; in particular it is able to estimate port utilization of OpenFlow switches in the network. The sub-component Link Load exploits Core functionalities to periodically poll switches to access their port usage statistics. Such statistics are stored in a Link Load cash, and accessed by the API handler when monitoring requests are received from the NetIC API layer. Information about all available switches in the network is retrieved from the Topology Cache.

48.5.1.6 *Network Frontend*

The Network Frontend is the network-specific and technology-specific representation of the network interfaces accessed by the functional modules.

48.6 Main Operations

Users will have two different ways to access the functions of NetIC, depending on the flavor of NetIC and the accessed functions:

- A message based interface (request/response, REST based) is provided by an information and control entity instance responsible for a particular network. This interface flavor will be used for detailed information gathering about network internals (like node/port status, link load, etc.) and control of network elements (e.g. activate or deactivate nodes/ports, setting up routing information). This interface is currently envisaged for NetIC implementations implementing the Virtual Network Provider, the network Element Virtualizer or the OpenFlow Network Module).
- A library will offer access to coarse-grained network information, i.e. a subset of NetIC. Offering a library facilitates the integration of NetIC functionality in applications (e. g., FI-WARE Generic Enablers or applications, Use Case Project software). The library will focus on providing easy-to-use access to network information at the level of detail useful to applications (e. g., for application-level load balancing). This simplification of use also implies that not all functions of a NetIC instance can be accessed. The initial implementation of the library will be available for Linux/Unix-based applications. This interface is currently envisaged for NetIC implementations implementing the Topology Information Module.

Depending on the authorization level, a customer is allowed to perform different actions. A proper authorization level might be acquired from the network operator by negotiating the NetIC GE usage terms by using the S3C GE.

48.7 Basic Design Principles

48.7.1 Rationale

The NetIC Generic Enabler will provide access to network status information to its users. Interfaces available today are already able to provide specific information, but the interface highly depends on the specific network technology. The aim of NetIC is to define a set of general functions to access network status information in a technology independent way, overcoming the heterogeneity of today's solutions.

48.7.2 Implementation agnostic

There are several standard technology and implementation dependent interfaces to control and manage specific networks. To overcome this heterogeneity, the objective of the NetIC Generic Enabler is to provide a generic interface to control and manage open networks. The interface shall be technology and implementation independent.

49 FIWARE ArchitectureDescription I2ND S3C

You can find the content of this chapter as well in the wiki of fi-ware.

49.1 Copyright

49.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

49.3 Introduction

All four GEs of I2ND have a different focus but will be interconnected through interfaces and S3C will be the central control and management entity/GE. However, to the outside world, each GE has an interface towards the network infrastructure as well partly to the FIWARE services, applications and Cloud services but the main interactions will be done through S3C. The interface format towards the network infrastructure will be handled by the respective GE. The Interface between the Cloud services and FIWARE services and applications should be a common interface and is under discussion. The interfaces between the GEs will be network oriented and up to now the template of IEEE 802.x standards for multiple layer management as described in IEEE 802.11 (Section 10 on Layer Management) [1] will be used. The specific layer management was designed to sustain the communication between the multiple layers for strict time constraint environments, involved through a simple set of primitives. The same format was further used by IEEE for multiple standards such as IEEE 802.11a, 802.11g, 802.16, 802.21. This format/template might be a good template for the exchanging the necessary information between the GEs. The specific implementation of the communication template proposed depends on the specific protocols selected for each of the reference points. For the time being, we are considering this template format in any cases where we have IP interfaces. An exception is the interface between S3C and NetIC and other legacy networks, here the Gx-Interface of the Evolved Packet Core (EPC) will be used and extended for exchanging the necessary information.

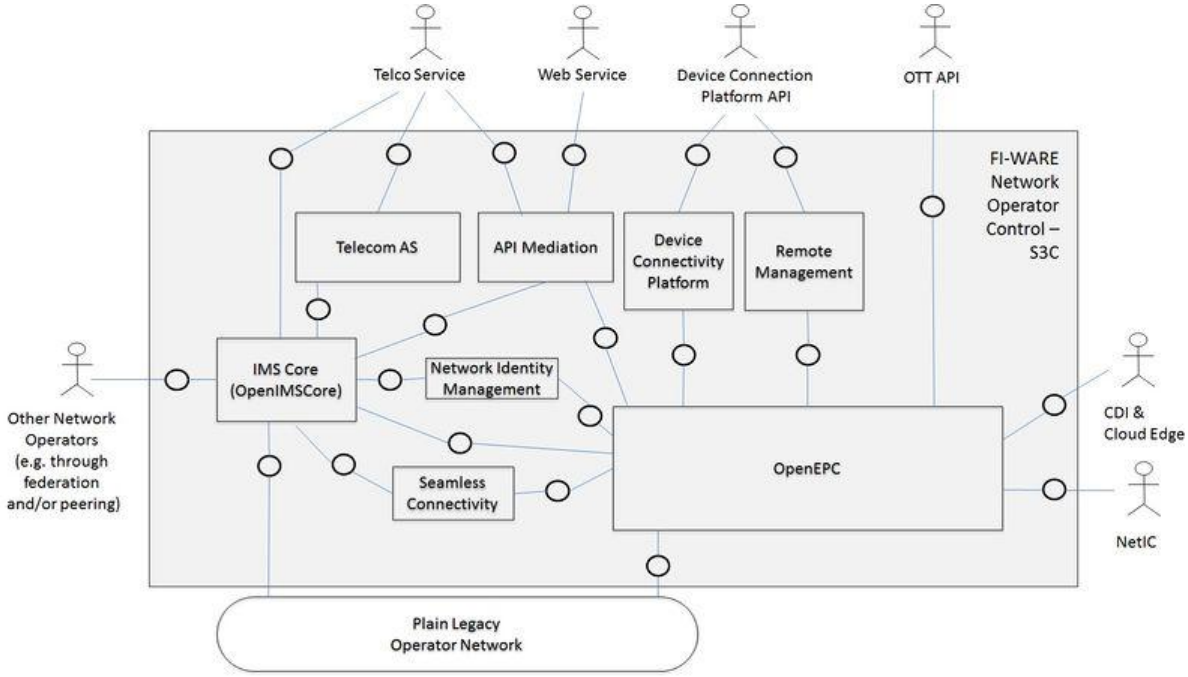


Figure 7.4.0: Graph of the Generic Enablers, APIs, and Interfaces to the outside world (FI-ware WPs, use case projects, other network operators)

49.3.1 Target usage

The Service, Capability, Connectivity, and Control (S3C) Generic Enabler is the manifestation of an adaption layer between the targeted network control layer for fixed-mobile-convergence: Evolved Packet Core (EPC) and all possible applications and services.

Driven by the rollout of new wireless access technologies providing only packet transport capabilities like the 3GPP Long Term Evolution (LTE), the future massive wireless broadband environment is bound to transform into a data dominant environment. In order to respond to the requirements of the new environment, 3rd Generation Partnership Project (3GPP) defined the Evolved Packet Core (EPC) as a new IP connectivity control platform enabling wireless access network diversity (including LTE, UMTS, WiMAX, WiFi, etc.) and offering seamless connectivity for the various service platforms. It maintains the same central concepts as previous 3GPP architectures, like IMS: policy based QoS and charging, subscription based access control, handover support and security, offering a scalable alternative to the current deployed architectures.

By using all-IP based communication protocols and functionality, the EPC is designed to support large number of subscribed devices, their signaling and data exchanges through its flat network design supporting mobility management inside the same or in different access technologies, subscription based resource management and accounting along with security support of the communication.

EPC provides a transparent convergent network layer for the IP Services. From the perspective of a service provider without a modification of the way the services communicate, it enables a high degree of satisfaction, by transparently supporting features like access control, QoS assurance, seamless mobility between the different access networks, prioritization and security.

Also due to the resource reservation mechanisms, the services have a guaranteed quality of the communication, which is an addition to the typical IP communication and a high added value for broadband communication on mobile devices with reduced processing power.

EPC provides also a set of control mechanisms between the service platform and the network core. Through these mechanisms, the EPC aware applications can transmit indications on the resources that have to be reserved for the specific users at specific moments of time. They can also receive upon request information on events happening at the link and network layers e.g. the connected device lost connectivity or a handover to another access network occurred. By these mechanisms, the applications can be adapted to the momentary context of the mobile device and to offer services customized not only based on the service level user profile, but also to the mobile device in use, the mobility pattern and to the surrounding network context.

Although not yet standardized, EPC is able to export a set of enablers to the applications, which offer even more flexibility in the service delivered to the mobile user. For example, services may use the location of the connected device or even ambient information on the vicinity of the connected device and the subscriber identity of the mobile device, in order to further more adapt to the environment conditions and to ensure a more secure communication.

With the deployment of new devices such as sensors and actuators along with the further increase in usage of the IP capable mobile devices such as laptops, tablets and smartphones, it is foreseen a high increase in signaling and data traffic expanding the overall required resources from the network.

Also new service paradigms are expected to be further developed such as mobile cloud computing or machine-to-machine communication which will even more broaden the types of communication both as communication patterns, mobility and resources required.

However, the EPC was designed with point-to-point human communication as the main service paradigm, which will require a new adjustment in customizing the IP connectivity according to the momentary needs of a high number of devices using the broad future internet services.

49.4 Overview

S3C aims at providing a unified and scalable control of the connectivity of the devices over heterogeneous access networks and core network technologies, transparent to the devices and to the services deployed on top. For this, S3C contains automated functionality related to the management of the various features related to connectivity such as the usage of the network context data and events for triggering resource management, mobility, connectivity management, and security operations. Additionally, it provides to the application and services platforms an extended service API that enables the configuration and control of the connectivity service with different complexity, granularity, and application specific requirement levels.

The S3C GE will have different interfaces towards the other work packages and the world, other operators, other I2ND-GEs and the S3C-assets. A service API will be defined and used towards the other work packages, use case specific functions and other third party state of the art service platforms and applications. It contains four different interfaces and protocol technologies to interact towards the service providers. One will be used for the telecommunication services (SIP) and another for the Over-The-Top Web-services (HTTP). In addition a device management and control interface is needed, named here Device Connection Platform API. Finally, all other applications and services will be interfaces through complete and partial usage of an Over-The-Top API (OTT API) addressing plain IP connectivity with the DIAMETER protocol for control.

The connection towards other operator networks will be established and controlled, such as described D4.3 [10] of the EU Project ETICS (Economics and Technologies for Inter-Carrier Services, [11]). They have defined an Abstraction/Intermediate Layer especially for the interconnection of operators: ETICS Deliverable D4.2 [2]. The other operator may deploy legacy or S3C based network operator control infrastructure.

Between the GEs Connected Device Interface (CDI) and Cloud Edge (CE) protocols like Plain IP and OMA DM (Open Mobile Alliance Device Management, [3]) will be used. Towards NetIC the control-interface from the EPC: Gx will be in place.

Within the S3C, it is planned to use plain IP, IMS SIP and the DIAMETER protocol (through interfaces such as the 3GPP Rx for resource reservations, Sh for subscriber profile management and Rf for charging correlation) to interlink with the assets.

For explanation and easier understanding of the following four pictures: The red blocks are Assets from the respective partners. The orange blocks are the features from the S3C GE according to the GE high level description [4].

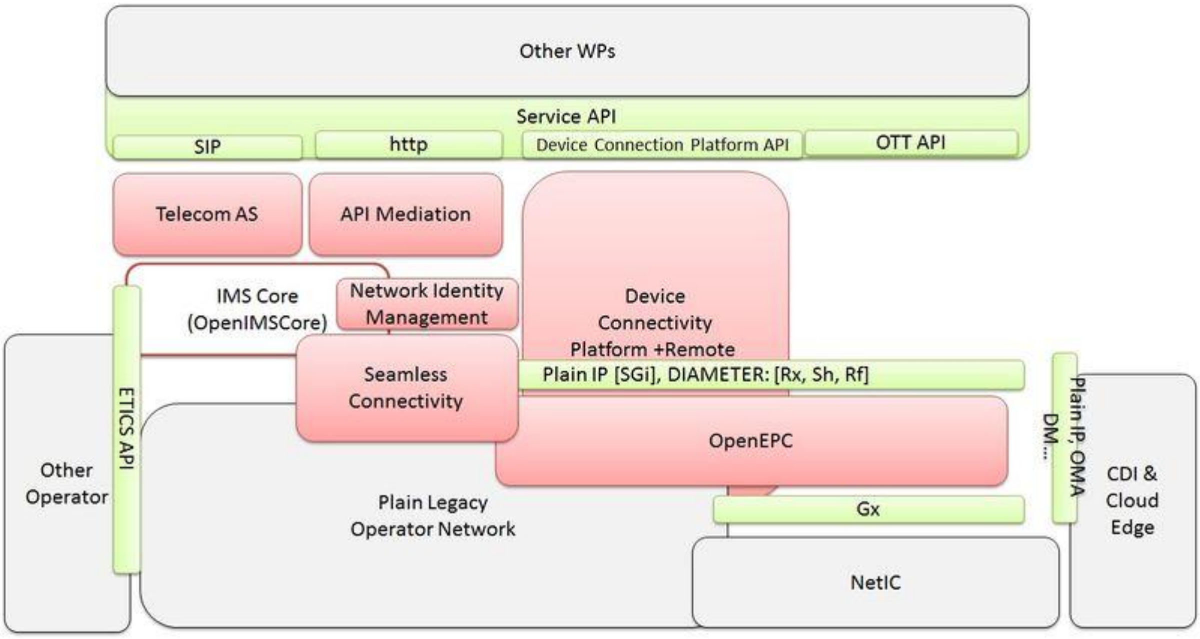


Figure 7.4.1: Overview of the interfaces towards outside of the GE and internally

49.5 Basic Concepts

49.5.1 Differentiation between feature groups

The S3C Abstraction layer/Generic Enabler contains independent features, which will interact to each other:

- Network Event Management
- Resource Management
- Network Identity Management

- OS4AC & LI – Operator Security, Authentication and Authorization, Accounting, Auditing and Charging & Legal Interception
- Connectivity Management
- Broadcast & Multicast Management
- Network Data and Caching Aggregation
- Small Data Push & Pull
- Network Context Data Management
- Inter-Carrier Management
- Positioning Management

The feature space is not complete, it might be necessary to add new features according to the needed functionality and the request from the usage areas. The features are grouped into three different classes, depending on the requirements that lead to their development.

- internal features – features that enable a scalable and efficient operation of the core network, transparent to the applications and services
- external features – features exposed to the application and services enabling the communication of requirements and events as well as of the actual data transferred between the devices and their correspondent nodes
- and special requirements – features addressing special requirements from the different application areas related to the efficient and scalable data delivery.

Parts of the features are already implemented and through a minimal adaptation can be integrated into the FI-WARE platform. Other features require specific extensions. These extensions are under discussion. A deeper analysis is in work.

49.5.2 Use Case scenario

The challenge in S3C is remarkable given the requirement for user connectivity coverage in a wide range of scenarios. A significant role S3C plays is in the field of eHealth. Where a typical use case scenario is an accident of several cars on the highway, in which a number of people have been injured. In such a case, people will call the ambulance and contact doctors through voice, video conferencing, and augmented reality aiming for highest quality local support. Being on a highway the emergency call is most likely to be done by a person in the area of the accident, in this case a caller’s location is not a fixed static address, but rather a mobile device somewhere on a mobile network. It is also highly expected that the person calling the ambulance is not able to accurately describe the place of the accident on the highway. Determining the exact position of the accident is a significant aspect in such a scenario because lives, or threatening situations to life, depend on it. Given the accurate position of the accident is determined, injured people will be transported to the hospital via the ambulance or helicopter. Augmented reality, video conferencing, and voice support helps on deciding for the process to select the heavy injured ones and to receive medical records in real time.

- The request will processed via SIP connections and will be handled by “Telecom AS” towards the IMS core (We will use for the prototypes the OpenIMS [9] platform.
- The Positioning Enabler mechanism, used for location-based services, is used for determining the location of the mobile device from which the emergency call originated, and hence the exact place of the accident. Moreover, it is used to route

the call to the ambulance in closest proximity or to the appropriate public safety answering points (PSAP), the caller's phone number and fixed location is automatically displayed for quick-response dispatching.

- Via the functions “Seamless Connectivity”, “Network Identity Management” and “OpenEPC” the request can be classified as important and will be treated with highest service quality through the plain legacy operator network. Or through a request to NetIC, a special QoS path can be established for end-to-end interconnection in one single domain.
- Additional communication with less importance will be safely delivered directly to other actors which may be involved in the procedure such as the health insurance companies, the car manufacturers and other legal entities.
- In the case of an end-to-end connection through several domains, the ETICS interface [11] can be used to set-up an end-to-end-QoS path.

49.6 Main interactions

For S3C, we have five different interfaces towards the world (in principle and especially to FI-WARE GEs, usage area projects, and other operators) and three towards the three GEs of I2ND:

- Telecommunication support through SIP - The telecommunication services will be handled through SIP, which is the common standard in an All-IP world [5].
- Web-service support through HTTP [6], which will enable OTT web and Cloud services in telecom networks.
- M2M support through a Device Connection Platform API - This interface is currently under discussion since it is a vendor specific interface.
- OTT support through a special DIAMETER API [7] for control and plain IP for data addressing OTT services of the same operator, in addition it is under examination if the IEEE 802.x standard template [1] for information exchange can be used.
- End-to-end connection support through different operator domains - This interface is defined by the EU-Project ETICS [2]. It is planned to have a close relationship to this project.

The three other GEs will communicate through S3C, therefore the S3C is the central GE. The interfaces are defined in the following.

49.7 Basic Design Principles

49.7.1 Differentiation between internal features, external features, and special requirements

The feature group for external functions and inter-connections are defined by the following assets and will cover the GE as shown in Figure 7.4.2.

- Inter-carrier Management – management of inter-operation issues. It is/will be defined by the ETICS project. It will offer the possibility to interconnect with other operators in an end-to-end fashion.

- Network Event Management – information about changes in the status of the network for the specific sessions of the specific device. It will offer the possibility to monitor and to react on network events while a specific service is active. As part of the considered services in this class, the basic reachability through network attachment as well as the sessions of the specific services.
- Network Context Data Management – location, device status, user status/activity, and user role. It will help to manage network specific context information and will help to optimize the efficient usage of network resources and will support the FI-WARE and usage area services and applications. The network context data management is not related directly to any active service on the mobile device. Selected information from these functions will be exposed to the external entities communicating through an S3C operator network as part of the Device Connection and Operator OTT APIs. The exact features depend on the use cases requirements while maintaining the subscriber privacy and information protection.

The positioning enabler mechanism provides functionalities needed to determine the location of a device in a network-centric approach. Positioning Enabler is supported partially through OMA Secure User Plane Location (SUPL) 2.0 [8]. Location information will be made available for device users as well as 3rd parties communicating with the S3C operator network.

- API mediation layer – Makes it possible for HTTP services to query and to use the network communication services for the specific service delivery.

It is important to be able to expose networks APIs to the HTTP realm in a managed way without the hassle to have to implement a mediation layer in each applications or network enabler. Third party providers could use Telco APIs to improve their own HTTP services. Network APIs should be easily discovered and used by end users. It should also be able to charge easily end users and third parties.

The API mediation component provides the functionalities needed to manage the exposed interfaces to several actors interacting with the network infrastructure.

The API mediation component deals with publishing, discovering and exposing APIs to stakeholders as well as managing some non-functional aspects such as provisioning and monitoring. It doesn't create or host APIs nor manage/provision customers, but makes it possible for API providers to register new APIs sets to the Exposition, QoS and resource management.

This API mediation layer belongs to S3C north interface. Its role is to publish/expose API and to authorize users, manage contract terms and create CDR if needed. API mediation layer's south interface interconnects with heterogeneous and legacy networks. North interface exposes those networks interfaces to HTTP services.

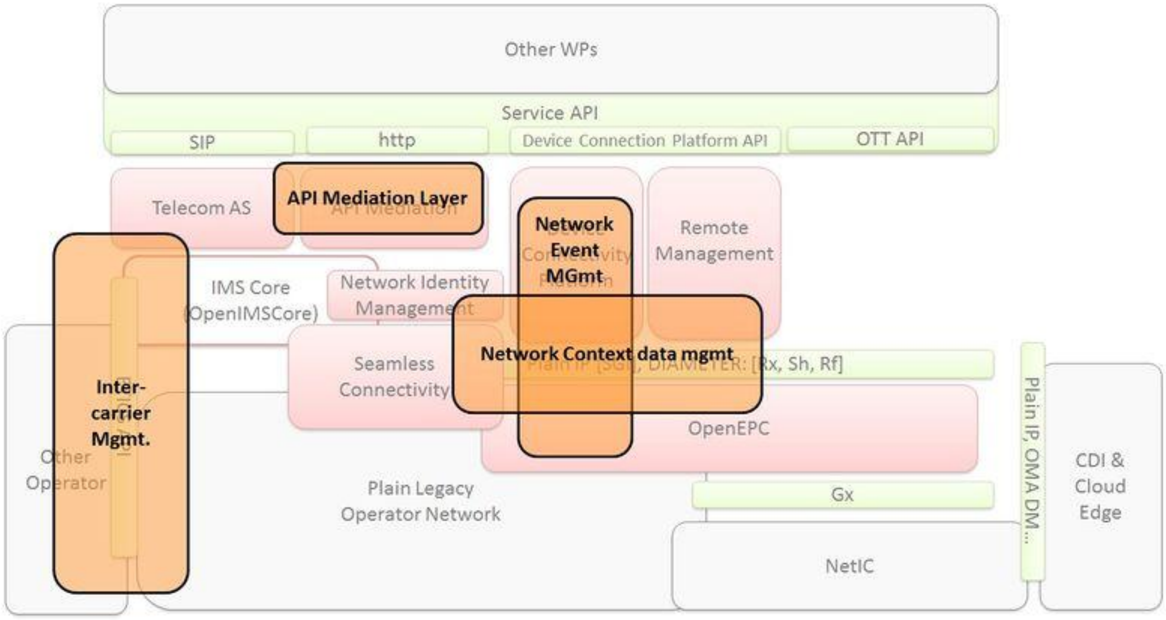


Figure 7.4.2: Feature group for external connectivity

49.7.2 Feature group for internal control functions

The feature group for internal functions and inter-actions are defined by the following assets and will cover the GE as shown in Figure 7.4.3.

- Network Identity Management – device, services, applications, and user identity at network level. It will support the management of different identities within the network. An interface to the ID management of the security work package will be supported and this is currently under discussion.
- OS4AC & LI – Operator Security, authentication and authorization, accounting, Auditing and Charging & Legal Interception. The feature will support the network internal mechanisms for security, AAA, auditing and charging. Through this means the appropriate usage of the network is ensured. The specific features rely on the specific network identity of the device. Through this identity and through the specific credentials, the device usage of the network is authorized and a binding is created between the current attachment point in the network and the services used. Lawful interception will give defined authority the restricted and lawful controlled access to network and used IDs
- Connectivity Management – monitoring and controlling the connectivity state of the devices. It will help to support the efficient monitoring and the control of network devices. This includes M2M equipment addressing, the scalable control of multiple devices pertaining to the same entity, such as multiple sensor and actuator devices in the same administrative domain or enterprise controlled devices.
- Resource Management – QoS control, access network and core path selection, and computation offload support. This feature will support the overall network resource management including the interconnection and information transfer towards the NetIC. The resource management functionality enables the external entities to require a specific level of resources between different nodes in the network. This may represent mobile devices, data centers or any other operator entity able to

communicate, as well as the entities which enable the communication with the other network operators. Based on these requirements, the resource management makes policy decisions and generates specific QoS and charging rules. These rules are then installed on the data path, guaranteeing that the resources are reserved according to the service applications. The installation may be done on plain legacy and operator networks or on the open networks controlled through the NetIC Generic Enabler. In case the resources cannot be guaranteed, due to a congestion state at the network level, a notification is transmitted as response to the resource requirements, enabling the connectivity requirements to be adapted to the current status of the network.

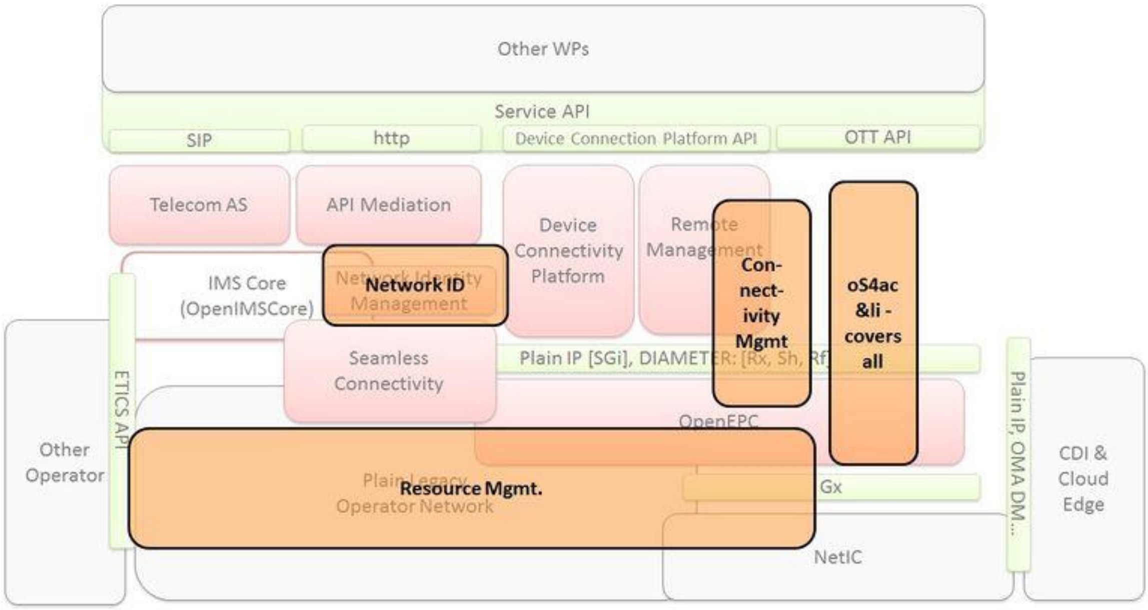


Figure 7.4.3: Feature group for internal actions

49.7.3 Modules for special network features

The feature group for special network functions is defined by the following assets and will cover the GE as shown in Figure 7.4.4.

- Broadcast & Multicast – to optimize the network resources for information, which do not necessarily be handled by unicast connections. Some services are delivered at the same time to different users or devices. In this case, a multicast or broadcast delivery will optimize the usage of the network resources.
- Small messages (pull and push modes) – for limited small information, e.g. sensor data. Small information "packets" might not necessarily be transferred through IP packet because of the addressing overhead. This feature will offer a scalable solution for a high number of connected devices.
- Caching and aggregation – to optimize network resources and for applications which are time critical, then it makes sense to optimize the information close to the terminal for all kind of data transfers. It will support locally information delivery to optimize and to reduce the end-to-end resource usage for Internet applications.

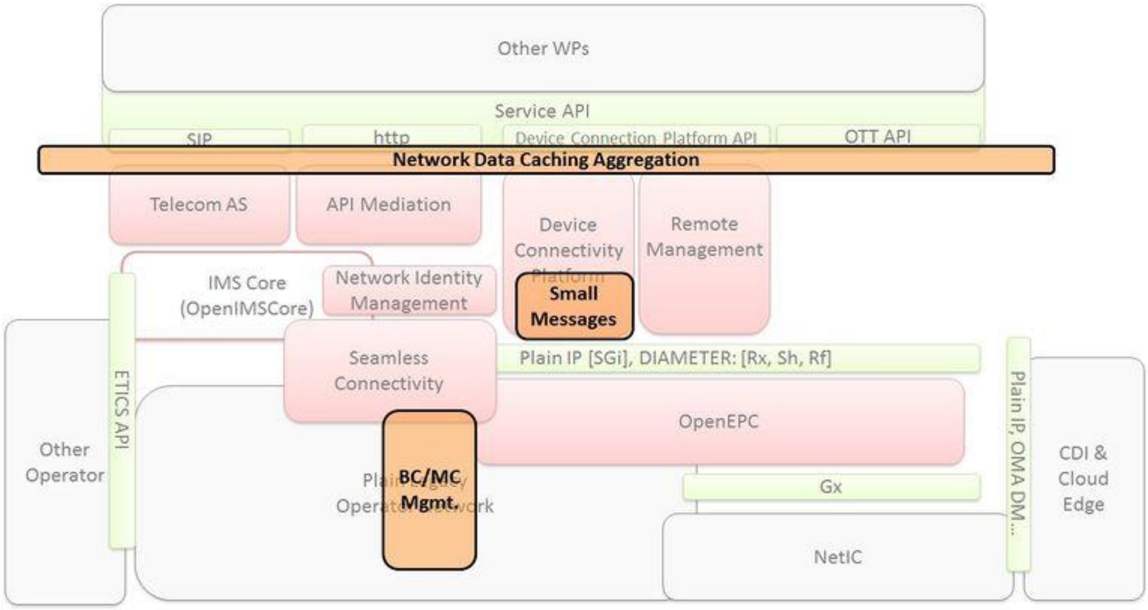


Figure 7.4.4: Feature group for special network features and issues

49.8 References

[1] IEEE “IEEE Standard for Information technology— Telecommunications and information exchange between systems— Local and metropolitan area networks—Specific requirements Part 11: Wireless LAN Medium Access Control (MAC)”, IEEE Std 802.11™-2007 (Revision of IEEE Std 802.11-1999), IEEE Standards, http://standards.ieee.org/getieee802/download/802.11-2007.pdf.

[2] ETICS Deliverable D4.2, https://www.ict-etics.eu/fileadmin/documents/publications/deliverables/D4_2_ETICS_architecture_and_functional_entities_high_level_design.pdf.

[3] Open Mobile Alliance Device Management Working Group, http://www.openmobilealliance.org/Technical/DM.aspx.

[4] Second release of FI-WARE High Level Description and reception of Requirements from UC projects, Deliverable D2.2b, http://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_Interface_to_Networks_and_Devices_(I2ND).

[5] SIP: Session Initiation Protocol, IETF Request for Comments, RFC 3261, http://www.ietf.org/rfc/rfc3261.txt.

[6] Hypertext Transfer Protocol -- HTTP/1.1, IETF Request for Comments, RFC 2616, http://www.ietf.org/rfc/rfc2616.txt.

[7] Diameter Base Protocol, IETF Request for Comments, RFC 3588, http://www.rfc-editor.org/rfc/rfc3588.txt.

[8] OMA Secure User Plane Location (SUPL), http://www.openmobilealliance.org/technical/release_program/supl_v2_0.aspx

[9] OpenIMS http://www.openimscore.org/

[10] ETICS Deliverable D4.3 https://bscw.ict-etics.eu/pub/bscw.cgi/d37005/ETICS_D4.3_v1.0.pdf

[11] Economics and Technologies for Inter-Carrier Services (ETICS) https://www.ict-etics.eu/

50 Developer Community and Tools Architecture

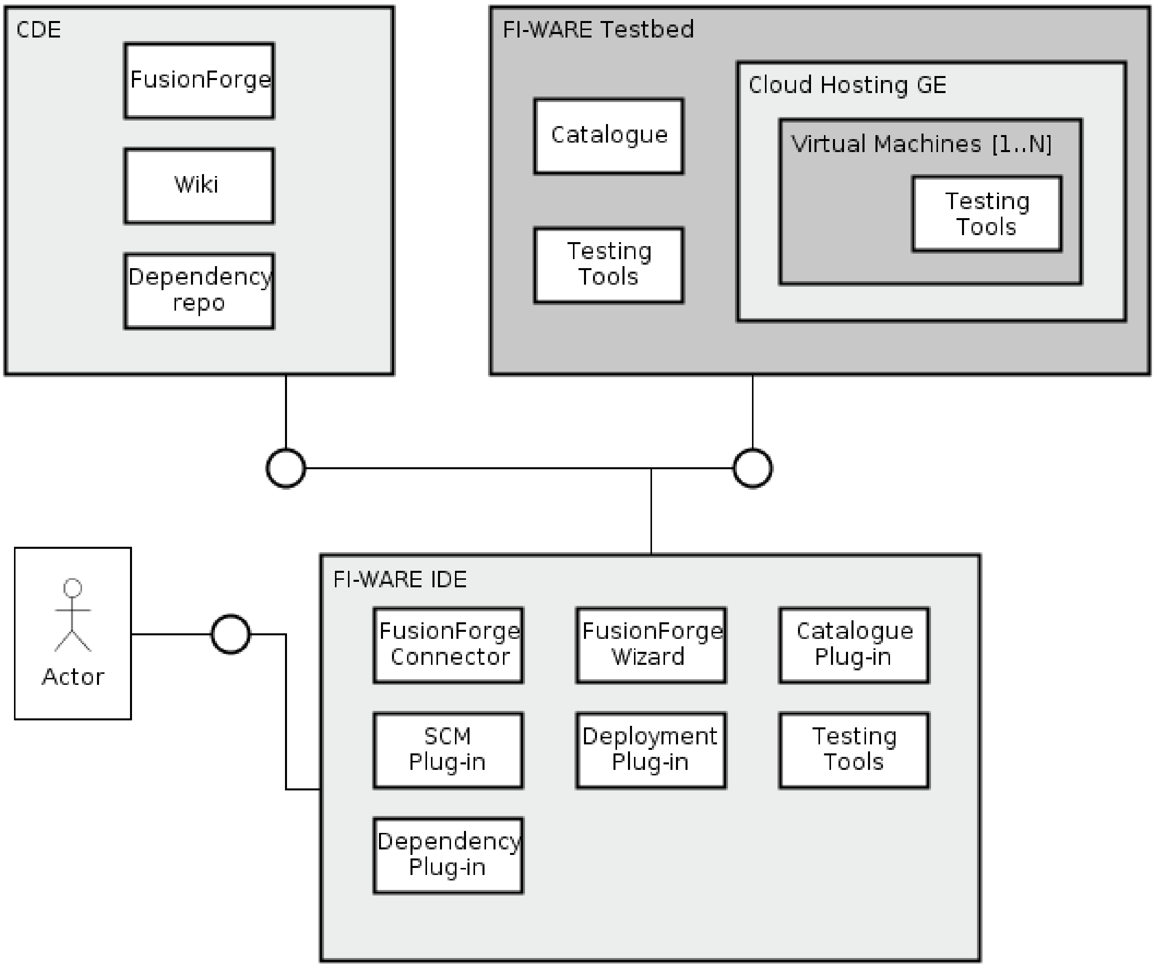
You can find the content of this chapter as well in the wiki of fi-ware.

50.1 Introduction

FI-WARE will provide various strategic and fundamental means enabling the development of Future Intenet Applications (FIApp). It is expected that around FI-WARE a community of developers will grown both to further work on the FI-WARE results (e.g. providing new Open Specification or improve the existing ones, to code new GE implementation, or set-up new FI-WARE Instances) and to develop new applications or doing business according to an open paradigm.

50.2 Architecture Overview

The primary goal of the **Developer Community and Tools** (DCT) is to offer a multi-functional development environment enabling the development and management of the **Future Internet Applications** (FIApp) built to address the needs of the Future Internet and based on the adoption and integration of the **Generic Enablers Implementation**. The DCT overview figure represents in a single diagram the main components provided to the Future Internet developer.



DCT Overview

Note: The FI-WARE Testbed infrastructure represented in DCT Overview figure is provided by a specific FI-WARE chapter.

All the functionalities are described in detail in the subsequent chapters grouped by major aspects:

- IDE-CDE: interaction between the IDE and the Collaborative Development Environment
- API and Catalogue: the Catalogue provides the API of the GEs to the development environment
- Deploy, Test and Validation: Future Internet Application deployment for testing and validation activities.

For the first version of the DCT it has been decided to restrict the target development environment to the Java language (version 1.6 and later). Additional languages will be added later according to the available effort.

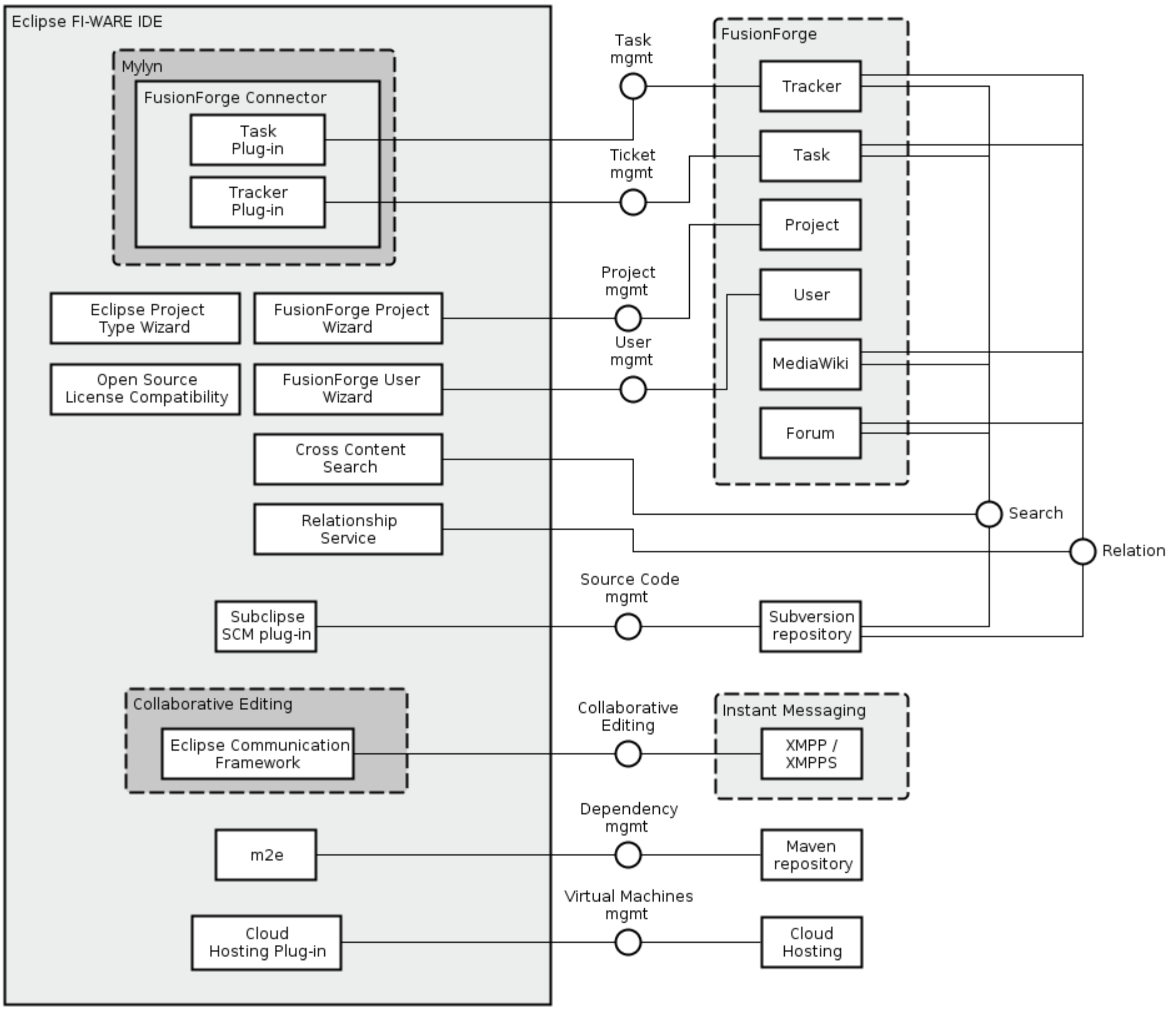
50.3 Basic Concepts

50.3.1 IDE-CDE

This chapter describes the components that support the developer in the interaction between the Collaborative Development Environment (CDE) and the IDE. The detailed figure (IDE-CDE) represents the components that are developed by FI-WARE, or third parties and integrated in a single IDE solution.

The selected reference implementation for the IDE is the **Eclipse IDE for Java EE Developers** while the CDE is composed by **FusionForge 5.x** and relative **MediaWiki** extension.

Note: FusionForge exposes its services by means of SOAP interfaces; those interfaces that were not available by default, they've been implemented and contributed to the main FusionForge source code base.



IDE-CDE Architecture

50.3.1.1 *Components*

All the subsequent components are developed in the context of the FI-WARE project but those indicated as provided by third parties. In those cases the plug-in has been selected from the leading solutions, tested, configured and integrated into the final environment in the proper way and according to the requirements.

FusionForge Connector

this component allows the developer to connect the IDE to a specific FusionForge instance in order to manage Tickets (Traker) and Tasks. It's based on the Mylyn infrastructure and the main supported features are: create a custom query to list the elements (tickets or taks); show the details of the selected element; insert and update elements. The connection with the FusionForge server is done by means of SOAP interfaces.

FusionForge Project Wizard

the developer can create, update and delete a project hosted into the FusionForge instance directly from the IDE. The connection with the FusionForge server is done by means of SOAP interfaces.

FusionForge User Wizard

the developer can create, update and delete a user account into the FusionForge instance directly from the IDE. The connection with the FusionForge server is done by means of SOAP interfaces.

Eclipse Project Type Wizard

this Eclipse extension adds to the predefined project types list the options to start with a new FI-WARE Application or a new Generic Enabler implementation. Specific meta-data are stored together with the project itself and are used by other CDT features (e.g. deployment).

Open Source License Compatibility

this plug-in supports the developer of a FLOSS project on checking and maintaining the compatibility of the licenses of the integrated components.

Cross Content Search

the developer can search for an argument in various repositories and obtain the results in a single and aggregated view within the IDE.

Relationship Service

this component allows the developer to establish a relationship among elements coming from different sources or repositories. This feature can provide an overall vision supporting the developer.

Subclipse SCM Plug-in

this plug-in allow the developer to connect with the Source Code Management system directly from the IDE, and perform the usual operations (e.g. commit, update, merge). Thanks to the Mylyn extension it's possible to reflect into the commit message some information taken from the current task/ticket under execution. This plug-in is provided by third parties and all the additional information can be found at the official web site.

Eclipse Communication Framework

this component has been selected to support the collaborative editing of the same file at the same time by two different developers. This features relies on the communication channel provided by the most common instant messaging services (e.g. XMPP/XMPPS, Google talk). This plug-in is provided by third parties and all the additional information can be found at the official web site.

m2e

this plug-in supports the developer in managing the dependencies (libraries) integrated into the project under development. This plug-in is provided by third parties and all the additional information can be found at the official web site.

Cloud Hosting Plug-in

this plug-in can be used by the developer to manage the virtual environments to use for deployment, test and validation activities. Starting from a user account it's possible to retrieve all the virtual environments created by that user, drop them and create new ones according to the requirements of the applications under development.

50.3.2 FI-WARE Catalogue

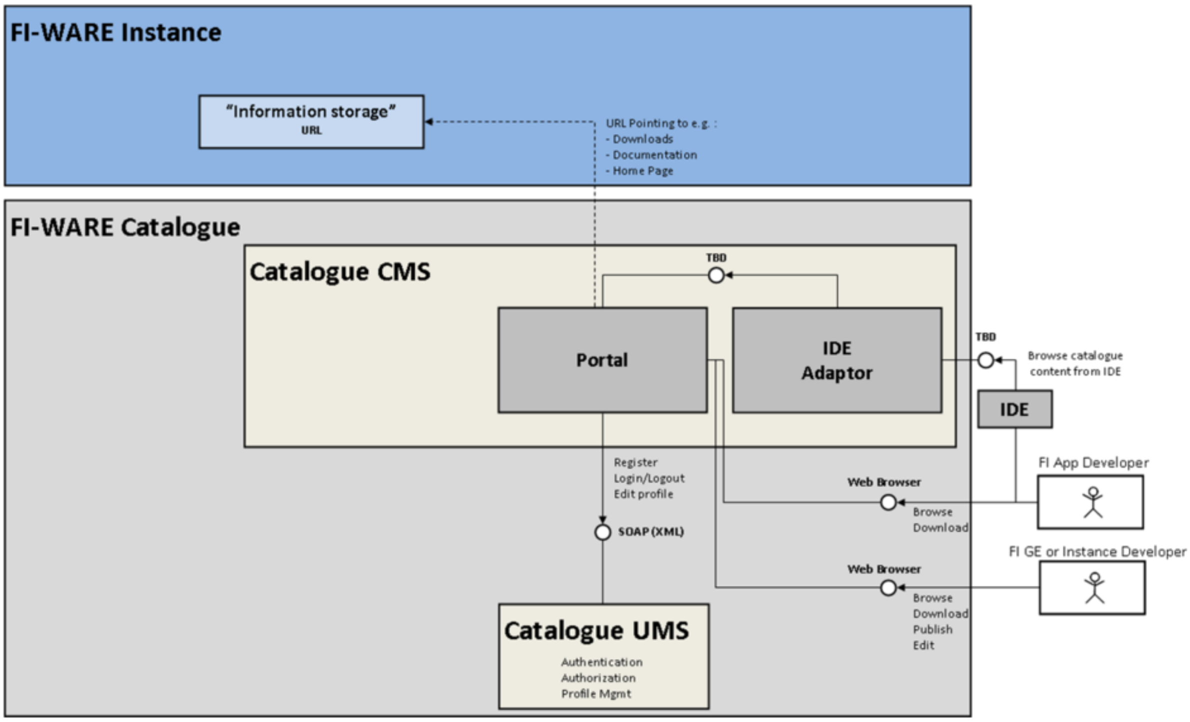
The Tools Chapter is responsible for providing tools to support the developers of Future Internet Applications (FIApp) and Generic Enablers (from now on also shortened like GE) implementations also making easy to find, navigate and understand what is offered by FI-WARE "one-shot". The idea is to provide a catalogue containing all the information required for a potential developer such as a descriptive ("selling") introduction, documentation, downloadable files etc. and allow them to be part of a community. In this way, the catalogue will have the role of FI-WARE toolbox or "one stop shop" where it will be possible to find all the means necessary for the development of FIApp and GE Implementations. The FI-WARE Catalogue could be somehow considered a replication of the Apps Marketplace provided in the Applications/Services Ecosystem and Delivery Framework chapter; this is not the case. In fact, the catalogue contains only the outcomes directly belonging from the FI-WARE project and which are compliant with the Generic Enabler Open Specifications defined. As such and according to the defition provided in [1], the Apps Marketplace GE Open Specification and its reference implementation will be part of the FI-WARE Catalogue and available for any FIApp developer which wants to use the Apps Marketplace to enable a service ecosystem for its application. Then, the catalogue is a content management system for Generic Enablers Open Specifications, Generic Enablers Implementations, Generic Enabler implemetations documentation and FI-WARE instance references (from now on simply assets - if not otherwise specified) published by assets providers and made available to developers who can browse the catalogue to find the specific assets they are interested in for download or use. It provides features for asset publishing, browsing of assets, community (optional) for discussions around these assets and references to the appropriate support functionalities available for each asset. The catalogue will also have an adaptor that enable browsing of assets directly from an IDE (Eclipse).

50.3.2.1 *Catalogue deployment view*

The catalogue will be deployed as part of the FI-WARE Testbed and expose the assets available in this testbed. The catalogue enables users (such as e.g. FIApp or GE implementations developers) to browse and interact with content as well as with FI-Ware GE and instances providers. The users can, in addition, also publish new GE implementations, info about GE Implementations and edit them. The catalogue is based on two main components:

- Catalogue Content Management System (CMS) - a portal including an adaptor for integrating with an IDE
- Catalogue User Management System (UMS)

The CMS/portal includes a publishing system and stores published content making it available for browsing. The IDE adaptor provides an interface to the IDE that enables browsing of catalogue content. The UMS takes care of registration, authentication, authorization and profile management. It is role based and it is possible to configure different roles and provide them different rights to the catalogue.



Catalogue Architecture

50.3.2.2 *Catalogue functionality*

The following sections provide more details on distinct parts of the Catalogue functionality, including the asset publishing, community functions and user management.

Asset Publishing

The Catalogue environment is in charge of publishing and making the various assets available so that these can be used to build FIApp and new assets. The catalogue will be configurable but it will by default contain the following basic information for any FI-WARE instance (e.g. FI-WARE Testbed):

- GE Open Specifications
- Downloadable files (e.g. GE implementation or reference to the API and related clients (if any))

- Supporting Documentation
- Links to the FI-WARE instances where the GE Implementation instance is available (e.g. FI-WARE Testbed)
- Community - blog and forum (optional)

This means that the FI-WARE Catalogue contains both GE Open Specifications as well as references to the FI-WARE instances where an implementation of those GE Open Specifications can be found (i.e. the FI-WARE Testbed).

Publishing is done directly in the catalogue using web-based forms. There will be two different publishing flows: one for adding new Generic Enablers (specifications, documentations, etc.) and one for adding references to FI-WARE instances. Whoever wishes to publish into the Catalogue needs to align to specified requirements on:

- Technical readiness (test&verification; …)
- Documentation
- Categorization and keywords to enable search

The publishing process includes verifying that requirements are met and, for this reason, the publishing of any GE implementation requires approval by an administrator. The publication of a reference to a FI-WARE instance is instead automatic and the reasoning is that the community can itself take care of quality assurance of instances: they will simply not use instances that are not working well. The FI-WARE Catalogue access policy is defined in order to regulate the access to the assets and related resources. Parts of the Catalogue will be visible to anyone, while parts are only available to registered users. The Catalogue will contain documentation specifying the rules for publishing and access.

Support

There is a back-end tracker system to handle support requests, which distributes requests to the right GE provider. This is not considered a part of the catalogue and will have to be handled by separate external request tracker system. The catalogue could link to this system.

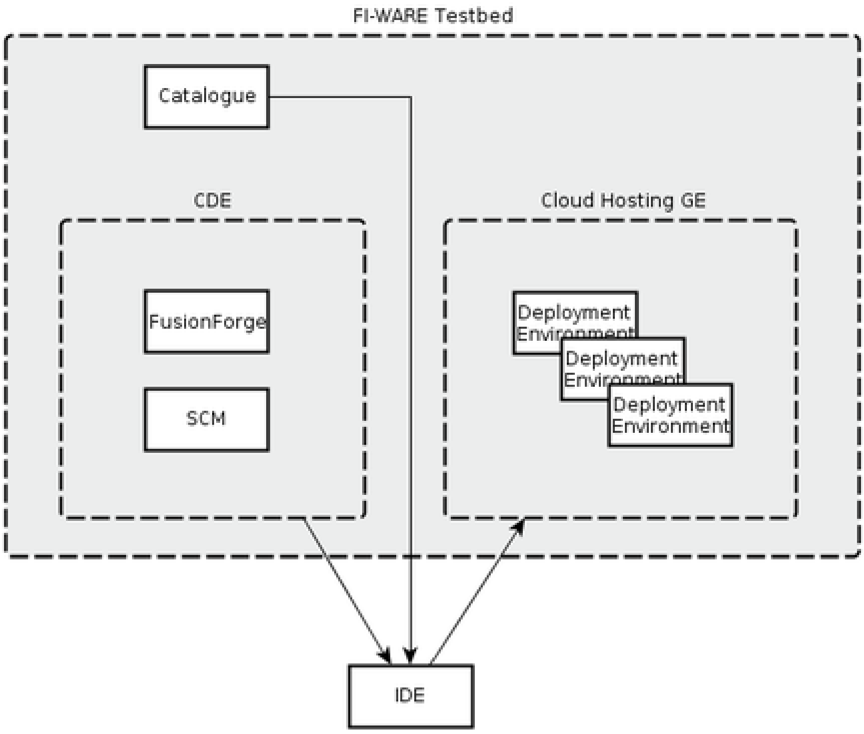
Developer Community - Optional

The catalogue also includes maintaining, monitoring and supporting the community of developers who use, develop or are just generally interested in the assets available in the Catalogue and/or the FI-WARE Instance as such. Keeping the community ‘alive’ and active is necessary in order to attract developers and drive usage. It cannot be left only up to the community itself, but some moderator activity needs to be continuously on-going. Developer community maintenance is largely not a technical issue, and so will be described in more detail in the methodology supporting the adoption of the FI-WARE SDK, here is defined only the technical tools to support it.

- News and blogs
- Developer fora/discussions
- Subscriptions

The catalogue can include linking to social media channels, to allow developers and the community responsible to share, promote and provide feedback.

50.3.3 Deployment Tool



Deployment overview

The aspects taken into account for the deployment are the ones enabling the test and validation of the the FIApp under development. The deployment steps and relative sequence are described in the methodology deliverable that supports the entire application life-cycle management of the FIApp. The implementation of these steps may vary from completely automated and tools supported ones to specific manual executions that are specifically derived from the nature of the application. In this second case, the application developer is in charge of the implementation of these steps that cannot be predefined by FI-WARE.

The deployment of an application implies the availability of the proper environment where to execute the application itself. Also in this case the nature of the application may affect the setup steps of this deployment environment. The Cloud Hosting GE provides the features and the flexibility to support the setup of the deployment environment and it is integrated by default into the FIApp development process.

The Cloud Hosting GE allows the definition of four main types of environments:

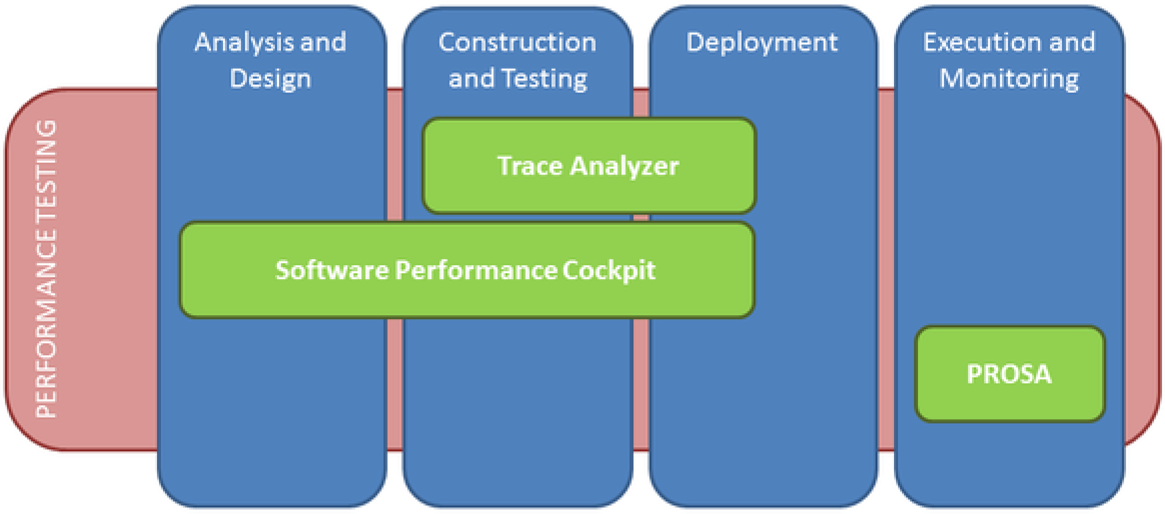
- Fixed Configuration
  - Single virtual machine
  - Composite (multiple virtual machines)
- Elastic Service
  - Self-scaling
  - Platform based scaling

Some activities for the management of these environments (e.g. create/delete environment, list/detail available environments) are directly executed from within the IDE thanks to a dedicated plug-in.

50.3.4 Testing and Validation Approach

While we can assume that state-of-the-art testing will be performed on FI-WARE applications under development (like functional module tests), this activity will focus on testing activities that are specific to the nature and architecture of FI-WARE applications and instances.

These are characterized by multiple distributed components and services running on a cloud environment. As another challenge, testing approaches suited for FI-WARE applications must cover the different stages of a FI App's lifecycle: (1) Analysis and Design, (2) Construction and Testing, (3) Deployment and (4) Execution and Monitoring. With these demands, different testing technologies can be defined in the context of FI-WARE as shown in Figure "Performance Testing Approach". These tools all focus on QoS-related aspects of a FI App because service quality is one of the main challenges in multi-stakeholder environments such as FI-WARE instances.



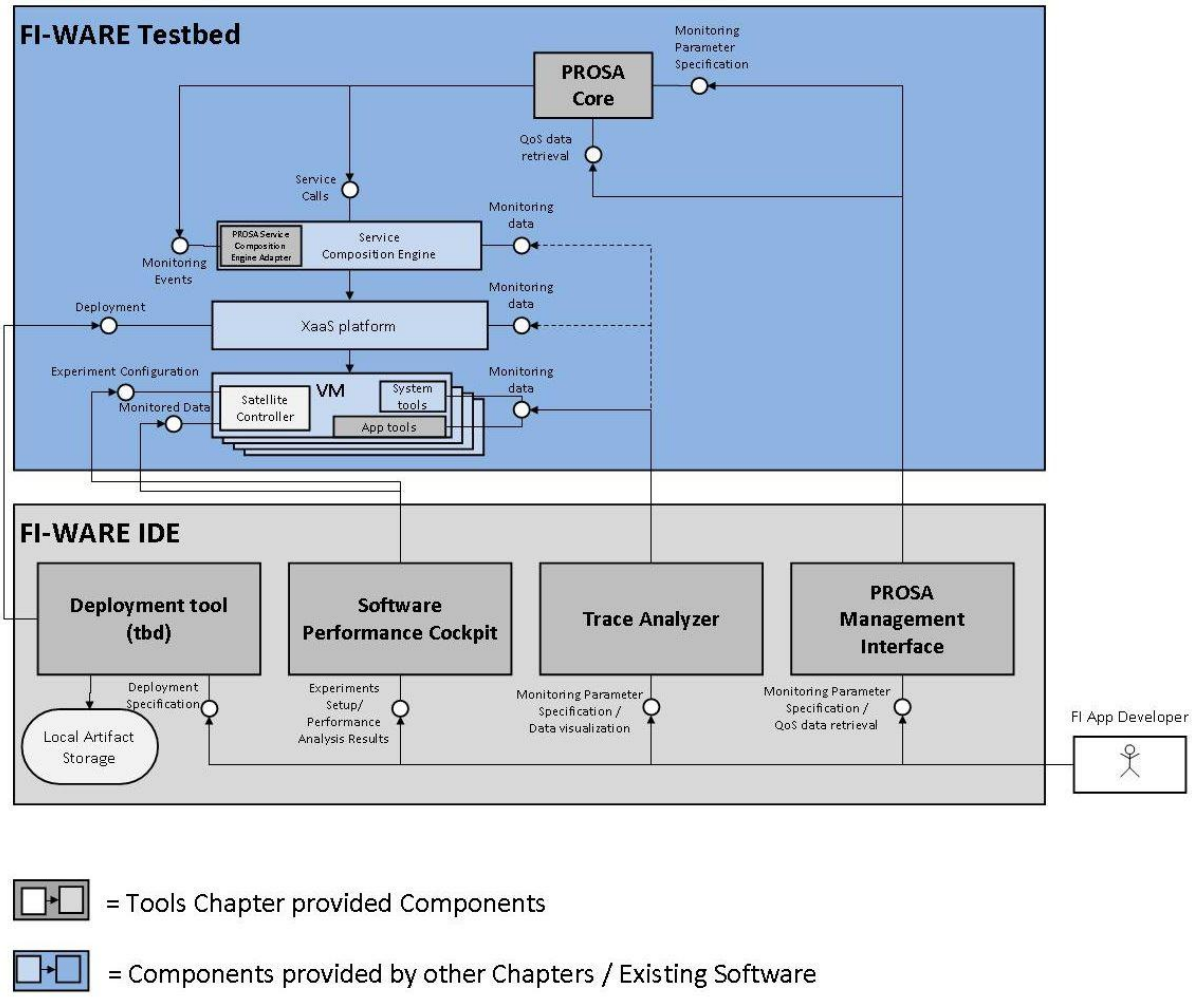
The Performance Testing Approach

The **Software Performance Cockpit (shortly named SoPeCo)** will allow the execution and statistical analysis of complex test scenarios for describing the performance behavior of a chosen aspect of an application. In FI-WARE, it mainly supports the performance evaluation of a FI App in the Construction and Deployment stage as well as the comparison and selection of different GE implementations.

The **Trace Analyzer** is a performance visualization tool allowing the manual analysis of performance-related application problems. By combining several sources of performance data, it helps the FI App developer to identify the causes for performance problems at the Development and Deployment stage.

The **PRO-active Self-Adaptation (shortly named PROSA)** is an online testing framework targeted to ensure the constant availability of QoS monitoring data for a given service (used by a FI App). The Runtime solution monitors the (real) usage of a given service and additionally performs additional service invocations (online testing) where necessary. PROSA thereby ensures that a minimum set of QoS data points is available for the given service for every time interval.

The following figure "Testing tool architecture and embedding into the FI-WARE platform" shows the functional building blocks of the FI-WARE testing framework as well as their integration into the overall FI-WARE platform. The blue/gray color coding reflects whether a component is to be provided by the DCT chapter (gray) or provided by another FI-WARE chapter or developer commmunity respectively (blue).



Testing tool architecture and embedding into the FI-WARE platform

The following component descriptions will provide more details on the graphical description:

- **Trace Analyzer** is a performance visualization tool (realized as an Eclipse plugin) used for performance testing and analysis during development, deployment and run time. Trace Analyzer receives data collected across different layers (h/w, os, middleware, services) and provides visualizations for the collected data, allows filtering and aggregating the collecting data, highlights performance anti-patterns, and identifies bottlenecks in the execution.
- **System monitoring** is a set of existing Linux tools (tcpdump and sar) that generates structured logs with data on the operating system execution characteristics such as cpu breakdown, and the service communication events. These logs will be analyzed and visualized with TA to help the user detecting OS and communication bottlenecks that may be addresses by reconfiguration.

- **App monitoring** is a set of libraries running with the application on the VM that will generate a log file with application level events such as synchronization events. These logs will be analyzed and visualized with TA to help the user understanding his application behavior and detecting application bottlenecks such as hot synchronization.
- **PROSA** is a background service running inside the FI-WARE platform (or FI-WARE Testbed), monitoring the QoS data of outbound service calls of service compositions in a service composition engine. By specifying monitoring parameters for specific services, PROSA is provided with the relevant data to ensure that the requested monitoring interval is maintained. PROSA therefore completes the set of monitoring data gathered from actual service utilization with data generated from online test executions (performed by the PROSA framework itself).
- The **PROSA Management Interface** is a client component inside the FI-WARE IDE providing the user the functionality to (1) manage the services / service composition engines monitored by the PROSA platform, (2) specify monitoring parameters for the monitored services and (3) view/analyze the historical QoS data.
- The **Software Performance Cockpit** is a framework for systematic performance evaluations of software systems. It encapsulates the knowledge about performance engineering, the system under test, and statistical analyses in a single framework and guides the performance analyst in conducting systematic performance evaluations. Adopting a plug-in based architecture, it allows the separation of concern and supports the reuse of performance evaluation artifacts.
- The **Satellite Controller** is a service component on the target machine that controls a set of software-adapters to access the measurement-related software installed on the machine. The Satellite Controller awaits calls from the experiment controller in SoPeCo, executes performance experiments on the target system, and returns the measurement results back to the experiment controller.

50.4 Main Interactions

50.4.1 FusionForge Connector

The FusionForge Connector is based on a SOAP client that enable the direct interaction with the specific FusionForge instance. The main interfaces that are used by that component are listed below.

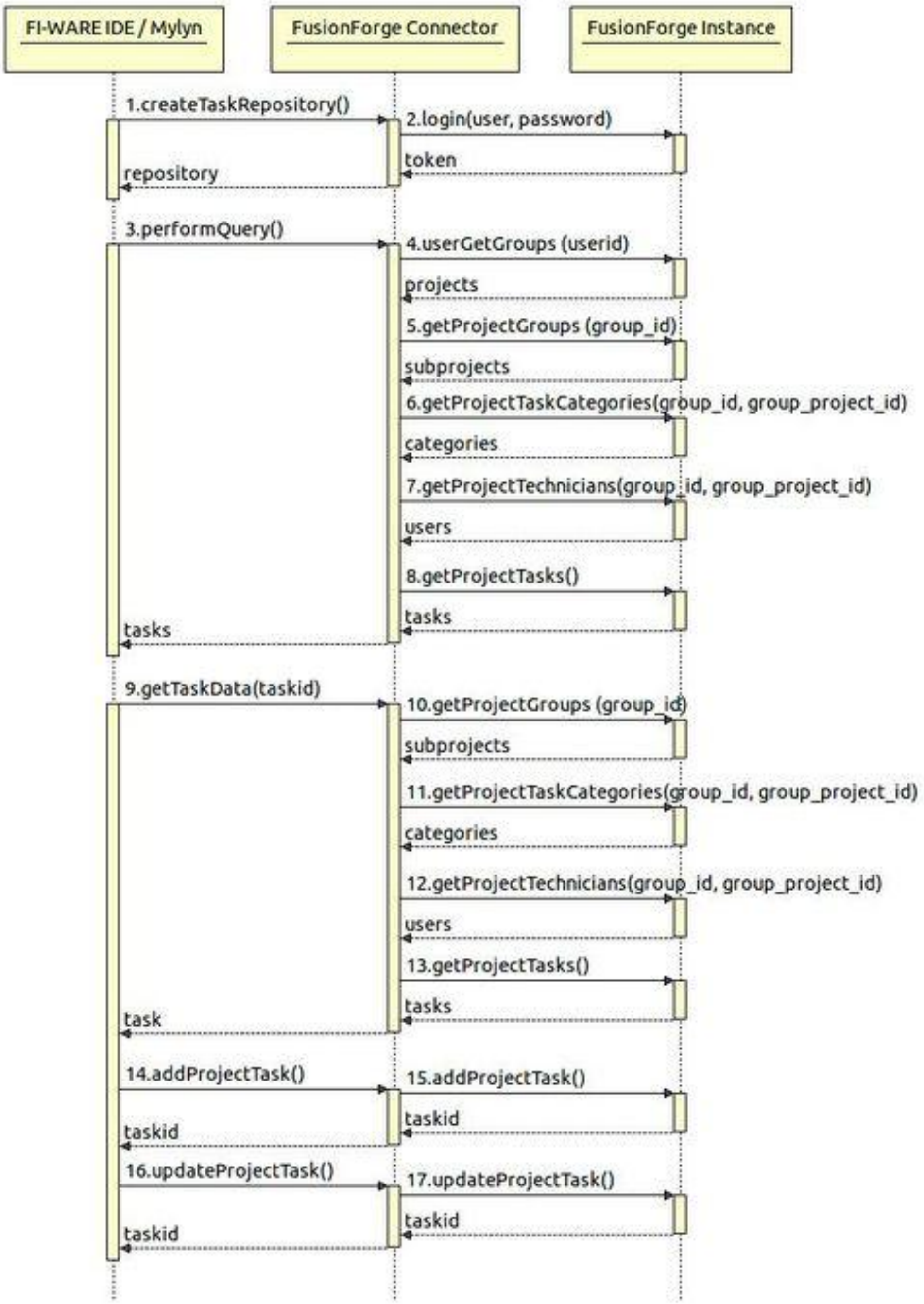
Validate Connection and Query:

- ***login***
- ***userGetGroups***

Task:

- ***getProjectTasks***
- ***addProjectTask***
- ***updateProjectTask***
- ***getProjectTaskCategories***
- ***getProjectGroups***

- ***getProjectTechnicians***



Task - Sequence diagram

Tracker/Context:

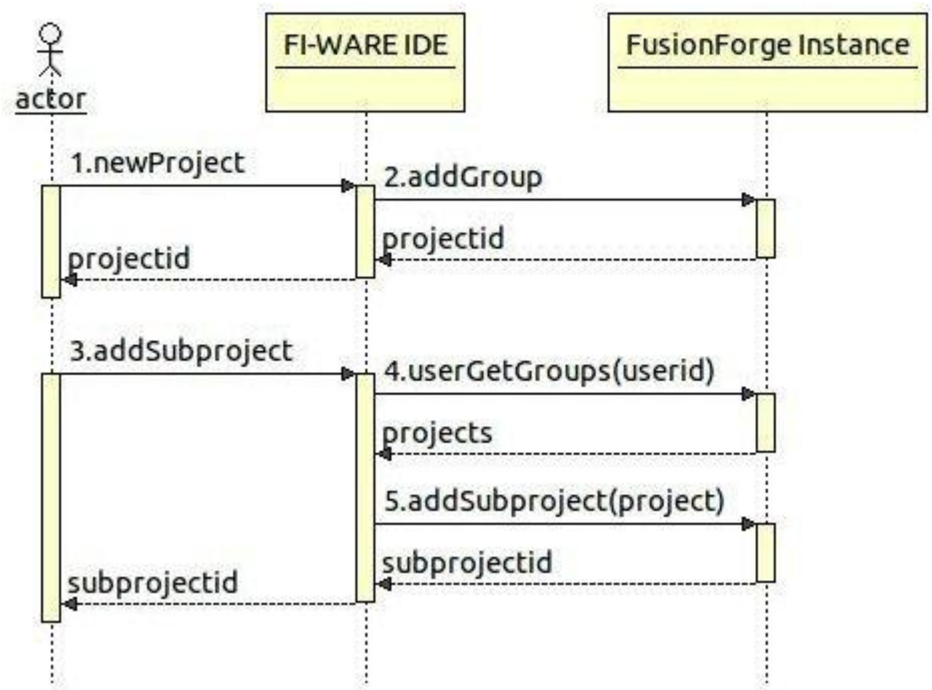
- ***getArtifactTypes***
- ***getArtifacts***
- ***addArtifact***

- ***updateArtifact***
- ***getArtifactFiles***
- ***getArtifactFileData***
- ***addArtifactFile***
- ***artifactFileDelete***
- ***getArtifactMessages***
- ***addArtifactMessage***
- ***getArtifactTechnicians***
- ***artifactSetMonitor***
- ***artifactIsMonitoring***
- ***artifactDelete***

The same sequence diagram presented for the Task management is applied for also for the Tickets management. The one for the Tickets will differ from the one for the Task for those interactions that retrieve specific attributes and fields (e.g. custom fields) that are not defined for the Task element.

50.4.2 FusionForge Project Wizard

The FusionForge Connector is based on a SOAP client that enable the direct interaction with the specific FusionForge instance. The main interfaces that are used by that component are listed below.



Project Wizard - Sequence diagram

Project:

- ***addGroup***

- ***updateGroup***
- ***deleteGroup***

Subproject:

- ***addSubproject***
- ***updateSubproject***
- ***deleteSubproject***

50.4.3 FusionForge User Wizard

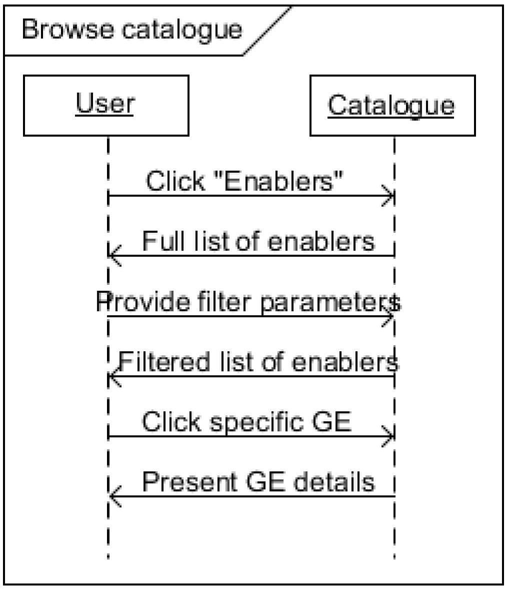
The FusionForge Connector is based on a SOAP client that enable the direct interaction with the specific FusionForge instance. The main interfaces that are used by that component are listed below.

- ***getUsers***
- ***getUsersByName***
- ***addUser***
- ***updateUser***
- ***deleteUser***
- ***changeStatus***
- ***changePassword***
- ***userGetGroups***

50.4.4 Catalogue

The catalogue enables GE providers to publish assets.

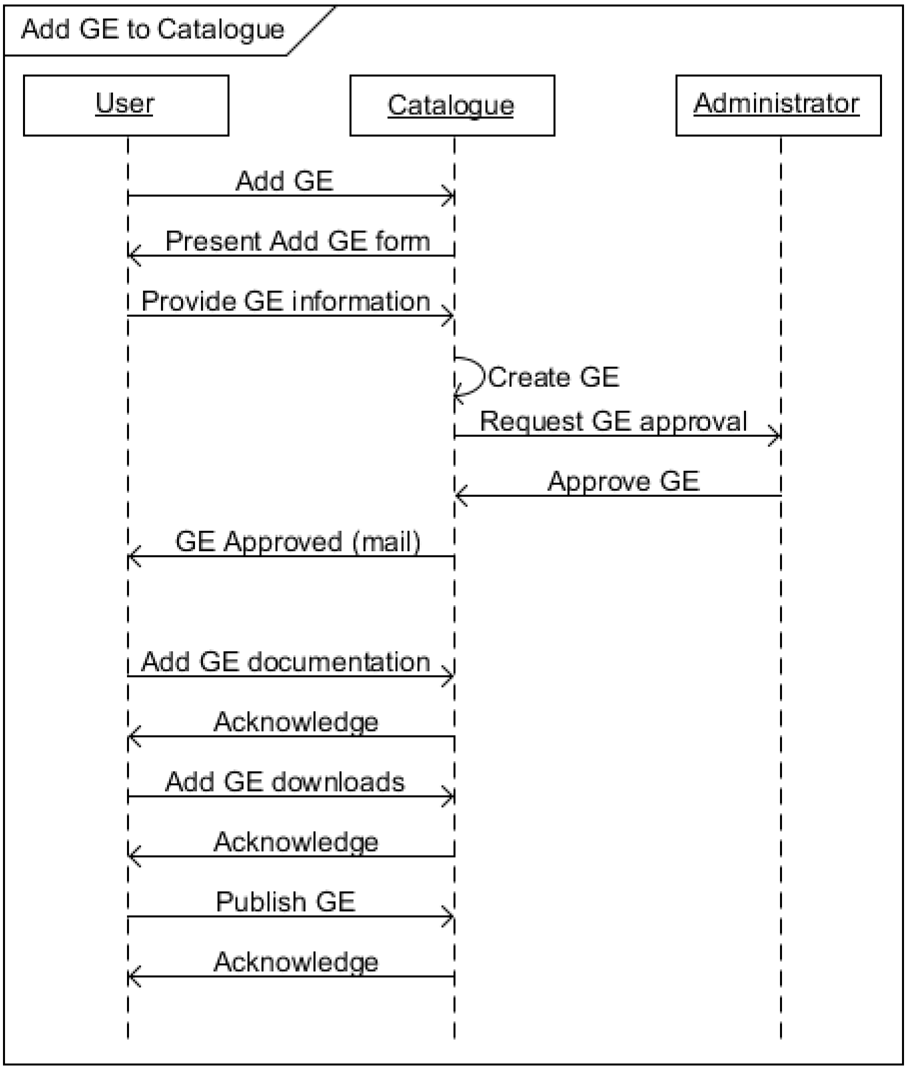
50.4.4.1 ***Browse Catalogue***



Browse Catalogue sequence diagram

50.4.4.2 ***Add GE to Catalogue***

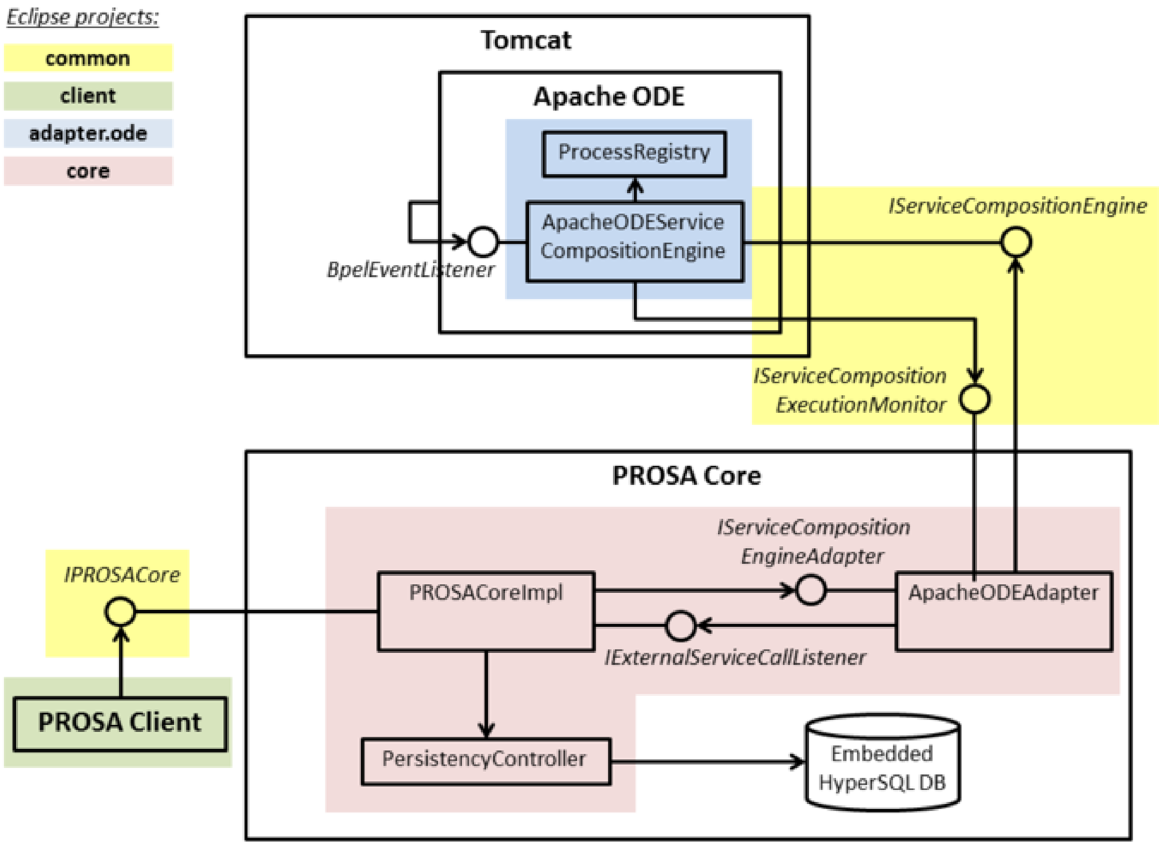
The following diagram presents all the sequence of methods invocation necessary to upload into the FI-WARE catalogue all the information about a GE: open specifications, implementation, documentation, etc.



Add GE to Catalogue sequence diagram

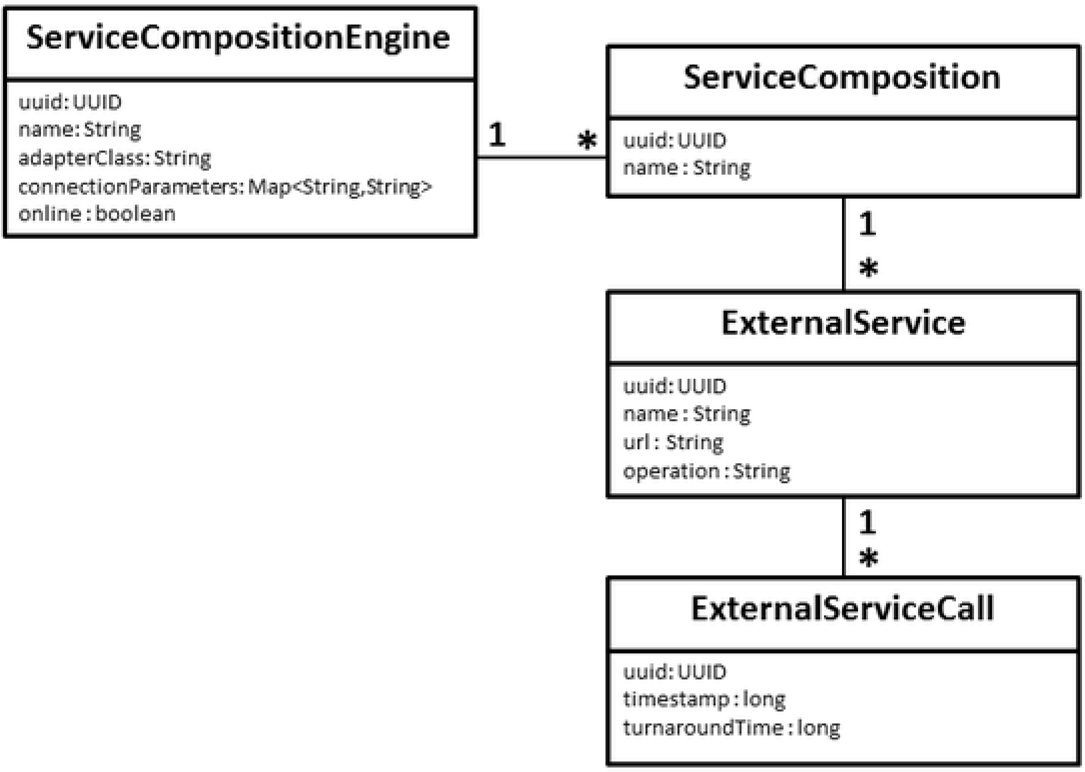
50.4.5 PROSA

50.4.5.1 *Architecture*



PROSA - internal architecture

50.4.5.2 *Data model*



PROSA - data model

50.4.5.3 *Interface descriptions*

PROSA Service Composition Engine Adapter (IServiceCompositionEngine)

The PROSA Service Composition Engine Adapter offers a Java RMI interface for the PROSA Core to retrieve Service Composition metadata and register as an execution monitor.

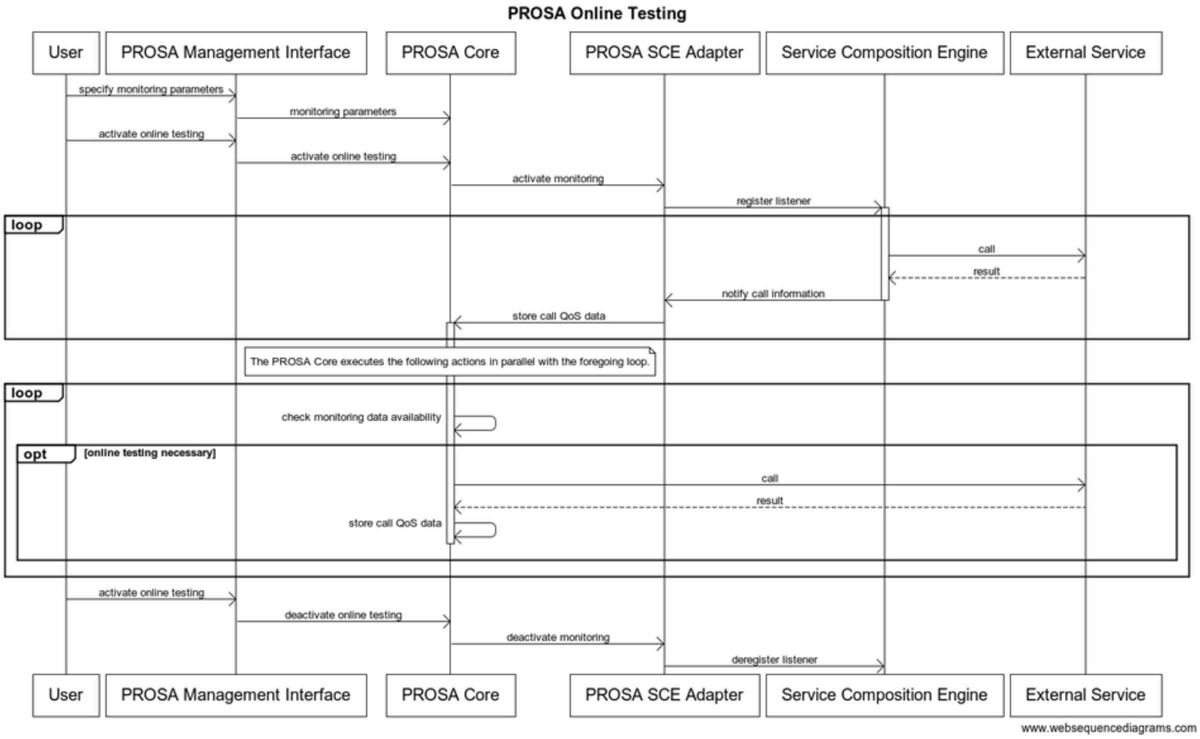
- ***void init(ServiceCompostionEngine engineData)***
- ***boolean isAlive()***
- ***Set<ServiceComposition> getAllServiceCompositions()***
- ***void registerMonitor(ServiceCompositionExecutionMonitor monitor)***
- ***void deregisterMonitor(ServiceCompositionExecutionMonitor monitor)***
- ***void setServiceCompositionMonitoring(String serviceCompositionId, boolean enabled)***

PROSA Core (IPROSACore)

- ***ServiceCompostionEngine addServiceCompositionEngine(ServiceCompostionEngine engine)***
- ***Set<ServiceCompostionEngine> getServiceCompostionEngines()***

- ***Set<ServiceComposition> getServiceCompositions(UUID serviceCompostionEngineUUID)***
- ***boolean testConnection(UUID serviceCompositionEngineUUID)***
- ***void setOnlineTestingEnabled(UUID serviceCompositionUUID, boolean enabled)***
- ***List<ExternalServiceCall> getAllCalls(UUID externalServiceUUID)***
- ***Set<ExternalService> getExternalServices(UUID serviceCompositionUUID)***

50.4.5.4 *Interaction*

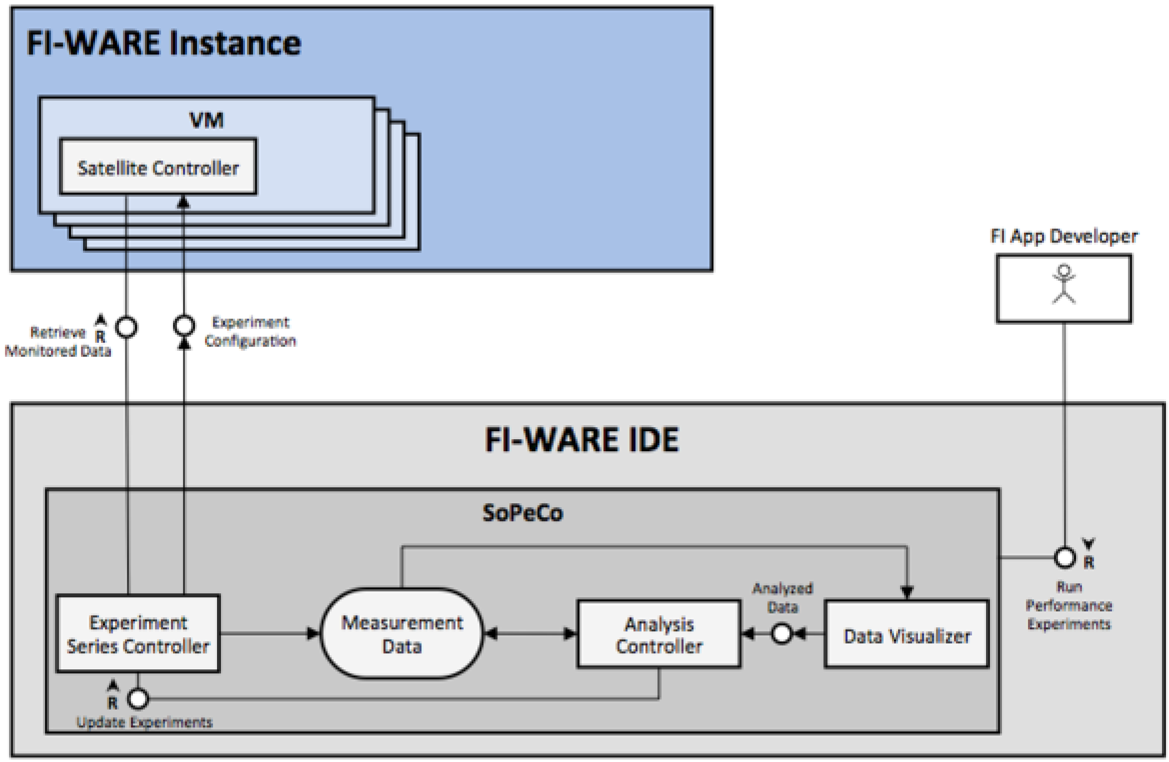


PROSA - sequence diagram

50.4.6 SoPeCo

50.4.6.1 *Main Components*

SoPeCo facilitates systematic performance evaluation of software systems. The high-level architecture of SoPeCo including its four components in the context of FI-WARE is shown here. The FI App developer can use SoPeCo to define experiment configurations, and then systematically run those experiments, gather the results and use the analysis controller and the data visualizer to view the outcome.



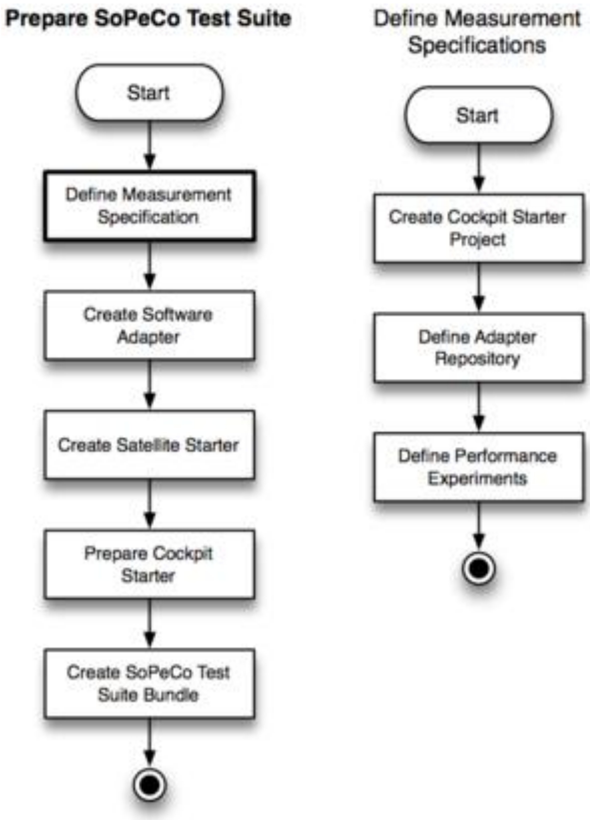
SoPeCo Architecture in FI-WARE

50.4.6.2 *Prepare SoPeCo Test Suites*

The process of preparing SoPeCo test suites is summarized in the following two diagrams. The user has to first define measurement specifications, which further details into defining the interface of the software adapters (in terms of input/out parameters of interest) and the specification of the experiments. For each experiment, a set of input parameters of interest, value variation strategies for each parameter, and the output parameter of interest are identified.

After defining the measurement specifications, a software adapter will be developed that serves as the interface between the target system (software under test) and the SoPeCo framework. Software adapters are called by the framework to run single experiments on the target system. In order to perform test on remote machines, satellite starters are developed that act as a service between the framework and the machine on which the performance is monitored. SoPeCo satellites then pass on the requests to the corresponding software adapters. In simple configurations where one has only one adapter per physical machine, the satellite and the adapter can be combined into a single component.

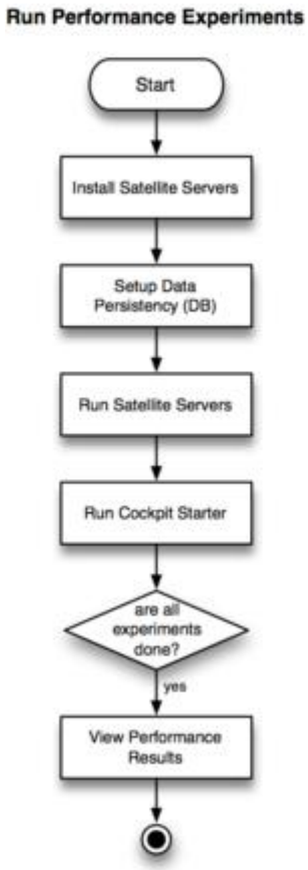
Finally, the developer should be able to bundle up these pieces into re-usable performance test packages.



Prepare SoPeCo Test Suites

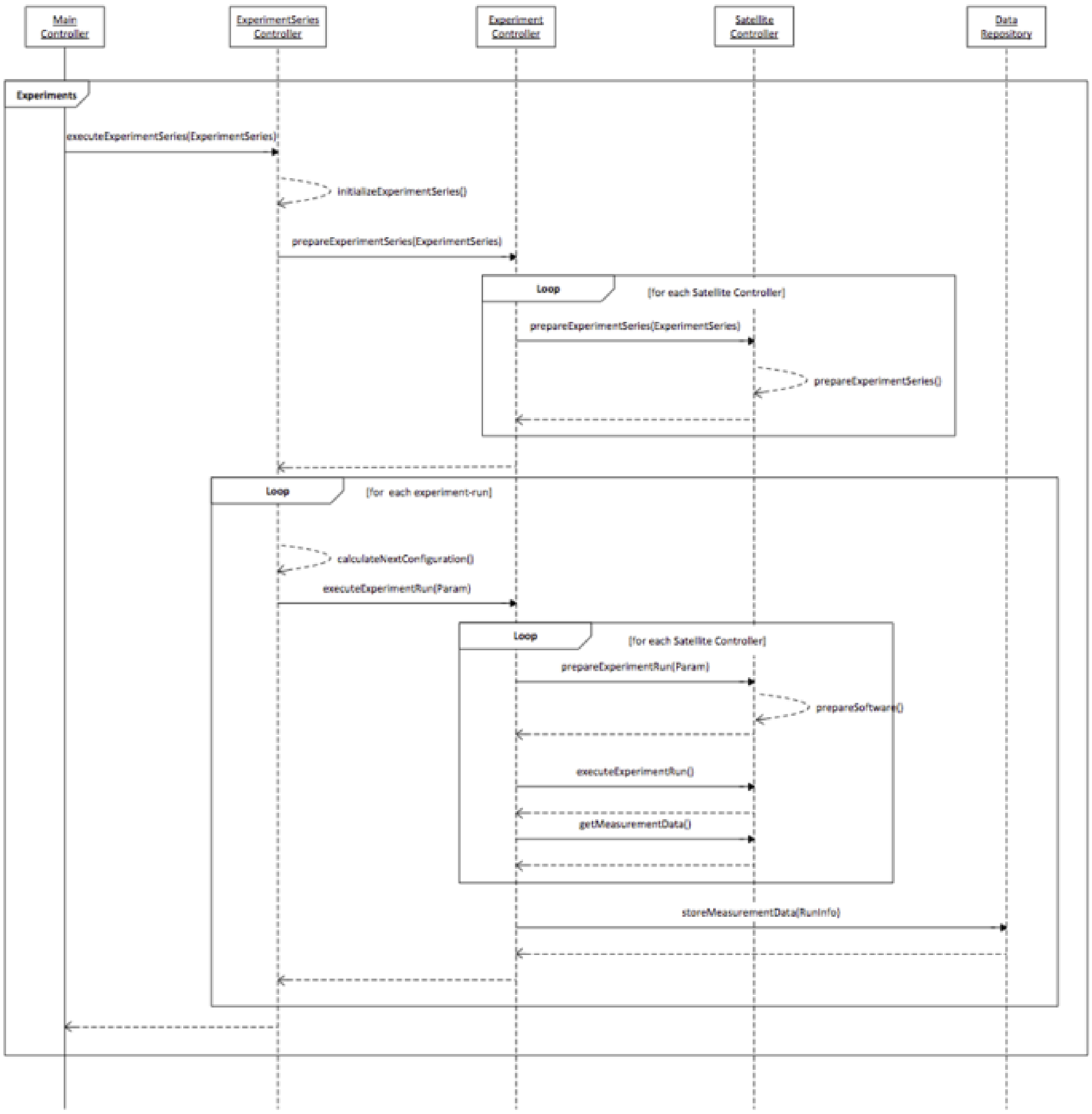
50.4.6.3 *Run Performance Test Suites*

In order to run performance tests, satellites have to be installed and running on the target machines. If needed, a database can be set up and the cockpit starter can be configured to use the database for data persistency. The cockpit starter then will be executed which automatically communicates with the satellites and commands the execution of experiments. Once the data is gathered, the data visualizer component can be used to observe and analyze the results.



Run Performance Experiments

To better understand the concept of experiments, the following sequence diagram further explains how the experiment executions are handled in SoPeCo.



Execute Experiments

50.4.7 Trace Analyzer

Trace analyzer plug-in has two major modules: the first one is the Trace Modeler that reads the trace, generates internal model of the trace and provide data on demand. The second one is trace analyzer that visualize the data in the trace model according to the selected view, or statistical query.

50.4.7.1 *Trace Modeler*

- ***buildTraceModel(trace)***
- ***getViewData(view_identifier)***
- ***getRecordDetails(record_id)***

- ***getStatistic(parameters)***

50.4.7.2 ***Trace Analyzer***

- ***analyzeTCP(interface, port)***
- ***analyzeOS()***
- ***selectView from the list: {TCP_sessions, TCP_packets, OS_CPU, OS_mem}***
- ***getRecordDetails(record_id)***
- ***getStatistic(query)***
- ***zoomIn(interval)***
- ***zoomOut***

50.4.7.3 ***System Monitoring***

In order to collect monitoring data Trace analyzer uses the tcpdump and sar tools in the following way:

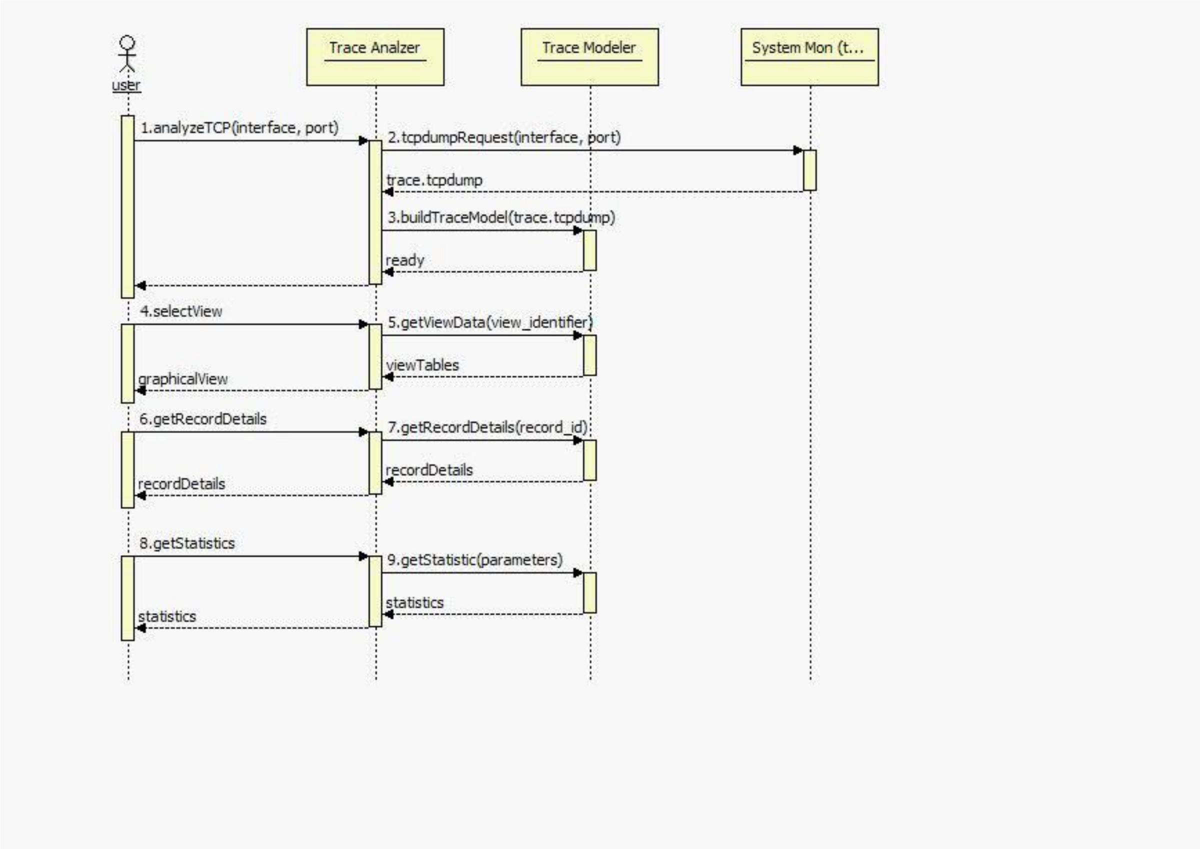
- tcpdumpReq(interface, portNum) is implemented by

```
tcpdump –i <interface> port <portNum>
```

- sarReq(intervals) is implemented by

```
sar –A <intervalLen> <intervalNum>
```

The following sequence diagram describe one of Trace Analyzer usage scenario.



Trace Analyzer - usage scenario sequence diagram