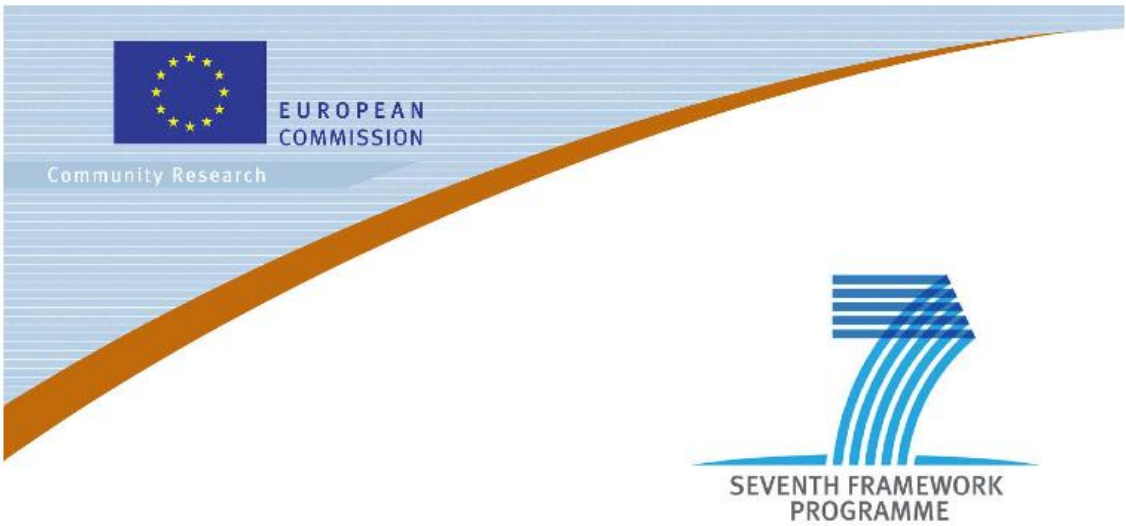


Private Public Partnership Project (PPP)

Large-scale Integrated Project (IP)

D.5.1.3: FI-WARE GE Open Specification

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-WP5-D.5.1.3
Project Document Date: 2014-11-20
Deliverable Type and Security: Public
Author: FI-WARE Consortium
Contributors: FI-WARE Consortium

1.1 Executive Summary

This document describes the Generic Enablers in the Internet of Things Service Enablement chapter, their basic functionality and their interaction. These Generic Enablers form the core business framework of the FI-WARE platform by supporting the business functionality for commercializing services.

The functionality of the frame work is illustrated with several abstract use case diagrams, which show how the individual GE can be used to construct a domain-specific application environment and system architecture. Each GE Open Specification is first described on a generic level, describing the functional and non-functional properties and is supplemented by a number of specifications according to the interface protocols, API and data formats.

1.2 About This Document

FI-WARE GE Open Specifications describe the open specifications linked to Generic Enablers GEs of the FI-WARE project (and their corresponding components) being developed in one particular chapter.

GE Open Specifications contain relevant information for users of FI-WARE to consume related GE implementations and/or to build compliant products which can work as alternative implementations of GEs developed in FI-WARE. The later may even replace a GE implementation developed in FI-WARE within a particular FI-WARE instance. GE Open Specifications typically include, but not necessarily are limited to, information such as:

- Description of the scope, behavior and intended use of the GE
- Terminology, definitions and abbreviations to clarify the meanings of the specification
- Signature and behavior of operations linked to APIs (Application Programming Interfaces) that the GE should export. Signature may be specified in a particular language binding or through a RESTful interface.
- Description of protocols that support interoperability with other GE or third party products
- Description of non-functional features

1.3 Intended Audience

The document targets interested parties in architecture and API design, implementation and usage of FI-WARE Generic Enablers from the FI-WARE project.

1.4 Chapter Context

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.

Please note that all the following chapters will use the same vocabulary and term definitions according to 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.

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.

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 proprietary 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.

Gateway

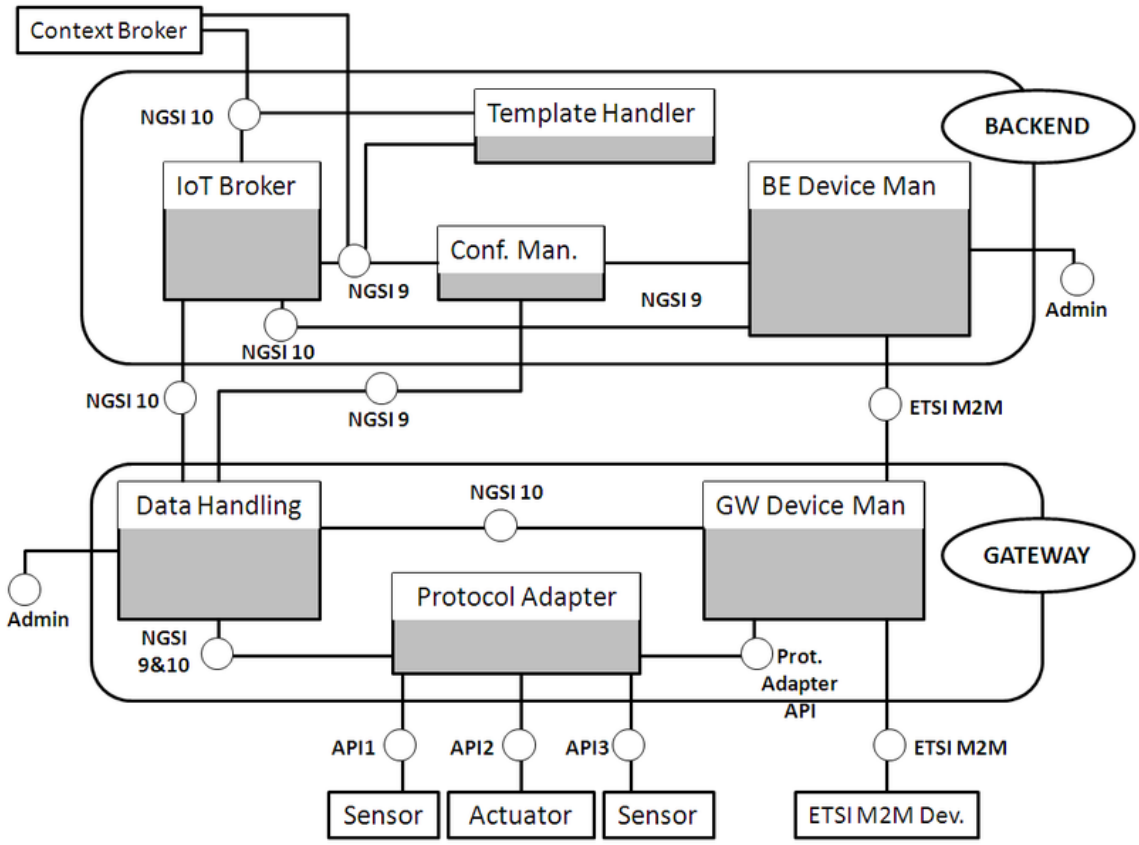
A gateway is providing inter-networking and protocol conversion functionalities between devices and the IoT backend. It is usually located at proximity of the devices to be connected. An example of an IoT gateway is a home gateway that may represent an aggregation point for all the sensors/actuators 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 communicate 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 to the things, to emphasize smart applications development.

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.



More information about the IoT Service Enablement Chapter and FI-WARE in general can be found within the following pages:

http://wiki.fi-ware.org

Internet_of_Things_Services_Enablement_Architecture

Materializing_Internet-Of-Things-Services-Enablement_in_FI-Ware

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:

D.5.1.3_FI-WARE_GE_Open_Specifications_front_page

FIWARE.OpenSpecification.IoT.Backend.IoTBroker

FIWARE.OpenSpecification.IoT.Backend.ConfMan

FIWARE.OpenSpecification.IoT.Backend.DeviceManagement

FIWARE.OpenSpecification.IoT.Backend.TemplateHandler

FIWARE.OpenSpecification.IoT.Gateway.DeviceManagement

FIWARE.OpenSpecification.IoT.Gateway.DataHandling

FIWARE.OpenSpecification.IoT.Gateway.ProtocolAdapter

FI-WARE NGSI Open RESTful API Specification

IETF_CoRE

ETSI_M2M_mId_Open_RESTful_API_Specification

FI-WARE_Open_Specification_Legal_Notice_(essential_patents_license)

FI-WARE_Open_Specification_Legal_Notice_(implicit_patents_license)

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

All partners within the IoT chapter contributed to this deliverable.

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
v1	First draft of deliverable submission	2014-02-26	TID
V2	Second draft – internal peer-review	2014-03-20	TI
V3	Final version M36	2014-05-01	Orange
V4	Third draft including Open specs 2nd wave	2014-05-28	Orange
V5	Fourth draft – internal peer review	2014-06-15	TI
V6	Fifth draft – inclusion of comments	2014-08-10	Orange
V7	Final version M40	2014-09-02	Orange

1.10 Table of Content

Contents

2 FIWARE.OpenSpecification.IoT.Backend.IoTBroker

Name	FIWARE.OpenSpecification.IoT.Backend.IoTBroker		
Chapter	IoT Services Enablement,		
Catalogue-Link to Implementation	IoT Broker	**Owner**	NEC, Tobias Jacobs

2.1 Preface

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 interactions enabled by the ubiquity of heterogeneous and resource-constrained devices.

From a physical architecture standpoint, IoT GEs have been spread in two different domains:

- **FI-WARE IoT Gateway.** A hardware device that hosts a number of features of one or several Gateway Generic Enablers of the IoT Service Enablement. It is usually located at proximity of the devices (sensors/actuators) to be connected. In the FI-WARE IoT model, the IoT Gateway is an optional element aiming to optimize the network traffic sent to the Backend and to IoT services, with the purpose of reaching higher efficiency and reliability. None, one or more IoT Gateways can be part of a FI-WARE IoT setting. Several M2M (Machine to Machine) technologies introduce specific gateway devices too; these solutions may be adopted when it is not feasible to install FI-WARE gateway features. These gateways are considered plain devices grouping other devices;, though supported, they are not strictly FI-WARE IoT Gateways.

- **FI-WARE IoT Backend.** IoT Backend is located in the cloud that hosts a number of features of one or several Generic Enablers of the IoT Service Enablement. It is typically part of a FI-WARE platform instance in a Datacenter. In the FI-WARE IoT model, at least one IoT Backend is mandatory, which will be connected to all IoT end devices either via IoT Gateway(s) and/or straight interfaces. Normally, during FI-WARE Releases (R1 to R3), the Backend will refer to the IoT Backend enablers installed in the FI-WARE Testbed or Open Innovation Lab (OIL), as described in the project Catalogue.

A key design statement is that, whenever present, IoT Gateways are not expected to be permanently connected to the Backend as per communications design or because of failures. Another relevant remark is that IoT Gateways are expected to be

constrained devices in some scenarios. Therefore, light-weight implementations of the same GEs, as in the IoT Backend, plus additional GEs interfaces helping to save unnecessary features/GEs are specially considered in the Gateway domain.

From the functionality point of view, FI-WARE IoT design aims to expose the "Things" abstraction to services developers, cope with different vertical M2M applications and provide a uniform access to heterogeneous M2M hardware and protocols. There is a number of IoT features which are somehow duplicated in the Backend and the Gateway domains in order to fulfill the goals and statements described above. For instance, a CEP engine at the Gateway level reduces the network overload and improves condition-based-events triggering time. Application developers will be able to access Things and devices observation and control interfaces in two ways:

- Directly, by using Northbound IoT interfaces as described in this document.
- Throughout Data/Context GEs, by configuring Backend IoT GEs (IoT Broker) as Context Providers of NGSI notifications sent by the Context Broker GE defined in the FI-WARE Data/Context chapter.

Nota Bene: For the reader, we are using in the following chapters the same vocabulary as 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 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.

2.2 Copyright

2.3 Legal Notice

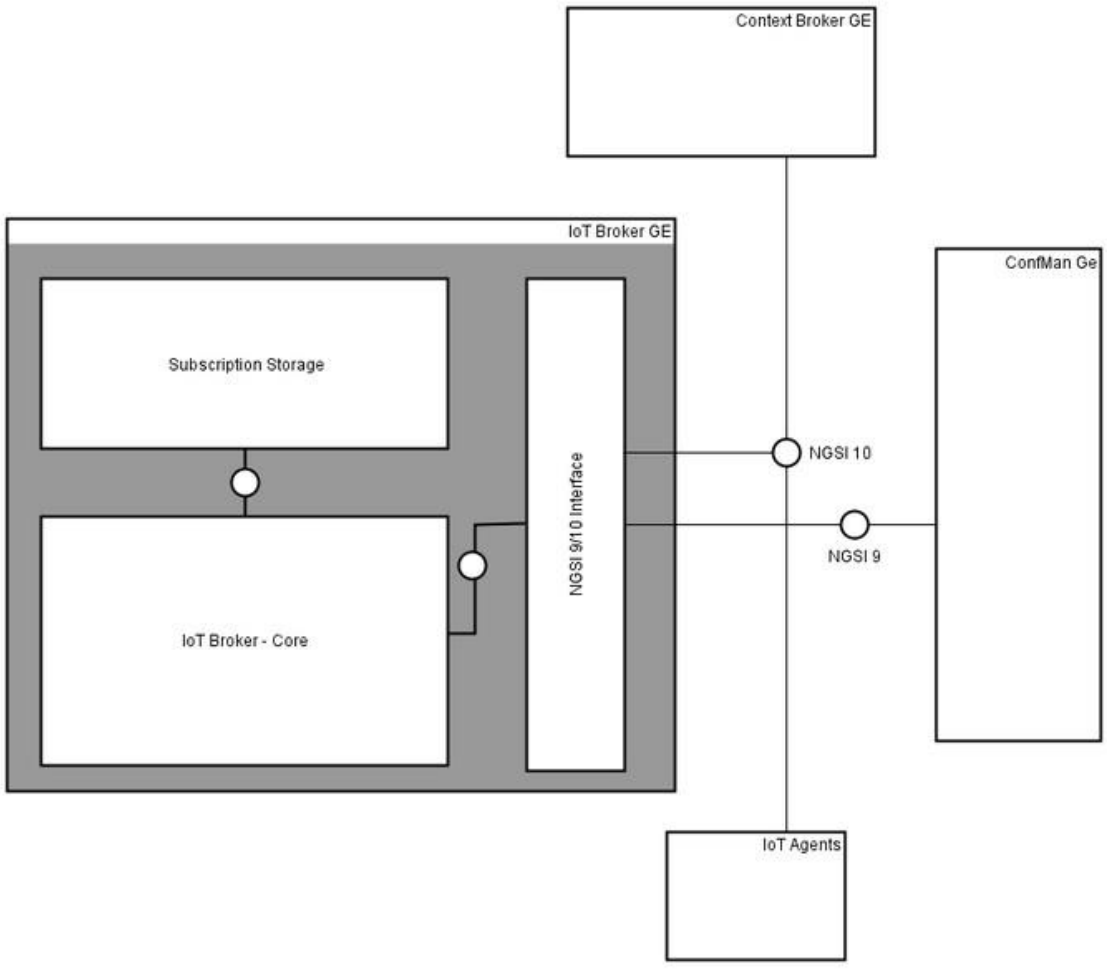
2.4 Overview

2.4.1 Data model outline

The IoT Broker GE is a component for retrieving and aggregating information from the Internet of Things. The underlying data model of this GE is based on the OMA NGSI (Next Generation Service Interface) Context Management Information Model, which is

centered on the concept of context entities. Context entities represent arbitrary objects of the real world, and the state of such objects is described in terms of the values of attributes. In addition, metadata can be associated to attribute values. In the context of IoT, context entities are used for representing devices like sensors and the values they measure, but also - and more importantly - arbitrary physical objects (Things) like rooms, persons, etc. and their attributes like temperature, geo-location, etc.

The OMA NGSI context management interface distinguishes two types of information. The first type is called *context information* and consists of attribute values and associated metadata, as described above. This kind of information is exchanged using the operations defined in OMA NGSI 10. The second type of information is *context availability information*, i.e. information on where context information can be retrieved by OMA NGSI 10 operations. This kind of information is exchanged using the operations defined in OMA NGSI 9. More details and references on these interfaces can be found below.



2.4.2 Functionality outline

The IoT Broker GE is the point of contact for accessing information about things and their attributes (see data model above). Applications can access this information using the NGSI 10 interface of the IoT Broker. In the overall FI-WARE reference architecture [7], the Context Broker GE from the Data & Context Chapter enables up-to-date access

to Context Information (not only things) by any application. Such Context Information can be of any nature and comprises, but is not limited to, the Context Information about things that is made available by the IoT Broker GE. However, the IoT Broker GE can as well be accessed by any number of applications directly.

The IoT Broker is a stateless component in the sense that it neither stores context information (i.e. attribute values) nor context availability information. Instead, it interacts with the whole IoT deployment in order to satisfy the requests it receives from the Context Broker GE or final applications: When receiving a query on Context Information, e.g. attributes of Context entities, the IoT Broker GE first contacts the ConfMan GE to perform a discovery. Using the resulting availability information, the IoT Broker then contacts the information providers using NGSI 10 to receive the actual attribute values. Note that the number of information providers to be contacted can be arbitrary. The information is then aggregated and returned to the application.

The NGSI context management interface describes three types of operations for exchanging context information. The first and most basic kind of operation is a simple query. When an application invokes the query, it expects to receive context information as the response. The second kind of operation is a subscription. When an application subscribes to certain context information, it just receives a subscription ID as the response. Context information is then sent to the application in the form of notifications. Depending on the kind of subscription, context information can be sent whenever attribute values change, whenever attribute values exceed or are inside some range, or simply at fixed time intervals.

Finally, there is a mode of information exchange defined in the NGSI context interface where information updates are sent without the need to ask for them. When the IoT Broker receives such updates, it forwards the information to some default application that is responsible for further processing and/or storing such updates. In the FI-WARE reference architecture this application is the Context Broker GE.

2.5 Main Concepts

2.5.1 Basic Concepts

The IoT Broker GE is based on the OMA NGSI context data model [1].

2.5.2 FI-WARE NGSI

The IoT Broker GE implements FI-WARE's RESTful binding specification of the OMA NGSI context management interface (FI-WARE NGSI specifications for short). More specifically, the GE implements the full set of NGSI-10 operations, where it acts both as a client and as a server. As for NGSI-9, the IoT Broker GE acts as a client and implements only the server function required for receiving context availability notifications.

FI-WARE NGSI Context Management specifications are based on the NGSI Context Management specifications defined by OMA (Open Mobile Alliance) [2]. 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 [3] and NGSI-10 [4]. 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 Open API Specification [5] to learn the main concepts underlying FI-WARE NGSI Context Management specifications. Please also refer to the e-learning course [9] on the IoT Broker Generic Enabler.

2.5.3 Additional Concepts

2.5.3.1 *Associations in FI-WARE NGSI*

One of the central features of the FI-WARE IoT Backend is its ability to derive information about arbitrary things (cars, people, houses, etc.) from device level information. The link between these two levels of abstraction is provided by the association concept defined for FI-WARE NGSI. A definition of the concept and its representation in NGSI-9 messages can be found on the NGSI association definition page [6]. An association is an ordered pair consisting of a sensor-level entity (possibly with an attribute name) and a thing-level entity (possibly enhanced with an attribute name). The semantics of such a pair defines how the sensor provides information about the thing.

Associations potentially play a role for every operation supported by the IoT Broker.

- The IoT Broker GE queries the ConfMan GE not only for context providers, but also for associations.
- For resolving a *queryContext* operation, the IoT Broker uses the association concept to find out which lower-level entities, typically IoT resources, it has to ask the context providers for. For example, for finding out the speed of a certain car, the IoT Broker might ask for the measurement of a speedometer.
- For the translation of thing-level subscriptions into device-level subscriptions the association concept is used in the same way as in the query case.
- When context providers call the update operation of the IoT Broker GE, the latter analyzes associations in order to find out if updates of attribute values of Thing-level entities have to be triggered.

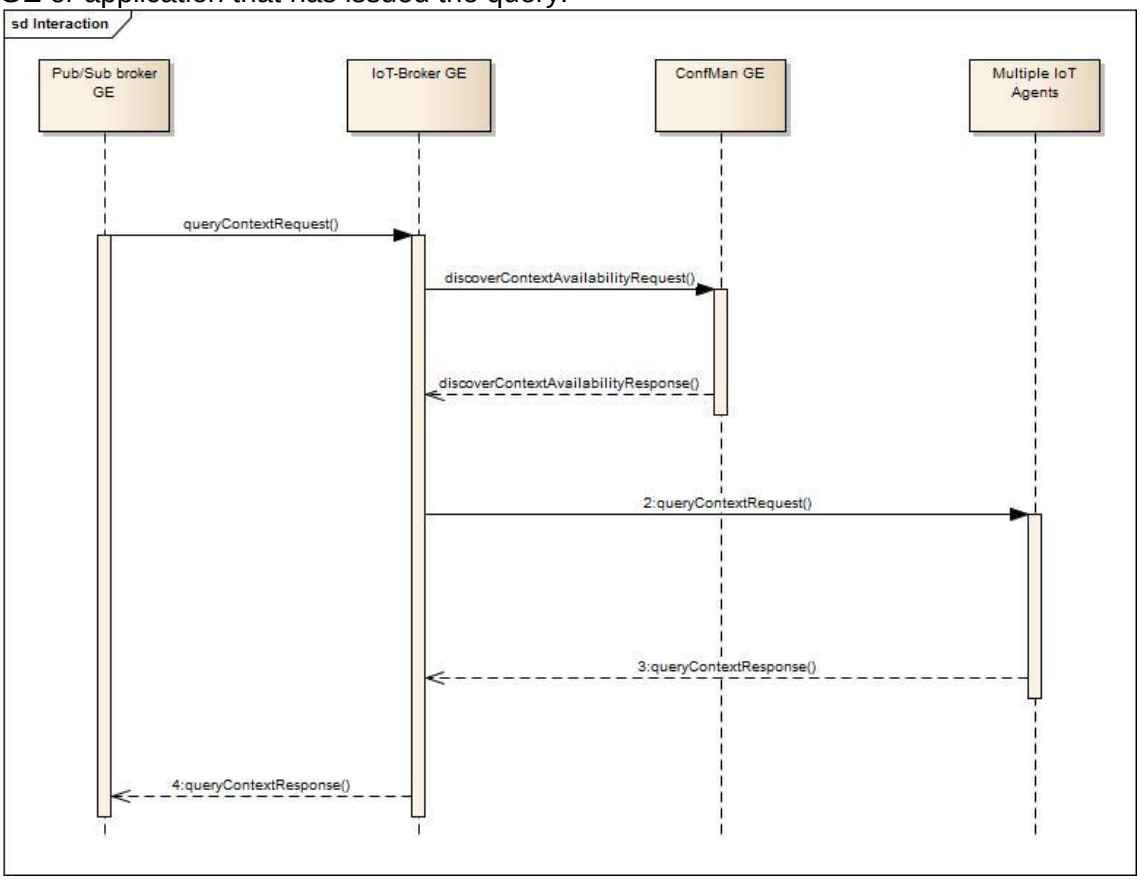
2.6 Main Interactions

The IoT Broker Generic Enabler is a middleware used for setting up and maintaining the data flows in IoT deployments. It is designed to interact with large numbers of IoT data providers and data consumers. On behalf of the consumers, the IoT Broker retrieves, assembles, and processes information from the providers, offering the consumers a simple interface and masking the complexity and heterogeneity of the Internet of Things. For this reason, the IoT Broker GE interacts potentially with a large number of Gateways, other Backend instances, Devices, and of course data consumers. Furthermore, the IoT Broker GE needs to interact with at least one instance of the Configuration Management GE. The latter is where the IoT Broker GE retrieves information about where information is available in the IoT installation.

In the FI-WARE architecture, the role of the data consumer is played by a Generic Enabler in the Data & Context Chapter: the Context Broker. The IoT Broker GE communicates with the Context Broker GE via the Northbound Interface. The Southbound interface is used to communicate with the so-called IoT Agents, which are providing data. The role of IoT Agent can be played by either the Backend Device Management GE, or by the Gateway Data Handling GE.

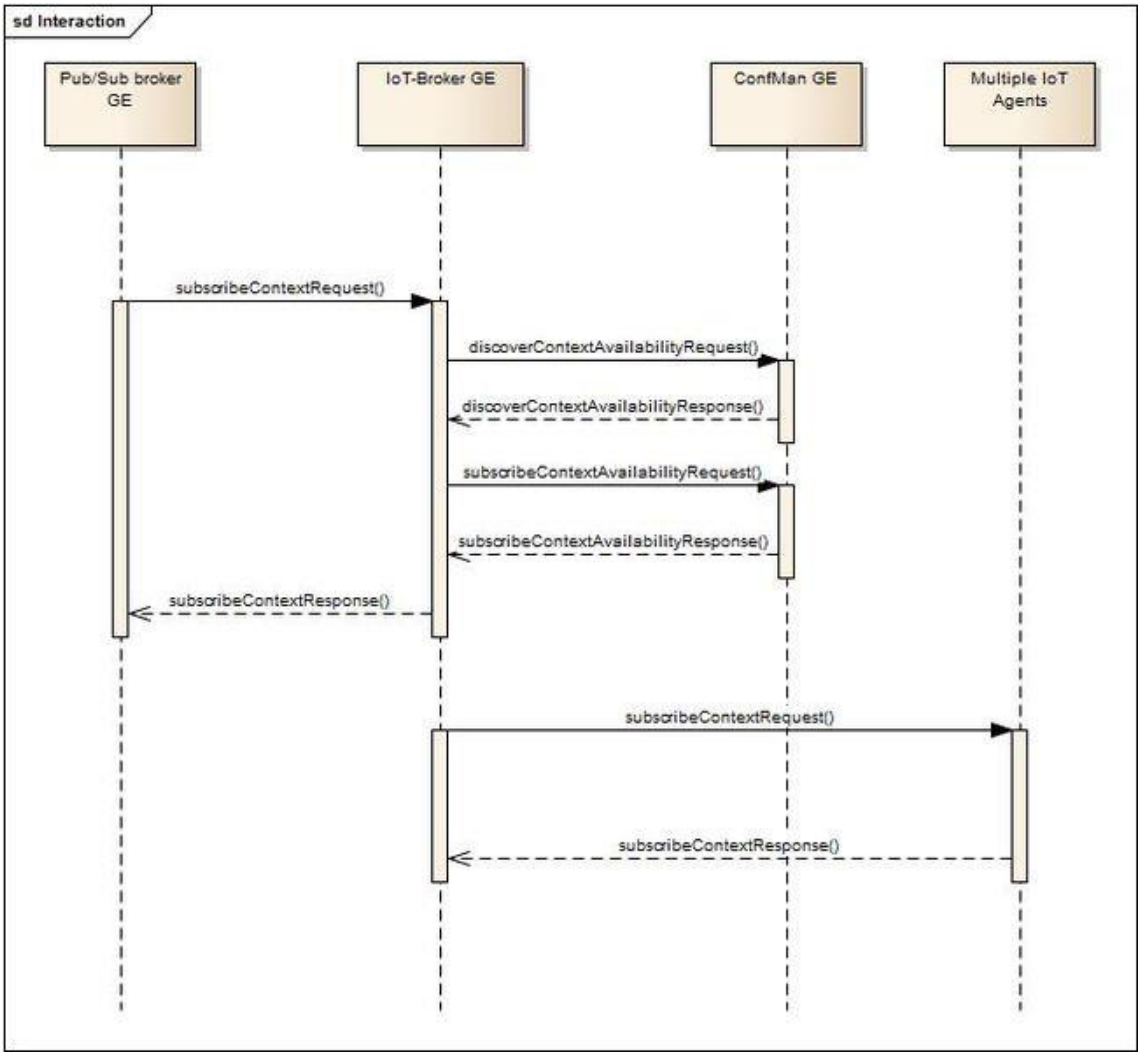
2.6.1 Query Handling

Queries are one-time requests for information. They are realized by the *queryContext* operation of OMA NGSI-10. With NGSI-10, GEs or applications can query for Thing-level information. In reaction to a query, the IoT Broker determines the set of IoT Agents that can provide the requested information. This will be done by sending a *discoverContextAvailability* request to the ConfMan GE, which returns relevant associations between Thing-level entities/attributes and Device-level entities/attributes, and provides addresses of the IoT Agents who can provide the required information. After this step, the IoT Broker GE queries the identified IoT agents, it aggregates the results, it maps Device-level entities to Thing-level entities and it forwards them to the GE or application that has issued the query.



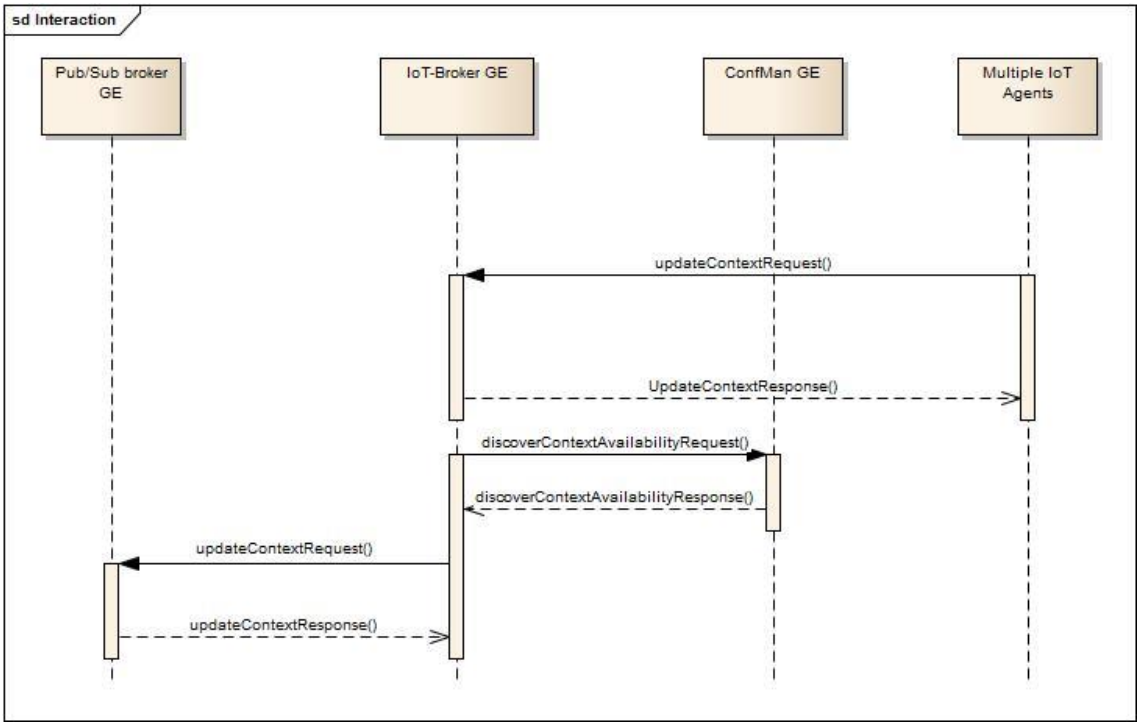
2.6.2 Subscription Handling

Subscriptions are requests for information updates the issuer wishes to receive, under certain conditions 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 also applies to the *updateContextSubscription* and to the *unsubscribeContext* operations. Also in these interactions, as in the Query Handling, the role of the ConfMan GE is to provide the relevant associations between the Device-level and Thing-level entities and the addresses of the relevant information sources (i.e. the IoT agents).



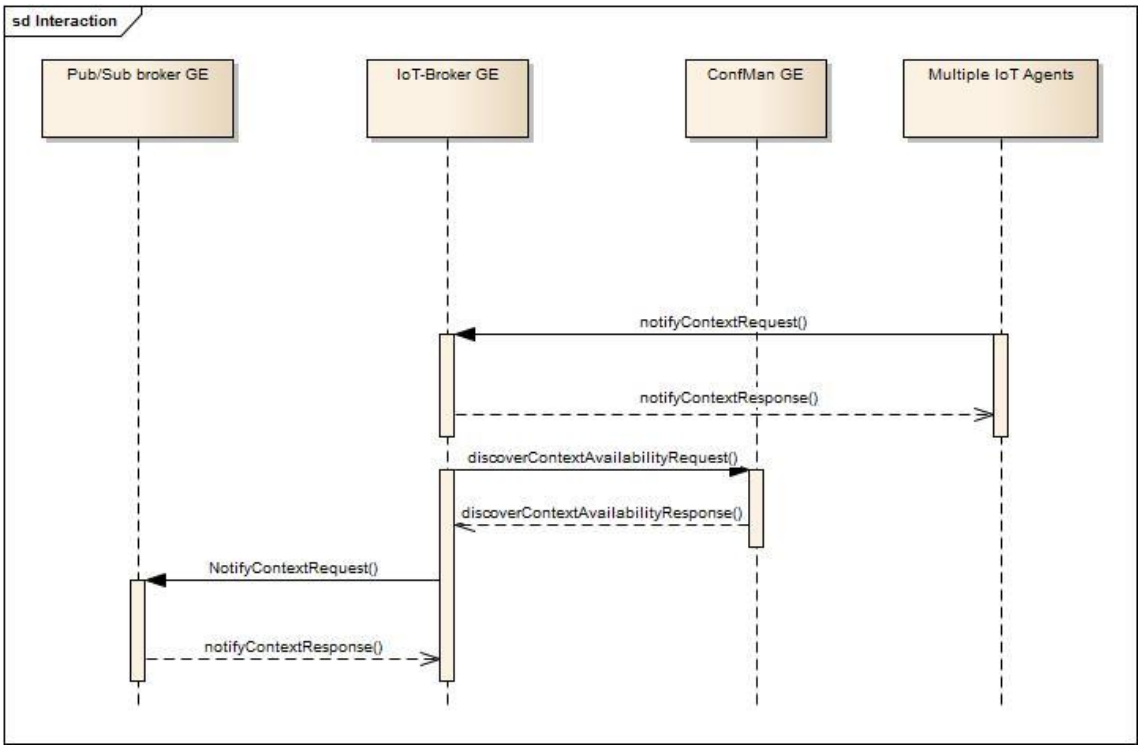
2.6.3 Update

In general, updates received by the IoT Broker GE are forwarded to the Context Broker GE. Before forwarding, the IoT Broker discovers the Thing-level entities associated to the Device-level entities about which it has received an update. For that reason the IoT Broker contacts the ConfMan GE using a *discoverContextAvailability* request. Note that a single received update may translate into several updates because attributes of several things may need to be updated as a result of an update of a device-level entity (e.g., the temperature measured by a given thermometer may lead to updates on a thing representing a room close to the thermometer, as well as updates on things representing the floor and the building where the room is located).



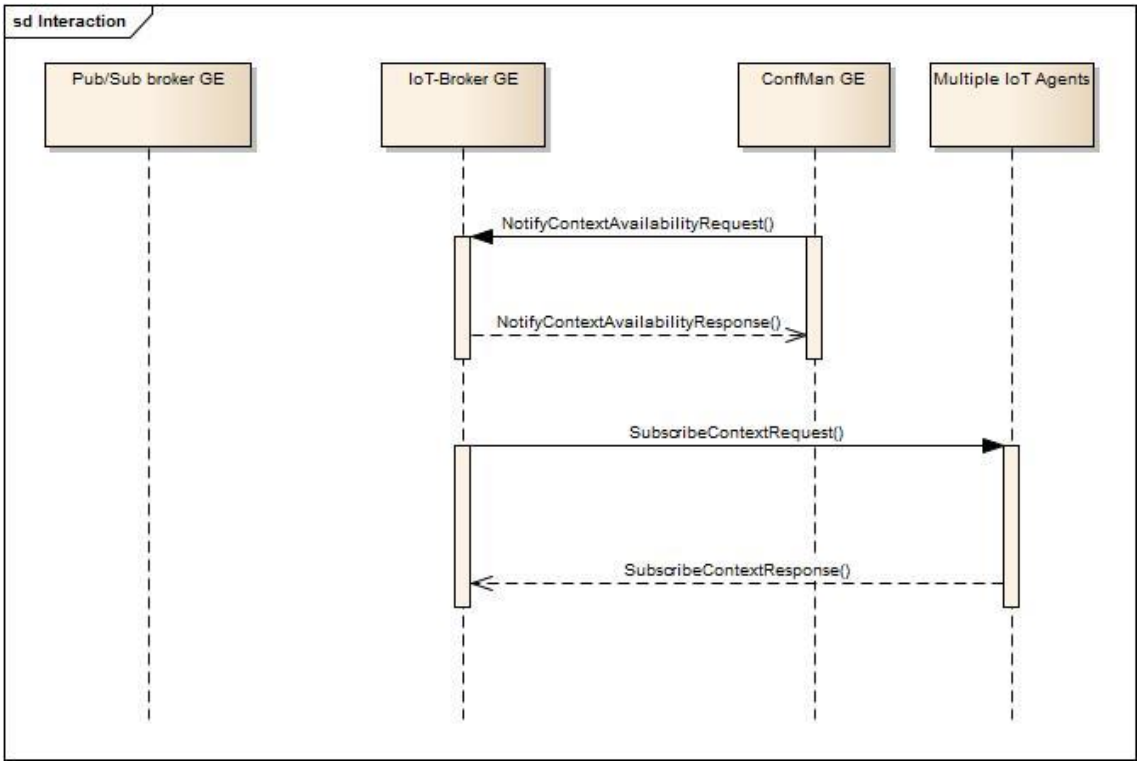
2.6.4 Notification

Notifications are the counterpart of subscriptions. A notification is sent whenever the condition that has been specified in the subscription is satisfied. Like in the update operation, also here the IoT Broker needs to resolve the relevant associations.



2.6.5 Availability Notification

When a new IoT Agent having information that is relevant for an existing subscription becomes available, the IoT Broker is notified about this in an availability notification, so that it can then make a subscription to the new IoT Agent.



2.7 Basic Design Principles

- **Stateless Design:** The IoT Broker GE is stateless in the sense that no context information is stored by it. Its role in the Internet of Things is not to serve as a central data repository, but as a central point which retrieves and aggregates data from various sources on behalf of applications. The absence of states enables easy replication of IoT Broker GE instances, or federated deployments where multiple instances of the IoT Broker GE are responsible for different parts or domains of the Internet of Things. In such cases each IoT Broker GE instance serves as an IoT Agent for the other instances, and each IoT Broker GE can in principle retrieve any piece of information available in the IoT deployment.
- **Silent System:** The IoT Broker enables IoT systems whose level of activity is proportional to the number and complexity of the requests by applications. This is due to the fact that the IoT Broker only queries IoT Agents when there are requests from applications. The principle of silent systems is particularly helpful when working with battery constrained devices. However, also systems that continuously produce data can be built using the IoT Broker GE. In this case, IoT Agents have to be configured to use the NGSI 10 update operation in order to deliver data independently of application requests.

2.8 References

2.9 Detailed 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.

2.9.1 Open API Specifications

The single interface exposed by the IoT Broker GE is FI-WARE NGSI 10. This interface is used by applications (or, in the FI-WARE reference architecture, the Publish/Subscribe Broker GE) that want to receive information from the IoT, and also from Gateways and Devices to send information towards the IoT Broker. Applications invoke the operations for context query and context subscription, while IoT Gateways use the operations for context notification and context update.

2.9.2 Other Open Specifications

N/A

2.10 Re-utilised Technologies/Specifications

The interfaces of the IoT Broker GE are based on RESTful Design Principles. The technologies and specifications used in this GE are:

- RESTful web services
- HTTP/1.1 (RFC2616)
- XML data serialization format.

2.11 Terms and definitions

This section comprises a summary of terms and definitions introduced during the previous sections. It intends to establish a vocabulary that will be help to carry out discussions internally and with third parties (e.g., Use Case projects in the EU FP7 Future Internet PPP). For a summary of terms and definitions managed at overall FI-WARE level, please refer to FIWARE Global Terms and Definitions

[1] https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/NGSI_9/10_information_model

[2] http://technical.openmobilealliance.org/Technical/release_program/docs/NGSI/V1_0-20120529-A/OMA-TS-NGSI_Context_Management-V1_0-20120529-A.pdf

[3] https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI-9_Open_RESTful_API_Specification_%28PRELIMINARY%29

[4] https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI-

10_Open_RESTful_API_Specification_%28PRELIMINARY%29

[5] https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI_Open_RESTful_API_Specification

[6] https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/NGSI_association

[7] https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_Architecture

[8] https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_Open_Specification_Legal_Notice_%28implicit_patents_license%29

[9] http://edu.fi-ware.eu/course/view.php?id=33

- **Thing**. One instance of a physical object, living organism, person or concept interesting to a person or an application from the perspective of a FI-WARE instance, one particular usage area or to several usage areas. Examples of physical objects are: building, table, bridge (classes), the Kreml, the tennis table from John's garden, the Tower bridge (instances). Examples of living organisms are: frog, tree, person (classes), the green frog from the garden, the oak tree in front of this building, Dr. Green.
- **Class of thing**. Defines the type of a thing in the style of object-oriented programming: a construct that is used as a blueprint to create instances, defining constituent members which enable class instances to have state and behavior. An example of class of thing is "person", whereas an instance of a thing is "Dr. Green", with all the static/dynamically changing properties of this particular person.
- **Group of things**. A physical/logical group of things. Examples are all office buildings from Munich, all bridges above the Thames, all screws from a drawer, the planes of Lufthansa currently above Germany, all cars inside a traffic jam driven by a female driver, the trees from the Amazonian forest, the squids from the Mediterranean see the mushrooms from Alice's garden, all the fish from the aquarium of John., the soccer team of Manchester, the colleagues from OrangeLabs currently working *abroad* (from the perspective of France), all patients above 60 years of age of Dr. Green, the traffic jams from Budapest, the wheat crop in France in the year 2011, the Research and Development Department of the company Telefónica.
- **Device**. Hardware entity, component or system that may be in a relationship with a *thing* or a *group of things* called *association*. A device has the means either to measure properties of a thing/group of things and convert it to an analog or digital signal that can be read by a program or user or to potentially influence the properties of a thing/group of things or both to measure/influence. In case when it can only measure the properties, then we call it a sensor. In case when it can potentially influence the properties of a thing, we call it an actuator. Sensors and actuators may be elementary or composed from a set of elementary sensors/actuators. The simplest elementary sensor is a piece of hardware measuring a simple quantity, such as speed of the wind, and displaying it in a mechanical fashion. Sensors may be the combination of software and hardware components, such as a vehicle on-board unit, that consists of simple sensors measuring various quantities such as the speed of

the car, fuel consumption etc and a wireless module transmitting the measurement result to an application platform. The simplest elementary actuator is a light switch. We have emphasized *potentially* because as a result of the light switch being switched on it is not sure that the effect will be that light bulb goes on: only an *associated* sensor will be able to determine whether it went on or not. Other examples of devices are smart meters, mobile POS devices. More sophisticated devices may have a unique identifier, embedded computing capabilities and local data storage utilities, as well as embedded communication capabilities over short and/or long distances to communicate with IoT backend. Simple devices may not have all these capabilities and they can for instance only disclose the measurement results via mechanical means. In this latter case the further disclosure of the measurement result towards IoT backend requires another kind of device, called IoT Gateway.

- **Association**. It is a physical/logical relationship between a device and a thing or a device and a group of things or a group of devices and a group of things from the perspective of a FI-WARE instance, an application within a usage area project or other stakeholder. A device is associated with a thing if it can sense or (potentially) influence at least one property of the thing, property capturing an aspect of the thing interesting to a person or an application from the perspective of a FI-WARE instance, one particular usage area or to several usage areas. Devices are associated with things in *fully dynamic* or *mainly static* or *fully static* manner. The association may have several different embodiments: *physical attachment*, *physical embedding*, *physical neighborhood, logical relation* etc. Physical attachment means that the device is physically attached to the thing in order to monitor and interact with it, enabling the thing to be connected to the Internet. An example is the on-board device installed inside the vehicle to allow sending sensor data from the car to the FI-WARE instances. Physical embedding means that the device is deeply built inside the thing or almost part of the thing. An example of physical embedding is the GPS sensor inside a mobile phone. Physical neighborhood means that the device is in the physical neighborhood of the thing, but not in physical contact with it. An example of physical neighborhood is the association of the mobile phone of a subscriber who is caught in a traffic jam with the traffic jam itself: the mobile phone is the device, the traffic jam is the thing and the association means that the mobile phone is in the physical neighborhood of the area inside which the traffic jam is constrained (e.g. within 100 m from the region where the average speed of cars is below 5 km/h). Logical association means that there is a relationship between the device and the thing which is neither fully physical in the sense of attachment or embedding, nor fully physical proximity related. An example of logical association is between the car and the garage door opener of the garage where the car usually parks during the night.
- **IoT gateway**. A device that additionally to or instead of sensing/actuating provides inter-networking and protocol conversion functionalities between devices and IoT backend potentially in any combination of these hosts 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. 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 devices such as the available computing, power, storage and energy consumption. The level of functional split between the IoT backend and the IoT gateway will also depend on the available resources on the IoT gateway, the cost and quality of connectivity and the desired level for the distribution of intelligence and service abstraction.

- **IoT resource**. Computational elements (software) that provide the technical means to perform sensing and/or actuation on the device. There may be one-to-one or one-to-many relationship between a device and its IoT resource. Actuation capabilities exposed by the IoT resource may comprise configuration of the management/application features of the device, such as connectivity, access control, information, while sensing may comprise the gathering of faults, performance metrics, accounting/administration data from the device, as well as application data about the properties of the thing with which the device is associated. The resource is usually hosted on the device.
- **Management service**. It is the feature of the IoT resource providing programmatic access to readable and/or writable data belonging to the functioning of the device, comprising a subset of the FCAPS categories, that is, configuration management, fault management, accounting /access management/administration, performance management/provisioning and security management. The extent of the subset depends on the usage area. Example of configuration management data is the IP endpoint of the IoT backend instance to which the device communicates. Example of fault management is an error given as a result of the overheating of the device because of extensive exposure to the sun. Example of accounting/administration is setting the link quota for applications using data from a certain sensor. Example of security management is the provisioning of a list of device ID-s of peer devices with which the device may directly communicate.
- **Application service**. It is the feature of the IoT resource providing programmatic access to readable or writable data in connection with the thing which is associated with the device hosting the resource. The application service exchanges application data with another device (including IoT gateway) and/or the IoT backend. Measured sensory data are example of application data flowing from devices to sensors. Setting the steering angle of a security camera or sending a “start wetting” command to the irrigation system are examples of application data flowing from the application towards the sensor.
- **Event**. An event can be defined as an activity that happens, occurs in a device, gateway, IoT backend or is created by a software component inside the IoT service enablement. The digital representation of this activity in a device, IoT resource, IoT gateway or IoT backend instance, or more generally, in a FI-WARE instance, is also called an event. Events may be simple and complex. A simple event is an event that is not an abstraction or composition of other events. An example of a simple event is that “smart meter X got broken”. A complex event is an abstraction of other events called member events. For example a stock market crash or a cellular network blackout is an abstraction denoting many thousand of member events. In many cases a complex event references the set of its members, the implication being that the event contains a reference. For example, the cellular network blackout may be caused by a power failure of the air conditioning in the operator’s data center. In other cases such a reference may not exist. For example there is no accepted agreement as to which events are members of a stock market crash complex event. Complex events may be created of simple events or other complex events by an IoT resource or the IoT backend instance.
- **IoT Backend.** Provides management functionalities for the devices and IoT domain-specific support for the applications. Integrant component of FI-WARE instances.
- **IoT application**. An application that uses the application programming interface of the IoT service enablement component. May have parts running on one/set of devices, one/set of IoT gateways and one/set of IoT backends.

- **Virtual thing**. It is the digital representation of a thing inside the IoT service enablement component. Consists of a set of properties which are interesting to a person or an application from the perspective of a FI-WARE instance, one particular usage area or to several usage areas. Classes of things have the same set of properties, only the value of the properties change from instance to instance.

3 FIWARE.OpenSpecification.IoT.Backend.ConfMan

Name	FIWARE.OpenSpecification.IoT.Backend.ConfMan		
Chapter	IoT Services Enablement,		
Catalogue-Link to Implementation	Conf Manager	**Owner**	Telefonica I+D, Fermín Galán

3.1 Preface

Within this document you find a self-contained open specification of a FI-WARE generic enabler, please consult as well the FI-WARE Product Vision, the website on http://www.fi-ware.org and similar pages in order to understand the complete context of the FI-WARE project.

3.2 Copyright

3.3 Legal Notice

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

3.4 Overview

3.4.1 Data model outline

The Configuration Manager GE (ConfMan GE for short) is the part of the IoT Backend which is responsible for the registration of the *context availability* information. The underlying data model of this GE is based on the OMA NGSI9 Context Management Information Model. This model relies on the concept of context entities, which are generic entities whose state is described by the means of 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. Additionally - and more importantly - arbitrary physical objects (Things) like rooms, people, etc. and their attributes like temperature, geo-location, etc. can be used as well.

The ConfMan GE also optionally supports IoT descriptions that are semantically-annotated. The models chosen to represent the IoT Concepts are based on the IoT-A

ontologies. They mainly describe an IoT Resource, a Virtual Entity and an IoT Service. When mapping to FIWARE IoT Concepts:

- the IoT-A IoT Resource would map to the FIWARE IoT Resource
- the IoT-A Virtual Entity would map to the FIWARE IoT Thing
- the IoT-A Service would map to the "providing application" endpoint where data can be retrieved from a FIWARE IoT Resource or Thing.

3.4.2 Functionality outline

The ConfMan GE is in charge of context availability registrations from IoT Agents, thus acting as the access point for information about entities and their attributes. In particular, the context availability information provided by the ConfMan GE is forwarded to IoT Agents exporting the FI-WARE NGSI-9 interface. IoT Agents can be either the Data Handling GE in IoT Gateways, or the Backend Device Management GE. Moreover, it is also allowed to provide a context information also by other IoT Backend systems.

Context availability information is typically forwarded to the FI-WARE Context Broker GE so that context information about Things becomes accessible to applications. However, note that the Context Broker GE may manage context availability information not necessarily provided by the ConfMan GE, therefore linked to the Internet of Things, but gathered from other parts of the application.

More precisely, using the FI-WARE NGSI-9 interface that the ConfMan 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 ConfMan GE has been designed to optionally support the registration, management and discovery of semantically-annotated IoT descriptions that are based on the IoT-A ontologies. The GE provides probabilistic and semantic search mechanisms for the discovery of IoT Descriptions that are stored in the local Configuration repository.

3.5 Basic Concepts

The ConfMan GE is based on the OMA NGSI context data model.

3.5.1 FI-WARE NGSI

The ConfMan GE implements the RESTful FI-WARE NGSI binding specification. FI-WARE NGSI Context Management specifications are based on the NGSI Context Management specifications defined by OMA (Open Mobile Alliance). They include two RESTful context management interfaces NGSI-9 and NGSI-10 (see Open RESTful API Specification), which 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.

3.5.2 ConfMan GE Architecture

This section describes the internal architecture of the FI-WARE ConfMan GE implementation.

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 NGSI-9 interface exposed by the Configuration Management.

- Announce the availability of context information towards the Context 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 subsection. This repository is used to answer NGSI-9 request in the exposed interfaces described above.

- Receive registrations from IoT Gateways and the Thing-level Adapter through the NGSI-9 southbound interface and store this information in the Configuration Repository, thus updating the context information registry.

3.5.2.1 ***Configuration Repository***

The Configuration Repository stores information on the availability of context information and can be accessed through the Configuration Management. When a NGSI9 client send a *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.

With this approach, the ConfMan GE can maintain more abstract-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 specific sensors provide information about same car could be maintained only by the ConfMan GE.

3.5.2.2 ***Optional Features***

There are some optional features which can provide more added-value and that could be included in the Generic Enabler ConfMan:

- Geo-discovery supporting the usage of geographic scopes in the operations "*discoverContextAvailability*" and "*subscribeContextAvailability*"
- Semantic and Probabilistic Discovery, using semantically-annotated descriptions and providing a set of search mechanisms for look-up and discovery

3.6 Additional Concepts

3.6.1 Associations in FI-WARE NGSI-9

ConfMan GE enables enhanced associations. See the following section in the IoT Broker GE Architecture page to have more information on this.

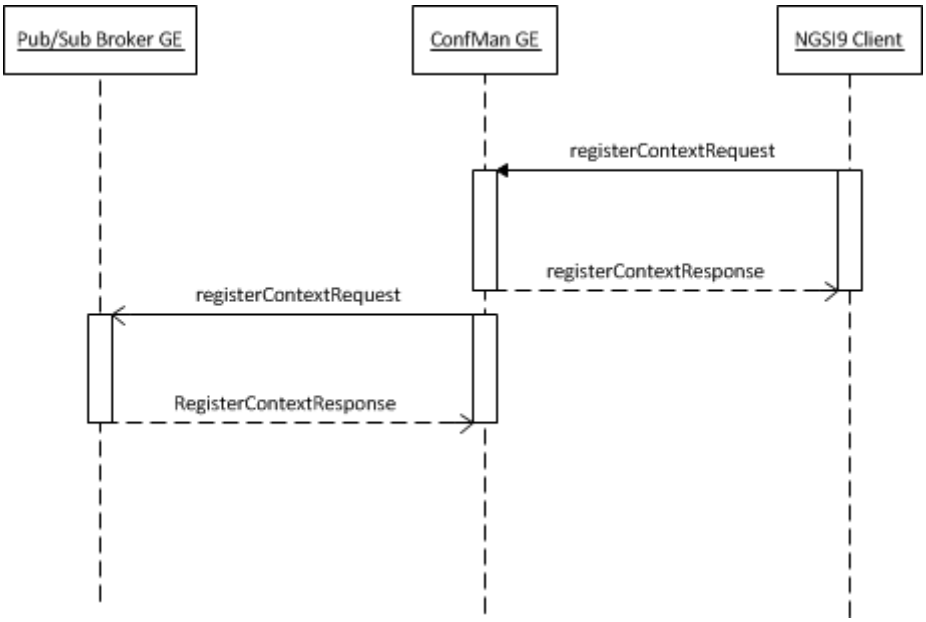
3.7 Main Interactions

The ConfMan 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 Context 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. There is also an interface between the IoT Broker GE and the ConfMan GE. In the following diagrams we abstract all these components using "NGSI9 Client".

3.7.1 Reception of RegisterContext operations from Agents

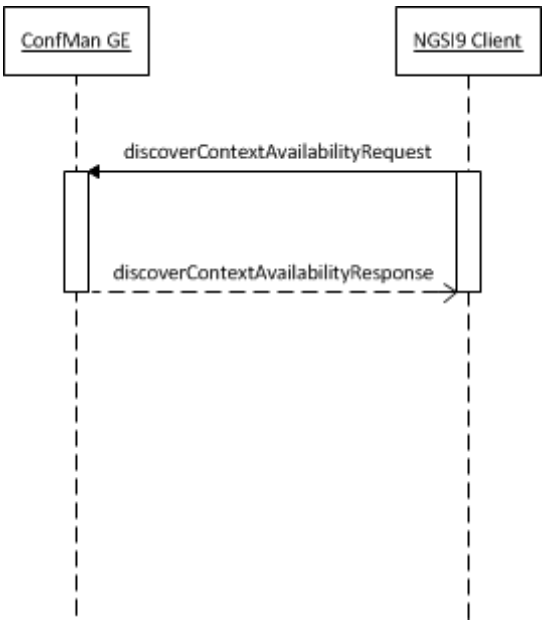
In order for their information on Things and/or Resources to be available at the Backend, IoT Agents (which play the role of "NGSI9 Client9") need to register their information to the ConfMan GE. This is done via the *RegisterContext* operation. Note that the registration can be performed at various 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). Also note that in absence of a Context Broker GE instance the registration is still stored in the ConfMan GE.

This sequence is used for both new registrations and registrations updates.



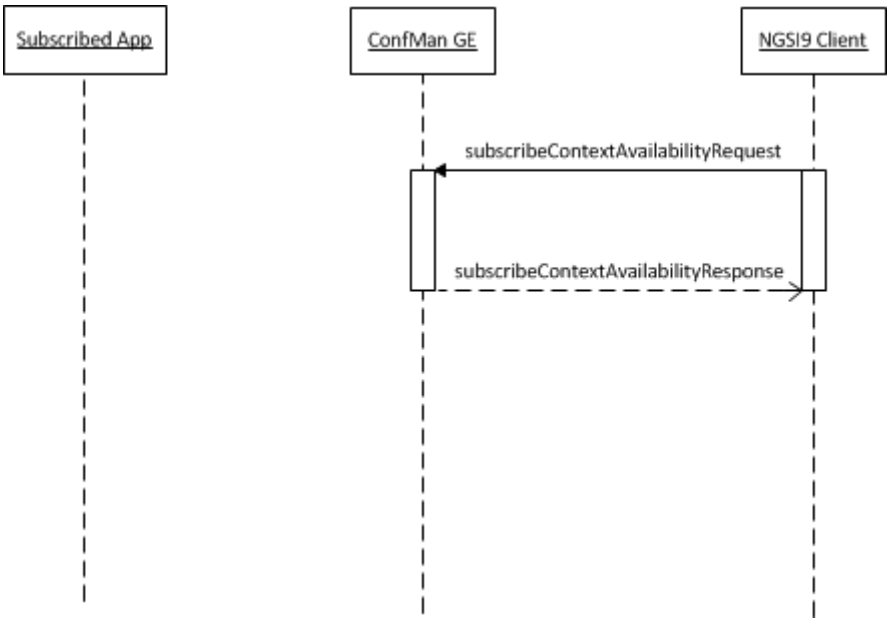
3.7.2 Query Discover Availability

The entities that typically play the "NGSI9 Client" role in this case are the IoT Broker GE (when requesting context availability information to process NGSI10 query/updates) or any other NGSI application that wants to know the context availability associated with a given entity/attribute.



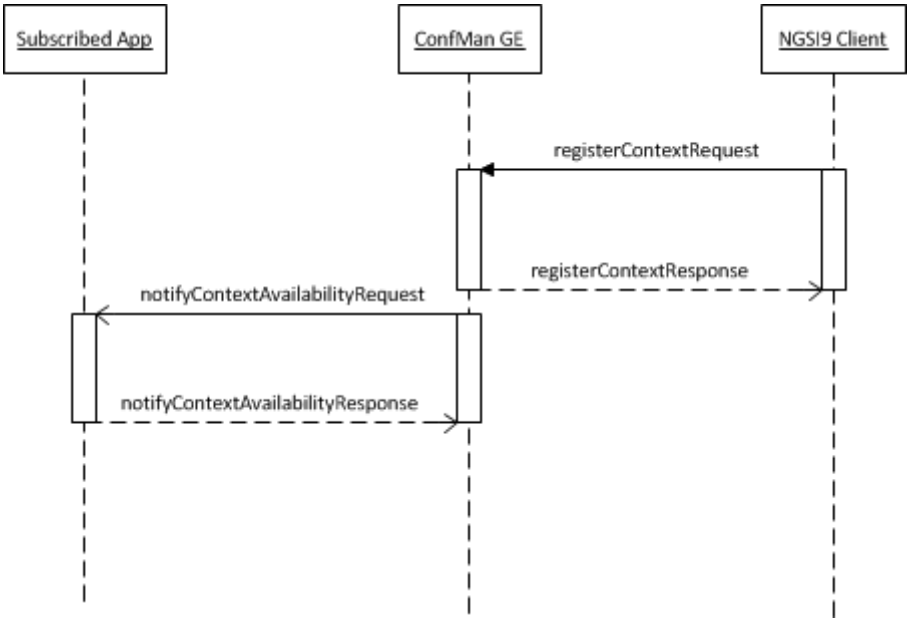
3.7.3 Subscription to Context Availability

The entities that potentially could play the "NGSI9 Client" role are any application that needs to be aware of changes in context availability information. The "Subscribed App" could be the "NGSI9 Client itself".



3.7.4 Notification

The entities that potentially could play the "NGSI9 Client" role are IoT Agents, while "Subscribed App" are applications subscribed to context availability information changed by the registration issued by the "NGSI0 Client" (we assume that such Subscribed Application are previously subscribed, as shown in the sequence diagram in previous section).



3.8 Added-value (Optional) Features

The following sections describe additional features that a Configuration Manager may implement. However, they are not mandatory for a basic implementation of this GE.

3.8.1 Geo-discovery

As an optional feature, the Configuration Management GE supports the usage of geographic scopes in the operations *discoverContextAvailability* and *subscribeContextAvailability*. In particular, the geographic scopes defined under keyword SimpleGeoLocation in Appendix D of the OMA NGSI 9/10 specification[1] will be supported.

For specifying the geographic location of entities and/or context providers, a new type of registration metadata will be defined and implemented. The exact syntax and semantics of this metadata type is under discussion.

Reference [1]

3.8.2 Probabilistic and Semantic Discovery

3.8.2.1 ***Overview***

This additional functionality is actually an optional component within the Configuration Manager GE targeted for:

- Context Producers using semantic descriptions based on the IoT-A ontology models for IoT Description registration and management.
    - IoT Agents
    - Device Gateways
- Context Consumers using SPARQL or FIWARE NGSI9 for discovering IoT Concepts.
    - Applications requiring Semantic Discovery of IoT Concepts.
    - Semantically-enabled GEs (Data/Context Chapter)

N.B: By “Context”, we refer to descriptions about IoT Concepts, which contains its Identification, Attributes and Metadata about specificities concerning its Attributes.

This component, which has been materialized in the IoT Discovery GE implementation, addresses the discovery of Internet of Things (IoT) Concepts, by providing a repository for IoT Context Producers to register their IoT Things, Resources, and Devices, using semantically-annotated Descriptions based on the IoT-A (Internet of Things Architecture) ontology models. In turn, it provides a set of search mechanisms for their look-up and discovery. One of the main goals of this GE is to make use of semantic annotation in order to apply formal naming and relational conventions to the description of an IoT Concept, which is explicitly absent in NGSI-9/10.

The component makes use of the Sense2Web IoT Linked Data Platform baseline asset, which provides a repository for the CRUD (Create, Read, Update and Delete) management of Semantic IoT descriptions, that complies with the IoT-A ontology models. These descriptions are termed as the Advanced IoT Descriptions. Sense2Web also associates different IoT concept ontologies to domain data and other resources on the Web, using Linked Open Data.

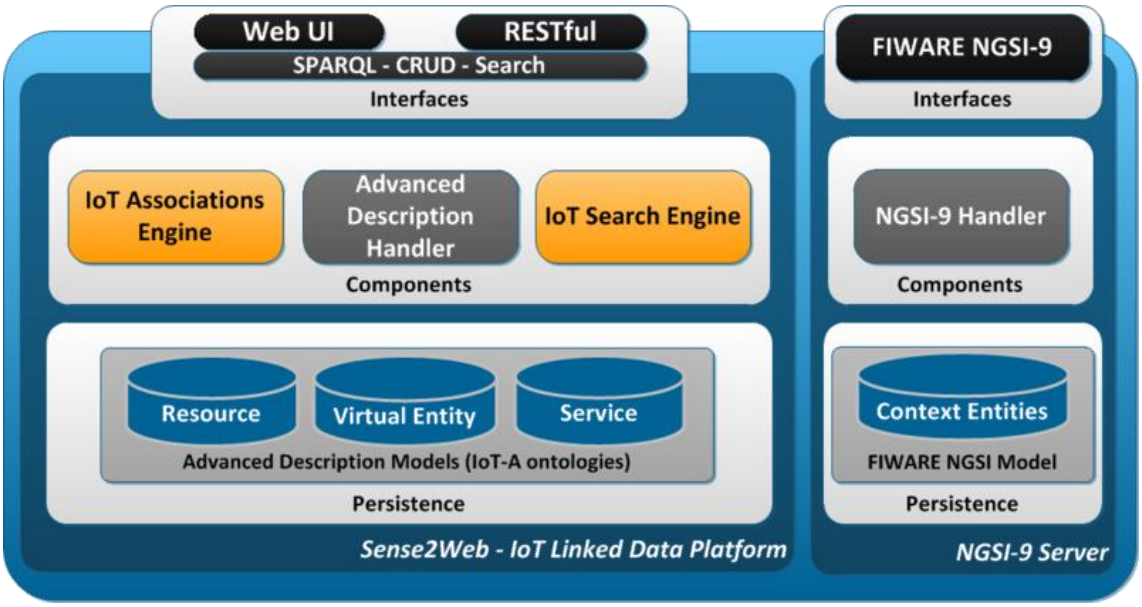


Figure 5: Sense2Web Platform and NGSI-9 Server

3.8.2.2 *New Interfaces Exposed*

This additional component provides a set of interfaces a user can interact with. The first is a Web User Interface (UI) whereby a user can perform CRUD (Create, Read, Update and Delete) operations on the IoT Descriptions, and to query the IoT Descriptions as well. When registering or updating, a user can either upload an IoT Description or complete a form which is then sent to the server to be converted to RDF format, and storing it in the RDF database. The second interface is a RESTful CRUD and SPARQL interface. This interface mainly supports M2M interactions. An application can also perform CRUD operations on the IoT descriptions in the repository, and query for a particular piece of information from the descriptions using SPARQL. The third interface allows users to query about an IoT description using keywords or templates that share the same structure as the IoT description. This type of query input is handled by the IoT Search Engine, which will search for the relevant query.

3.8.2.3 *Main Concepts / Subcomponents*

The main subcomponents for this component are the **IoT Search Engine**, which is a probabilistic search mechanism that is based on text analysis for indexing and searching, and the **IoT Associations Engine** which is a semantic search mechanism that is used to establish and maintain associations between IoT Concepts.

3.8.2.4 *IoT Search Engine (Probabilistic)*

3.8.2.4.1 Introduction

The IoT Search Engine is a component in the Configuration Management GE that uses machine learning (ML) techniques for resolving IoT Descriptions (whether XML or RDF). It uses unsupervised probabilistic ML techniques to group similar IoT Descriptions. Its purpose is to allow automated discovery, ranking and recommendation of IoT Descriptions. It provides an efficient mechanism for the search and discovery of registered IoT Descriptions, by automatically clustering all the IoT Descriptions and hence allowing the IoT Search Engine to be scalable to large object repositories. It also automates the ranking of results in order of relevance.

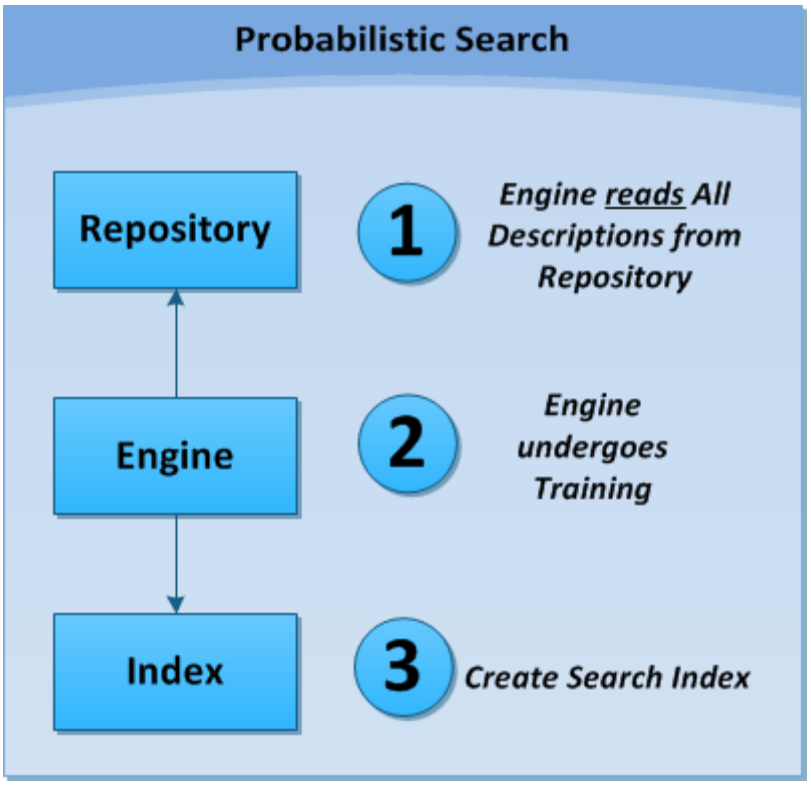


Figure 6: Probabilistic Search Overview

(2)Training the Engine for Index Creation and Clustering

The IoT Search Engine is able to efficiently refine the searching process, going through several stages of operation. First stage involves a training process whereby all the Descriptions for each model type are retrieved from the repository. The search engine will then extract "textual concepts" from the Descriptions using a parsing mechanism. The concepts are essentially the elements, attributes and values of a Description. The extracted textual concepts will then be used to create a "Document-term matrix", which is a mathematical matrix that is used to reflect the frequency of textual concepts in a particular IoT Description, whereby the rows represent the number of IoT Descriptions and the columns represent the numbers of concepts.

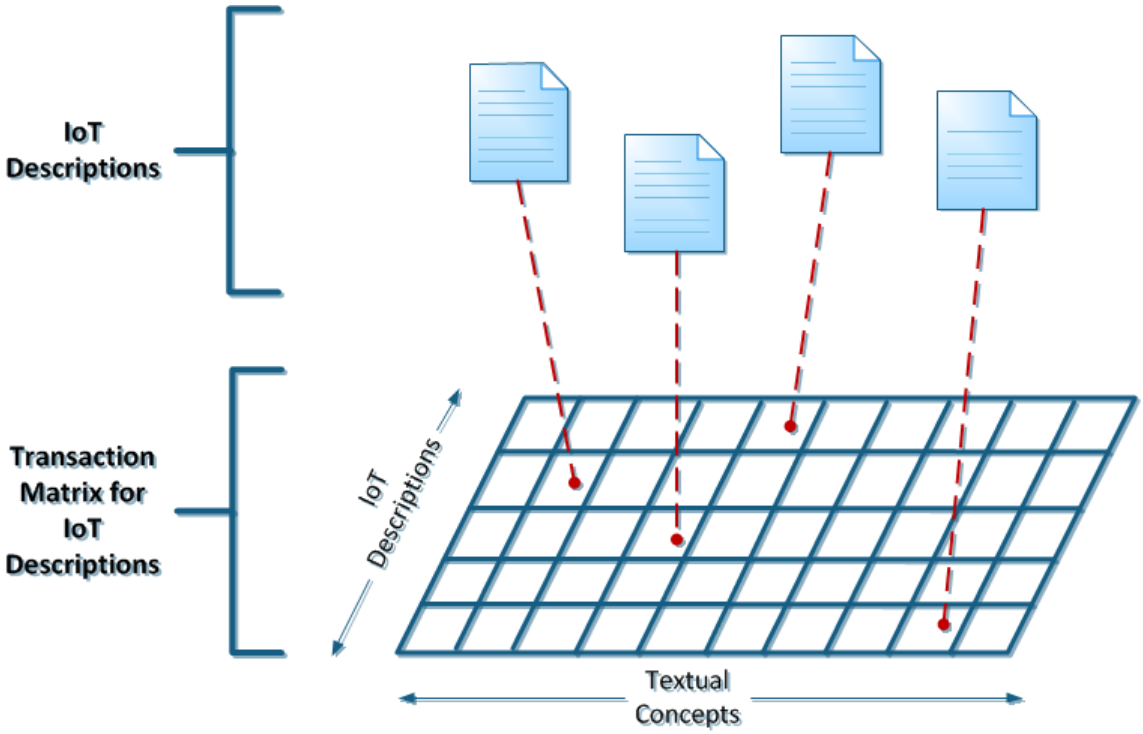


Figure 7: Concept Extraction

The next stage is another training process for clustering the IoT descriptions. The training involves a ML technique known as the Latent Dirichlet Allocation. The aim of this technique is to discover a hidden dimension behind the vector of concepts. This dimension essentially reflects a "topic" or "theme", which is shared by certain IoT Descriptions. The topic itself is modeled as Latent Factors, which allows any type of Descriptions to be represented in vector form, and hence making the search much more efficient compared to keeping them in text format. The model then associates the Latent Factors with the distribution of Concepts.

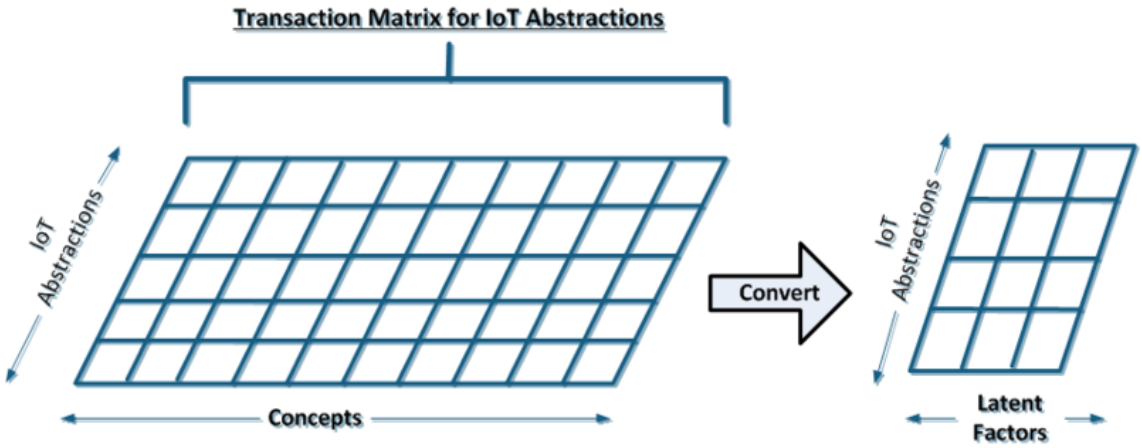


Figure 8: Latent Factor Extraction

Once the second training process is done, the clusters are then created. The number of clusters created is based on the number of Latent Factors generated. The vector of

Latent Factors that corresponds to an IoT Description is used to determine which Latent Factor best describes it.

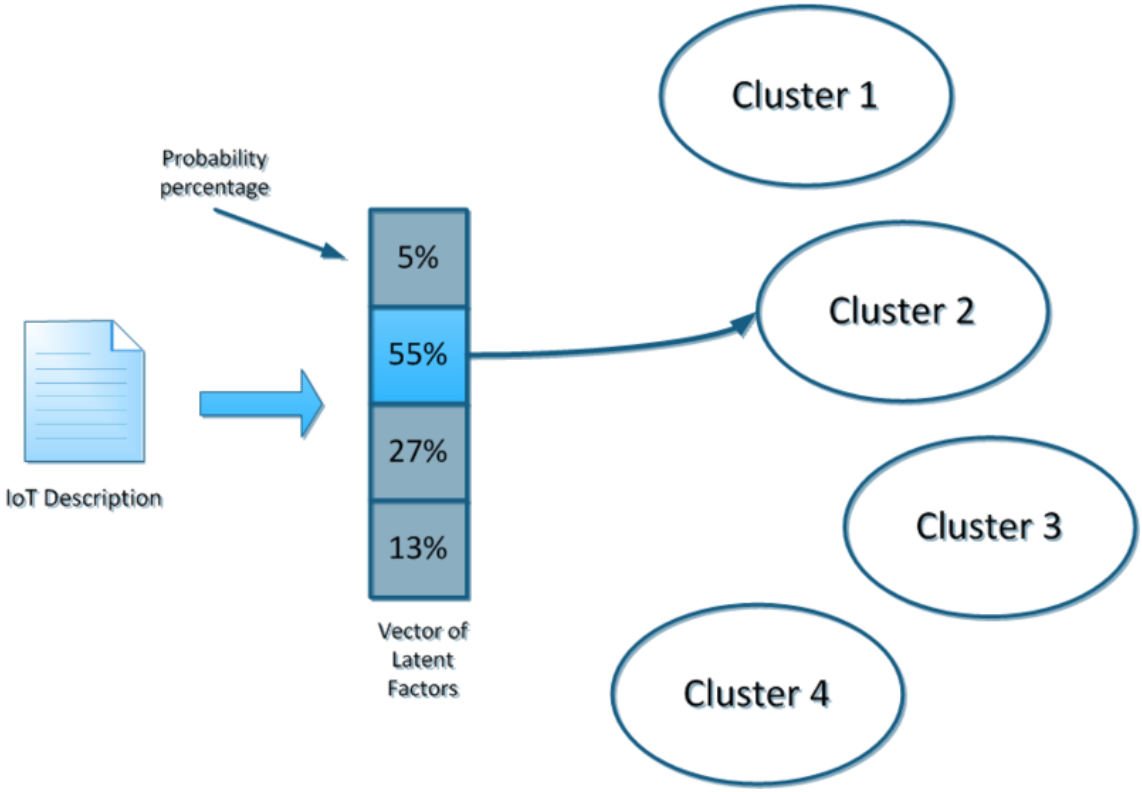


Figure 9: Folding in new Descriptions

After the training process, the IoT Search Engine becomes online and is ready to receive requests. During runtime, the repository is expected to receive more registrations of IoT Descriptions. When this occurs the IoT Search Engine will read the new IoT Description and "fold" it in to one of the clusters based in its probability of similarity with one of the Latent Factors in the vector.

3.8.2.4.1 Querying the Engine for Discovery

For discovery, a user can query the IoT Search Engine by providing a description template or a SPARQL query as the input to the search engine. The template should include concepts that are relevant to what the user is searching for. The search engine will then convert the query into Latent Factor vector - as done previously in the second training stage - and then match the query vector to the vector of Latent Factors that represents the IoT Descriptions stored in the repository. A query example is shown below.

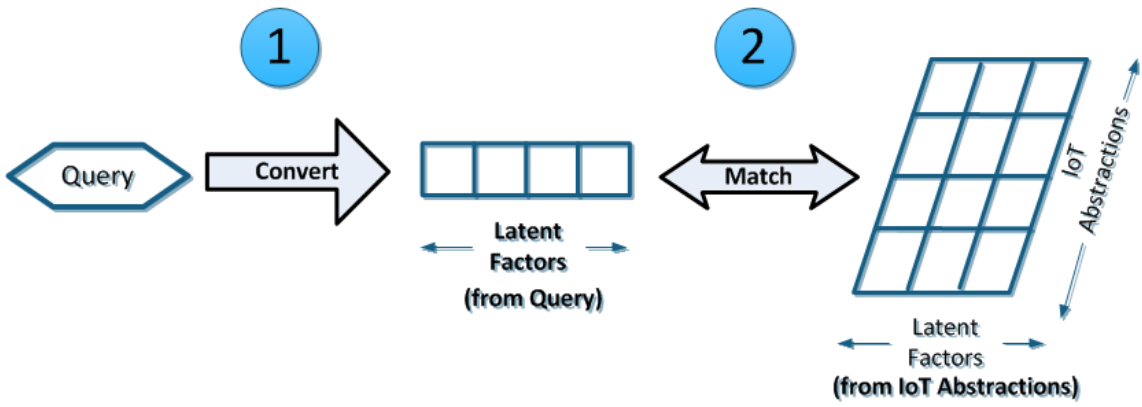


Figure 10: Query Match

A query about an Advanced Description is given in the form:

```
<?xml version="1.0"?>
 <rdf:RDF
    <vm:VirtualEntity>
    <vm:hasDomainAttribute>
     <vm:DomainAttribute>
       <vm:hasAttributeType
rdf:resource="http://purl.oclc.org/NET/ssnx/qu/quantity#humidity
"/>
       <vm:hasAttributeName
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
       >humidity</vm:hasAttributeName>
     </vm:DomainAttribute>
   </vm:hasDomainAttribute>
   <vm:hasName
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
   >BA 38 01</vm:hasName>
   <vm:hasA rdf:resource="#HumidityAttributeBA38"/>
   <vm:hasType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"

>http://www.surrey.ac.uk/ccsr/ontologies/LocationModel.owl#Room<
/vm:hasType>
 </vm:VirtualEntity>
</rdf:RDF>
```

In the above example, the query is asking for an Entity - in this case a room (38) - that has Humidity information about it.

The Search Engine does as follows:

1. The query is converted to a vector of latent factors: ***{0.038, 0.916, 0.006, 0.006, 0.0, 0.006, 0.01, 0.008, 0.003, 0.0}***
2. After analysing the vector, the query belongs to latent factor #2 with the highest probability.
3. Query is forwarded to the registry responsible for cluster #2 and query is matched within that cluster.
4. The first five results obtained are:

- Rank **#1**: 38_BA_01.owl with a score of **0.998479530076855**
- Rank **#2**: 44_BA_01.owl with a score of **0.998479530076855**

- Rank #**3**: 01_BA_02.owl with a score of **0.998479530076855**
- Rank #**4**: 37_BA_01.owl with a score of **0.998479530076855**
- Rank #**5**: 18_BA_02.owl with a score of **0.998479530076855**

Once this is done, the respective IoT Descriptions are retrieved from the repository and returned to the user.

3.8.2.5 *IoT Associations Engine (Semantic)*

3.8.2.5.1 Introduction

The Association Engine (AE) is a component that establishes and maintains associations between IoT Advanced Descriptions such as Things/Entities and Resources, via a type of Advanced description known as a Service Description, which acts as the representative for interacting with a Resource. The associations are made based on spatial, temporal and thematic relativity:

- **Thematic** (feature) match - using domain ontologies that capture an Entity's attributes and IoT Service's input/output parameter.
- **Spatial** match - using either co-ordinate comparison between IoT Entities and Services for determining close proximity, or location ontologies that model logical locations with properties such as 'contains'
- **Temporal** match - utilising temporal aspects of entities which have a temporal aspect, such as meeting rooms with the IoT Service's observation_schedule

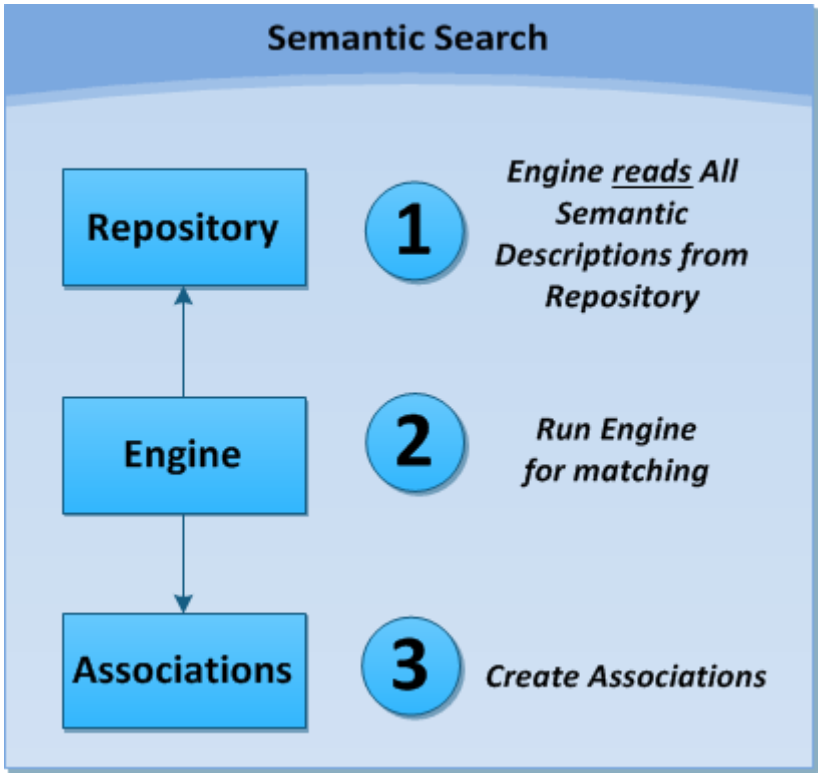


Figure 11: Association Process Overview

(2)Creating the Associations

Formulation of associations is done using the IoT-A Entity and Service models (i.e. Advanced Descriptions). Dynamic association inference is maintained through rules that incrementally reason on theme, spatial aspects and time. At startup, the AE will retrieve from the database all the different types of Advanced Descriptions - Entities, and Services (N.B. Services represent Resources) - and processes them through an Association Engine for matching. The matching process first looks for similar themes between Entity Attributes and Service observation types, e.g. temperature, light intensity, acoustic level. It will then look for similar locations. Lastly in the matching process it will look for temporal availability of a Service. Once the matching process is done, the associations generated will then be stored, and can then be queried via SPARQL.

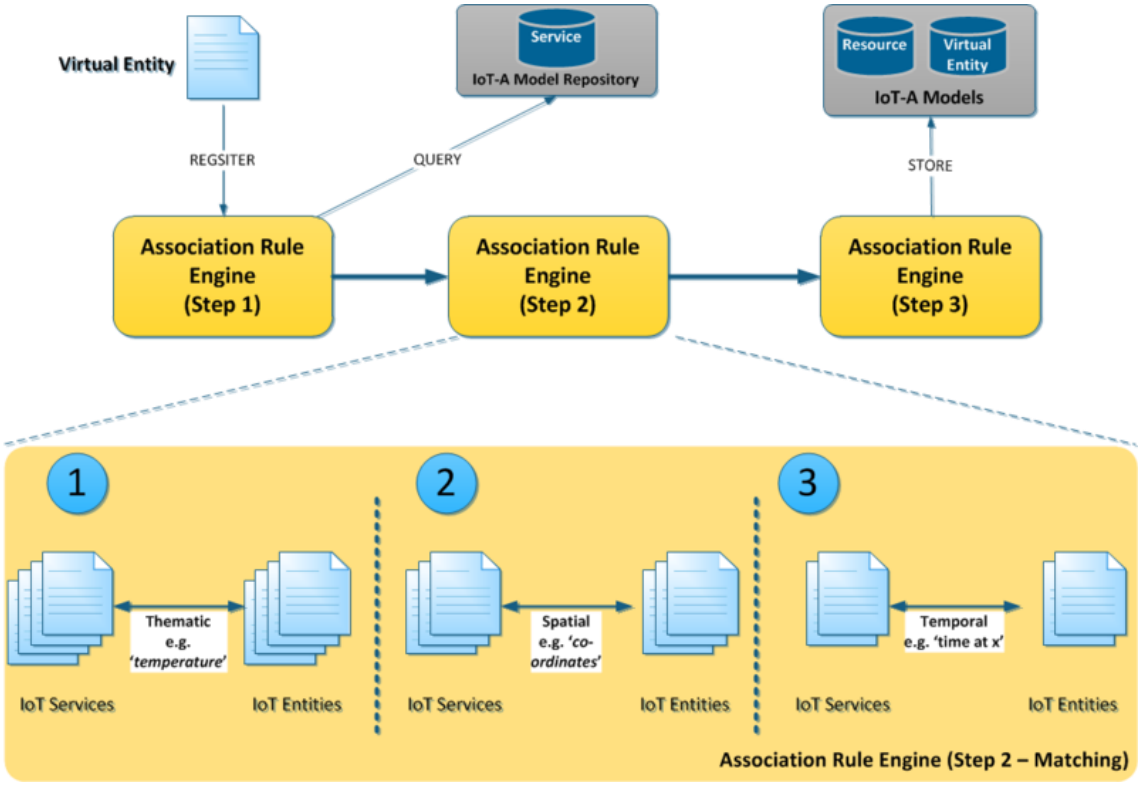


Figure 12: Association Mechanism

3.8.2.5.1 Querying for Associations

The association can then be queried by users via the SPARQL interface. In the query the predicate used to retrieve the association is "*isAssociatedToService*". This will return the URI of the IoT-A service description, which can then be retrieved by querying the IoT-A Service repository. It is this description that will provide access to contextual information (e.g. temperature reading) produced by the IoT-A Resource (e.g. temperature sensor).

3.8.2.5.2 Linking NGSI Entities to Advanced Descriptions

A Context Producer can link an NGSI Entity Description to an Advanced Description by adding a Context Attribute that provides the URI of the corresponding Advanced Description, when registering the NGSI Entity via the *registerContextRequest* operation. This requires the Advanced Description be registered beforehand.

3.8.2.6 *New Interactions*

3.8.2.6.1 Context Broker Semantic Extension (Data/Context Chapter)

The Sense2Web module may interact with the Semantic Extension of the Context Broker by acting as a repository for providing ontologies that are specifically related to the IoT. The Context Broker is able to query the repository via the SPARQL interface. Since the context broker receives ContextML and NGSI descriptions, the Context Broker needs to retrieve more information or metadata about the Entity in question. This can be done by linking the semantically-converted ContextML description to the IoT Descriptions stored in the Semantic Repository of the Sense2Web module. For example, if a ContextML description contains information about a room and its temperature value, then metadata with conventional naming about the room and temperature can be retrieved, e.g. room location, ambient entities, and temperature unit. The issue with only using NGSI to provide this information is that NGSI does not provide a convention for any defined entity and its attributes.

3.9 Terms and definitions

This section comprises a summary of terms and definitions introduced during the previous sections. It intends to establish a vocabulary that will be help to carry out discussions internally and with third parties (e.g., Use Case projects in the EU FP7 Future Internet PPP). For a summary of terms and definitions managed at overall FI-WARE level, please refer to FIWARE Global Terms and Definitions

- **Thing**. One instance of a physical object, living organism, person or concept interesting to a person or an application from the perspective of a FI-WARE instance, one particular usage area or to several usage areas. Examples of physical objects are: building, table, bridge (classes), the Kreml, the tennis table from John's garden, the Tower bridge (instances). Examples of living organisms are: frog, tree, person (classes), the green frog from the garden, the oak tree in front of this building, Dr. Green.
- **Class of thing**. Defines the type of a thing in the style of object-oriented programming: a construct that is used as a blueprint to create instances, defining constituent members which enable class instances to have state and behavior. An example of class of thing is "person", whereas an instance of a thing is "Dr. Green", with all the static/dynamically changing properties of this particular person.
- **Group of things**. A physical/logical group of things. Examples are all office buildings from Munich, all bridges above the Thames, all screws from a drawer, the planes of Lufthansa currently above Germany, all cars inside a traffic jam driven by a female driver, the trees from the Amazonian forest, the squids from the Mediterranean see the mushrooms from Alice's garden, all the fish from the aquarium of John., the soccer team of Manchester, the colleagues from OrangeLabs currently working *abroad* (from the perspective of France), all patients above 60 years of age of Dr. Green, the traffic jams from Budapest, the wheat crop in France in the year 2011, the Research and Development Department of the company Telefónica.
- **Device**. Hardware entity, component or system that may be in a relationship with a *thing* or a *group of things* called *association*. A device has the means either to measure properties of a thing/group of things and convert it to an analog or digital signal that can be read by a program or user or to potentially influence the properties of a thing/group of things or both to measure/influence.

In case when it can only measure the properties, then we call it a sensor. In case when it can potentially influence the properties of a thing, we call it an actuator. Sensors and actuators may be elementary or composed from a set of elementary sensors/actuators. The simplest elementary sensor is a piece of hardware measuring a simple quantity, such as speed of the wind, and displaying it in a mechanical fashion. Sensors may be the combination of software and hardware components, such as a vehicle on-board unit, that consists of simple sensors measuring various quantities such as the speed of the car, fuel consumption etc and a wireless module transmitting the measurement result to an application platform. The simplest elementary actuator is a light switch. We have emphasized *potentially* because as a result of the light switch being switched on it is not sure that the effect will be that light bulb goes on: only an *associated* sensor will be able to determine whether it went on or not. Other examples of devices are smart meters, mobile POS devices. More sophisticated devices may have a unique identifier, embedded computing capabilities and local data storage utilities, as well as embedded communication capabilities over short and/or long distances to communicate with IoT backend. Simple devices may not have all these capabilities and they can for instance only disclose the measurement results via mechanical means. In this latter case the further disclosure of the measurement result towards IoT backend requires another kind of device, called IoT Gateway.

- **Association**. It is a physical/logical relationship between a device and a thing or a device and a group of things or a group of devices and a group of things from the perspective of a FI-WARE instance, an application within a usage area project or other stakeholder. A device is associated with a thing if it can sense or (potentially) influence at least one property of the thing, property capturing an aspect of the thing interesting to a person or an application from the perspective of a FI-WARE instance, one particular usage area or to several usage areas. Devices are associated with things in *fully dynamic* or *mainly static* or *fully static* manner. The association may have several different embodiments: *physical attachment*, *physical embedding*, *physical neighborhood, logical relation* etc. Physical attachment means that the device is physically attached to the thing in order to monitor and interact with it, enabling the thing to be connected to the Internet. An example is the on-board device installed inside the vehicle to allow sending sensor data from the car to the FI-WARE instances. Physical embedding means that the device is deeply built inside the thing or almost part of the thing. An example of physical embedding is the GPS sensor inside a mobile phone. Physical neighborhood means that the device is in the physical neighborhood of the thing, but not in physical contact with it. An example of physical neighborhood is the association of the mobile phone of a subscriber who is caught in a traffic jam with the traffic jam itself: the mobile phone is the device, the traffic jam is the thing and the association means that the mobile phone is in the physical neighborhood of the area inside which the traffic jam is constrained (e.g. within 100 m from the region where the average speed of cars is below 5 km/h). Logical association means that there is a relationship between the device and the thing which is neither fully physical in the sense of attachment or embedding, nor fully physical proximity related. An example of logical association is between the car and the garage door opener of the garage where the car usually parks during the night.
- **IoT gateway**. A device that additionally to or instead of sensing/actuating provides inter-networking and protocol conversion functionalities between devices and IoT backend potentially in any combination of these hosts 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. 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 devices such as the available computing, power, storage and energy consumption. The level of functional split between the IoT backend and the IoT gateway will also depend on the available resources on the IoT gateway, the cost and quality of connectivity and the desired level for the distribution of intelligence and service abstraction.

- **IoT resource**. Computational elements (software) that provide the technical means to perform sensing and/or actuation on the device. There may be one-to-one or one-to-many relationship between a device and its IoT resource. Actuation capabilities exposed by the IoT resource may comprise configuration of the management/application features of the device, such as connectivity, access control, information, while sensing may comprise the gathering of faults, performance metrics, accounting/administration data from the device, as well as application data about the properties of the thing with which the device is associated. The resource is usually hosted on the device.
- **Management service**. It is the feature of the IoT resource providing programmatic access to readable and/or writable data belonging to the functioning of the device, comprising a subset of the FCAPS categories, that is, configuration management, fault management, accounting /access management/administration, performance management/provisioning and security management. The extent of the subset depends on the usage area. Example of configuration management data is the IP endpoint of the IoT backend instance to which the device communicates. Example of fault management is an error given as a result of the overheating of the device because of extensive exposure to the sun. Example of accounting/administration is setting the link quota for applications using data from a certain sensor. Example of security management is the provisioning of a list of device ID-s of peer devices with which the device may directly communicate.
- **Application service**. It is the feature of the IoT resource providing programmatic access to readable or writable data in connection with the thing which is associated with the device hosting the resource. The application service exchanges application data with another device (including IoT gateway) and/or the IoT backend. Measured sensory data are example of application data flowing from devices to sensors. Setting the steering angle of a security camera or sending a “start wetting” command to the irrigation system are examples of application data flowing from the application towards the sensor.
- **Event**. An event can be defined as an activity that happens, occurs in a device, gateway, IoT backend or is created by a software component inside the IoT service enablement. The digital representation of this activity in a device, IoT resource, IoT gateway or IoT backend instance, or more generally, in a FI-WARE instance, is also called an event. Events may be simple and complex. A simple event is an event that is not an abstraction or composition of other events. An example of a simple event is that “smart meter X got broken”. A complex event is an abstraction of other events called member events. For example a stock market crash or a cellular network blackout is an abstraction denoting many thousand of member events. In many cases a complex event references the set of its members, the implication being that the event contains a reference. For example, the cellular network blackout may be caused by a power failure of the air conditioning in the operator’s data center. In other cases such a reference may not exist. For example there is no accepted agreement as to which events are members of a stock market crash complex event. Complex events may be created of simple events or other complex events by an IoT resource or the IoT backend instance.

- **IoT Backend.** Provides management functionalities for the devices and IoT domain-specific support for the applications. Integrant component of FI-WARE instances.
- **IoT application**. An application that uses the application programming interface of the IoT service enablement component. May have parts running on one/set of devices, one/set of IoT gateways and one/set of IoT backends.
- **Virtual thing**. It is the digital representation of a thing inside the IoT service enablement component. Consists of a set of properties which are interesting to a person or an application from the perspective of a FI-WARE instance, one particular usage area or to several usage areas. Classes of things have the same set of properties, only the value of the properties change from instance to instance.

4 FIWARE.OpenSpecification.IoT.Backend.DeviceManagement

Name	FIWARE.OpenSpecification.IoT.Backend.DeviceManagement		
Chapter	IoT Services Enablement,		
Catalogue-Link to Implementation	http://catalogue.fi-ware.org/enablers/backend-device-management	**Owner**	Telefonica I+D, Carlos Ralli

4.1 Preface

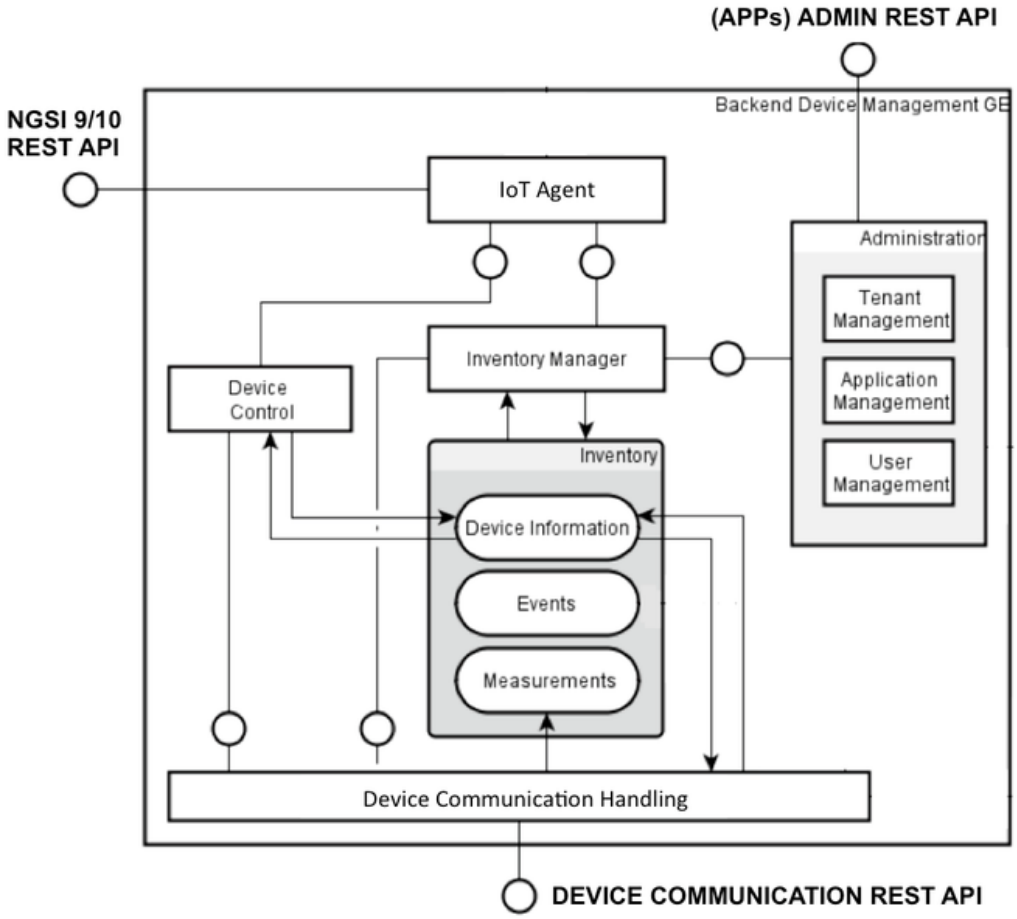
Within this document you find a self-contained open specification of a FI-WARE generic enabler, please consult as well the FI-WARE_Product_Vision, the website on http://www.fi-ware.org and similar pages in order to understand the complete context of the FI-WARE project.

4.2 Copyright

4.3 Legal Notice

4.4 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 devices connectivity.



4.5 Main Components

4.5.1 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 access to 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.

4.5.2 Inventory

The Inventory component allows to:

- 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, as detailed later, and Fragments. The Inventory component consists of subcomponents for capturing device information, events, readings, alarms.

4.5.2.1 *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.

4.5.2.2 *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.

4.5.2.3 *Measurements*

Measurements represent regularly acquired readings and statistics from sensors. Measurements consist of a timestamp (when the measurement was taken) and the unique identifiers of the source of the measurement.

4.5.3 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.

4.5.4 Device Communication Handling

To shield the backend applications from the diversity of IoT protocols, parameters and network connectivity options, the Backend Device Management GE uses the so-called agents or plug-ins at the Device Communication Handling level. 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 sent to an agent ("push") or queried by the agent ("poll") through a device-specific protocol. The agent will convert these messages into the protocol that the Backend Device Management GE understands. 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 and eventshave 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 used 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.

4.5.5 Administration

The Administration component is a set of REST-based admin interfaces responsible for Management of Tenant (or M2M service framework), Application and User.

4.5.5.1 *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.

4.5.5.2 *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 basis.

4.5.5.3 *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.

4.5.6 Restful API Framework

The Backend Device Management GE provides a set of REST interfaces for the applications.

4.6 Basic Concepts

The Backend Device Management GE is based on the usage of the following protocols: FI-WARE NGSI and ETSI M2M.

4.6.1 FI-WARE NGSI

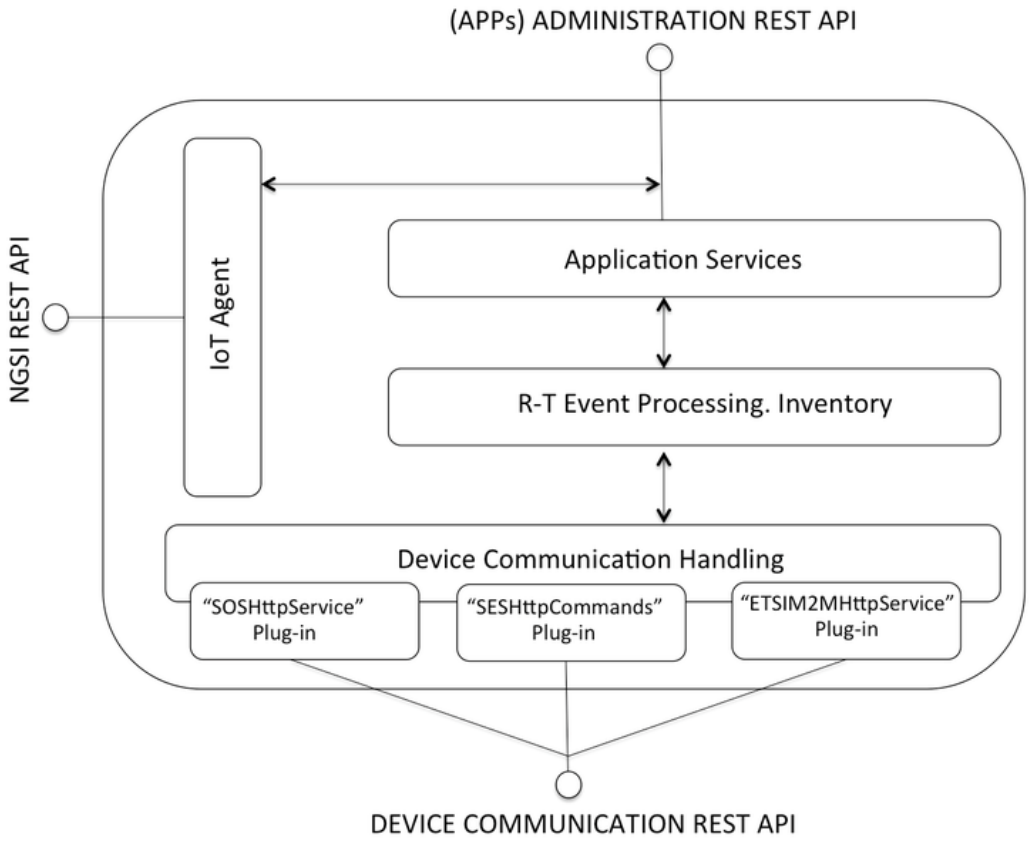
- NGSI Context Management - Open Mobile Alliance V1.0 Aug 3, 2010
- FI-WARE NGSI Open API Specification
- Context Information interface OMA_NGSI_10

4.6.2 ETSI M2M

- ETSI M2M Architecture Overview
- ETSI M2M Specifications rel. 1

4.7 Main Interactions

The following picture depicts the interaction model of the Backend Device Management GE.



While the diagram showed in the previous section is more a functional one, the one right above shows the actual architecture as a set of assets, components, interfaces and APIs of the Backend Device Management GE.

4.7.1 Application M2M Service Creation

The first operational step of the BE Device Management GE is to create one (or more) M2M service framework. This M2M service will hold afterwards a number of devices to communicate with. For this purpose, Applications or end users will access the ADMINISTRATION REST API and create an M2M service.

4.7.2 Device Register

Once an M2M service has been created by an end-user, devices are able to register into that specific framework by sending a request to the SensorML COMMUNICATION REST API of the Backend Device Management GE.

4.7.3 Device Observation

Once a device is registered into an existing M2M service, it may send periodic or standalone observations (i.e. sensor readings) by sending requests to the SensorML COMMUNICATION REST API at the BE Device Management GE.

4.7.4 Application Subscription Service

Applications are able to subscribe to device notification events in two ways:

- Subscription to specific devices within a M2M service.
- Subscription to all devices within a M2M service.

The subscription is realized by sending the proper request to the ADMINISTRATION REST API of the Backend Device Management GE.

4.7.5 Application Device Command Service

Applications are able to send commands to bidirectional devices that have provided before a callback URL for that purpose. For this to happen, Applications send a request to the ADMINISTRATION REST API of the Backend Device Management GE that may return in turn the reslt of the command or a connectivity error if not accessible.

4.7.6 Interconnection with NGSI enabled components

Applications may interconnect an M2M service framework with other functional NGSI-enabled GEs via the NGSI9/10 API of the Backend Device Management GE. Those NGSI-enabled components will be receiving NGSI notifications whenever devices events (register, observations) occur.

4.8 Basic Design Principles

Today's IoT framework considers a number of vertical silos exploiting different M2M technologies and protocols. Thus, the Backend Device Management GE aims to offer a single Administration REST API interface for Applications as well as an NGSI adaptor (NGSI REST API) transforming device events into NGSI notifications. In order to cope with several technologies, the Device Communication REST API will support several ones: SensorML, ETSI M2M, etc.

4.9 Re-utilised Technologies/Specifications

The Backend Device Management GE is based on RESTful Design Principles. The technologies and specifications used in this GE are:

- RESTful web services
- HTTP/1.1 (RFC2616)
- XML data serialization format.

4.10 Terms and definitions

This section comprises a summary of terms and definitions introduced during the previous sections. It intends to establish a vocabulary that will be help to carry out discussions internally and with third parties (e.g., Use Case projects in the EU FP7 Future Internet PPP). For a summary of terms and definitions managed at overall FI-WARE level, please refer to FIWARE Global Terms and Definitions

- **Thing**. One instance of a physical object, living organism, person or concept interesting to a person or an application from the perspective of a FI-WARE instance, one particular usage area or to several usage areas. Examples of physical objects are: building, table, bridge (classes), the Kreml, the tennis table from John's garden, the Tower bridge (instances). Examples of living organisms

are: frog, tree, person (classes), the green frog from the garden, the oak tree in front of this building, Dr. Green.

- **Class of thing**. Defines the type of a thing in the style of object-oriented programming: a construct that is used as a blueprint to create instances, defining constituent members which enable class instances to have state and behavior. An example of class of thing is “person”, whereas an instance of a thing is “Dr. Green”, with all the static/dynamically changing properties of this particular person.
- **Group of things**. A physical/logical group of things. Examples are all office buildings from Munich, all bridges above the Thames, all screws from a drawer, the planes of Lufthansa currently above Germany, all cars inside a traffic jam driven by a female driver, the trees from the Amazonian forest, the squids from the Mediterranean see the mushrooms from Alice’s garden, all the fish from the aquarium of John., the soccer team of Manchester, the colleagues from OrangeLabs currently working *abroad* (from the perspective of France), all patients above 60 years of age of Dr. Green, the traffic jams from Budapest, the wheat crop in France in the year 2011, the Research and Development Department of the company Telefónica.
- **Device**. Hardware entity, component or system that may be in a relationship with a *thing* or a *group of things* called *association*. A device has the means either to measure properties of a thing/group of things and convert it to an analog or digital signal that can be read by a program or user or to potentially influence the properties of a thing/group of things or both to measure/influence. In case when it can only measure the properties, then we call it a sensor. In case when it can potentially influence the properties of a thing, we call it an actuator. Sensors and actuators may be elementary or composed from a set of elementary sensors/actuators. The simplest elementary sensor is a piece of hardware measuring a simple quantity, such as speed of the wind, and displaying it in a mechanical fashion. Sensors may be the combination of software and hardware components, such as a vehicle on-board unit, that consists of simple sensors measuring various quantities such as the speed of the car, fuel consumption etc and a wireless module transmitting the measurement result to an application platform. The simplest elementary actuator is a light switch. We have emphasized *potentially* because as a result of the light switch being switched on it is not sure that the effect will be that light bulb goes on: only an *associated* sensor will be able to determine whether it went on or not. Other examples of devices are smart meters, mobile POS devices. More sophisticated devices may have a unique identifier, embedded computing capabilities and local data storage utilities, as well as embedded communication capabilities over short and/or long distances to communicate with IoT backend. Simple devices may not have all these capabilities and they can for instance only disclose the measurement results via mechanical means. In this latter case the further disclosure of the measurement result towards IoT backend requires another kind of device, called IoT Gateway.
- **Association**. It is a physical/logical relationship between a device and a thing or a device and a group of things or a group of devices and a group of things from the perspective of a FI-WARE instance, an application within a usage area project or other stakeholder. A device is associated with a thing if it can sense or (potentially) influence at least one property of the thing, property capturing an aspect of the thing interesting to a person or an application from the perspective of a FI-WARE instance, one particular usage area or to several usage areas. Devices are associated with things in *fully dynamic* or *mainly static* or *fully static* manner. The association may have several different embodiments: *physical attachment*, *physical embedding*, *physical neighborhood, logical relation* etc.

Physical attachment means that the device is physically attached to the thing in order to monitor and interact with it, enabling the thing to be connected to the Internet. An example is the on-board device installed inside the vehicle to allow sending sensor data from the car to the FI-WARE instances. Physical embedding means that the device is deeply built inside the thing or almost part of the thing. An example of physical embedding is the GPS sensor inside a mobile phone. Physical neighborhood means that the device is in the physical neighborhood of the thing, but not in physical contact with it. An example of physical neighborhood is the association of the mobile phone of a subscriber who is caught in a traffic jam with the traffic jam itself: the mobile phone is the device, the traffic jam is the thing and the association means that the mobile phone is in the physical neighborhood of the area inside which the traffic jam is constrained (e.g. within 100 m from the region where the average speed of cars is below 5 km/h). Logical association means that there is a relationship between the device and the thing which is neither fully physical in the sense of attachment or embedding, nor fully physical proximity related. An example of logical association is between the car and the garage door opener of the garage where the car usually parks during the night.

- **IoT gateway**. A device that additionally to or instead of sensing/actuating provides inter-networking and protocol conversion functionalities between devices and IoT backend potentially in any combination of these hosts 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. 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 devices such as the available computing, power, storage and energy consumption. The level of functional split between the IoT backend and the IoT gateway will also depend on the available resources on the IoT gateway, the cost and quality of connectivity and the desired level for the distribution of intelligence and service abstraction.
- **IoT resource**. Computational elements (software) that provide the technical means to perform sensing and/or actuation on the device. There may be one-to-one or one-to-many relationship between a device and its IoT resource. Actuation capabilities exposed by the IoT resource may comprise configuration of the management/application features of the device, such as connectivity, access control, information, while sensing may comprise the gathering of faults, performance metrics, accounting/administration data from the device, as well as application data about the properties of the thing with which the device is associated. The resource is usually hosted on the device.
- **Management service**. It is the feature of the IoT resource providing programmatic access to readable and/or writable data belonging to the functioning of the device, comprising a subset of the FCAPS categories, that is, configuration management, fault management, accounting /access management/administration, performance management/provisioning and security management. The extent of the subset depends on the usage area. Example of configuration management data is the IP endpoint of the IoT backend instance to which the device communicates. Example of fault management is an error given as a result of the overheating of the device because of extensive exposure to the sun. Example of accounting/administration is setting the link quota for applications using data from a certain sensor. Example of security management is the provisioning of a list of device ID-s of peer devices with which the device may directly communicate.

- **Application service**. It is the feature of the IoT resource providing programmatic access to readable or writable data in connection with the thing which is associated with the device hosting the resource. The application service exchanges application data with another device (including IoT gateway) and/or the IoT backend. Measured sensory data are example of application data flowing from devices to sensors. Setting the steering angle of a security camera or sending a “start wetting” command to the irrigation system are examples of application data flowing from the application towards the sensor.
- **Event**. An event can be defined as an activity that happens, occurs in a device, gateway, IoT backend or is created by a software component inside the IoT service enablement. The digital representation of this activity in a device, IoT resource, IoT gateway or IoT backend instance, or more generally, in a FI-WARE instance, is also called an event. Events may be simple and complex. A simple event is an event that is not an abstraction or composition of other events. An example of a simple event is that “smart meter X got broken”. A complex event is an abstraction of other events called member events. For example a stock market crash or a cellular network blackout is an abstraction denoting many thousand of member events. In many cases a complex event references the set of its members, the implication being that the event contains a reference. For example, the cellular network blackout may be caused by a power failure of the air conditioning in the operator’s data center. In other cases such a reference may not exist. For example there is no accepted agreement as to which events are members of a stock market crash complex event. Complex events may be created of simple events or other complex events by an IoT resource or the IoT backend instance.
- **IoT Backend.** Provides management functionalities for the devices and IoT domain-specific support for the applications. Integrant component of FI-WARE instances.
- **IoT application**. An application that uses the application programming interface of the IoT service enablement component. May have parts running on one/set of devices, one/set of IoT gateways and one/set of IoT backends.
- **Virtual thing**. It is the digital representation of a thing inside the IoT service enablement component. Consists of a set of properties which are interesting to a person or an application from the perspective of a FI-WARE instance, one particular usage area or to several usage areas. Classes of things have the same set of properties, only the value of the properties change from instance to instance.

5 FIWARE OpenSpecification IoT Backend TemplateHandler

Name	FIWARE.OpenSpecification.IoT.Backend.TemplateHandler		
Chapter	IoT Services Enablement,		
Catalogue-Link to Implementation	[(to be inserted) Template Handler] Not validated by catalogue team	**Owner**	SAP, Carsten Magerkurth

5.1 Preface

Within this document you find a self-contained open specification of a FI-WARE generic enabler, please consult as well the FI-WARE Product Vision, the website on http://www.fi-ware.eu and similar pages in order to understand the complete context of the FI-WARE project.

5.2 Copyright

5.3 Legal Notice

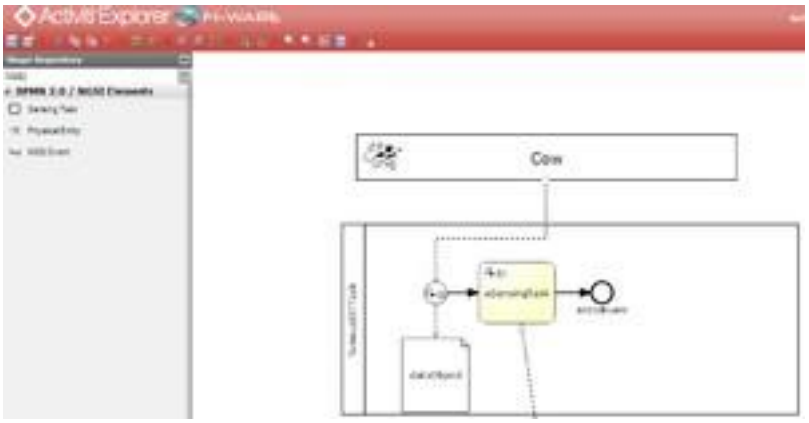
5.4 Overview

The NGSI Backend Template Handler allows modeling and executing BPMN business processes. BPMN is a description standard for business processes. A quick introduction as well as the full standard can be found via the official BPMN home page. The Template Handler enriches this standard by additional NGSI-enabled components.

Template Handler consists of two components:

5.4.1 Modeler

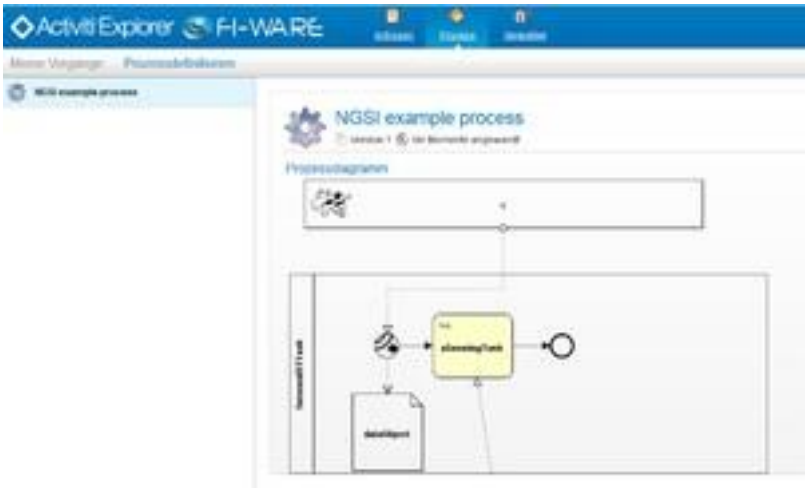
The modeler allows users to model BPMN processes and enriches BPMN with the NGSI specific extensions.



Modeler user interface

5.4.2 Execution environment

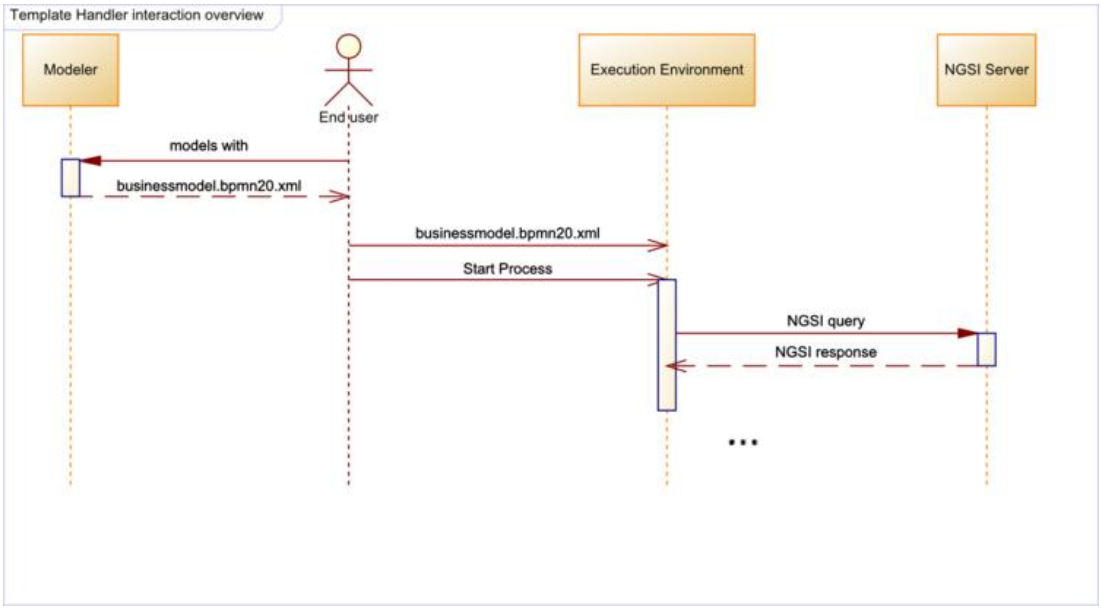
Activiti is an environment for BPMN process execution. It has been extended for the execution of NGSI sensing tasks to request values from the NGSI server. Additionally, start events may trigger business processes on receipt of NGSI notifications.



Execution engine's web interface

5.4.3 Data model outline

Template Handler bridges the gap between business process modelling level and IoT concepts. Existing graphical and executable process modelling languages are not designed for representing IoT-aware business processes. Therefore Template Handler introduces an extension of the existing BPMN seeking to lower the barrier for applying IoT technology like sensors and actuators to current and new business processes. Process modellers without a deep programmatic understanding get an advanced guidance during the modelling activities, so that they can create IoT-aware processes. The created and abstract process models are stored in files based on BPMN 2.0 XML format extended by IoT-specific elements. These new elements are derived from the NGSI specification defining how to retrieve context information in a NGSI compliant environment. This abstract process models are constitutively completed by a process resolution environment that will create an executable IoT-aware process model. Finally, the process models presented in the IAPMC BPMN 2.0 integration can be executed by the IAPMC BPMN 2.0 compliant process execution engine, which is also part of the Template Handler.



5.4.4 Functionality outline

The functionality of the Template Handler Theme is to provide the means to define and execute IoT-aware business process templates/models. For the process template provision within the Template Handler Theme, it is essential to provide means of explicitly modelling process models or templates with respective IoT-specific BPMN extensions, in addition to the semi-automatic recommendations of the template handler. This aspect relates to the Exposure Theme which will be capable of utilizing process models with such IoT extensions. The IoT extensions take into account the idiosyncrasies of IoT services such as the inherent uncertainty of sensor information, the mobility of IoT resources, or the temporal dynamicity of physical things. In order to model IoT-aware business processes with the respective BPMN extensions, the Template Handler will provide a set of new modelling concepts in addition to the standard concepts form the BPMN world. This Enabler will be capable of serializing the process models for the Process Handler of the Exposure.

5.5 Main Concepts

5.5.1 Basic Concepts

5.5.1.1 *FI-WARE NGSI*

To gather information about the context Template Handler follows the NGSI standard, to be found under NGSI Context Management v1.0. The communication protocol is HTTP and follows NGSI RestFUL Binding, v1.0

There is a FI-WARE NGSI Context Management tutorial.

5.5.1.2 *BPMN*

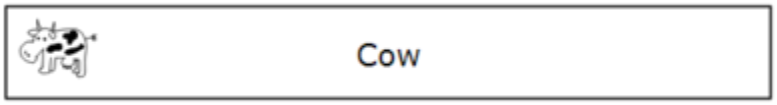
Business process modeling complies with the BPMN v2.0 standard. An unofficial overview for the most important elements can be found at this tutorial.

5.5.2 Additional Concepts

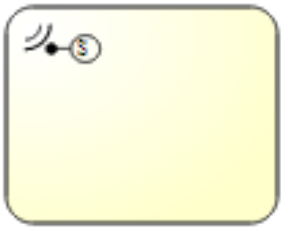
5.5.2.1 *NGSI specific BPMN extensions*

In general, Template Handler is a modeling and execution platform for BPMN processes. In order to be integrated in a real world environment, Template Handler enables them to access information on the existing entities. The transmission is based on the NGSI standard. However, Template Handler acts on a very high level of abstraction. Users without any detailed knowledge of the NGSI protocol are able to use measured information about these real world entities within business processes. The BPMN standard is extended by the following new elements:

- **PhysicalEntity**: A Physical Entity in an IoT-aware business process represents the real-world entity, with which the process interacts. Its graphical representation is a collapsed pool decorated with a cow. In order to specify the real-world object, an NGSI EntityId has to be defined by the process modeler. Its symbol is the picture below, a box with a cow picture and the entity's id as label.



- **SensingTask**: A Sensing Task is a newly introduced BPMN task type needed for the interaction of the process with the Physical Entity. It is drawn as a BPMN task decorated with a WLAN symbol. For the execution of such a task, an NGSI10 queryContext operation is executed. Therefore the ContextAttribute to be measured must be specified in the SensingTask by the process modeler. The EntityId for the NGSI queryContext operation is taken from the PhysicalEntity to which the SensingTask is connected. After the execution of the queryContext operation the value returned from the NGSI Context Management Component is written to the DataObject connected to the SensingTask, so that it is available during the remainder of the execution of the process. Its symbol is shown below, a task symbol with a start event in the upper left corner.



- **NGSIStartEvent**: If a business process is to be started due to certain events in the real world, an NGSIStartEvent can be used as the first element of a sequence flow. Like the SensingTask, this is connected to a PhysicalEntity, which holds the specification of the NGSI EntityId, and a DataObject, which receives the real-world attribute value. When a business process with an NGSIStartEvent is loaded into a business process execution engine, the engine sends a subscribeContextRequest to the NGSI Context Management Component. Therefore the attributes to be observed for events, the NGSI NotifyCondition and the duration of the subscription have to be specified in the NGSIStartEvent by the process modeler. When the execution engine receives the notifyContextRequest from the NGSI Context Management Component, it

writes the real-world value into the associated DataObject and starts the process. Its symbol is the following:

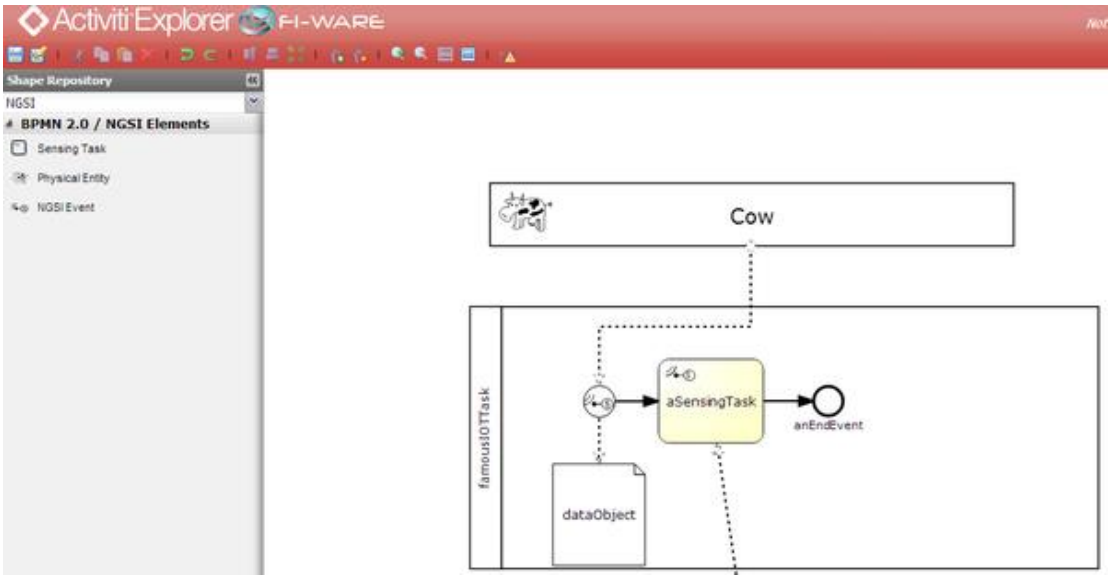
5.6 Main Interactions

The Template Handler GE offers interfaces to interact with end users on the one hand and NGSI capable network devices on the other hand.

5.6.1 End User Interactions

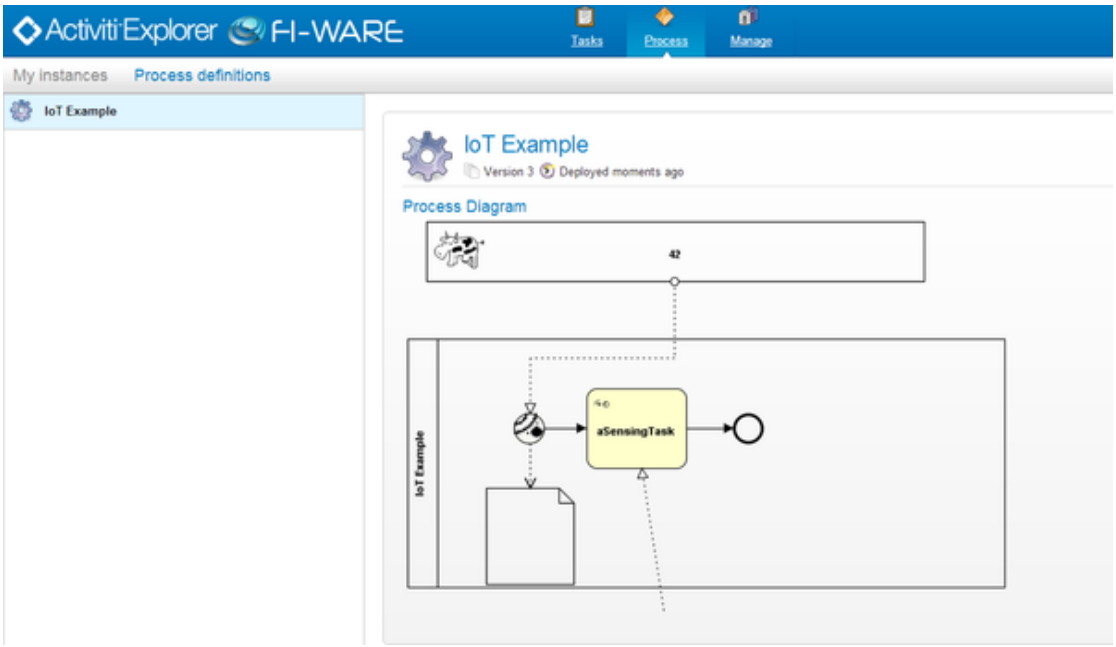
5.6.1.1 *Modeling processes*

The modeler allows users to model BPMN processes and enriches BPMN with the NGSI specific extensions.



5.6.1.2 *Executing processes*

The Activiti Explorer is an environment for BPMN process execution. It has been extended for the execution of NGSI sensing tasks to request values from the NGSI server. Additionally, start events may trigger business processes on receipt of NGSI notifications.



5.6.2 NGSI Interactions

The NGSI server component is a part of the Template Handler. It is intended to receive and store context information from other participants in the IoT network environment.

5.6.2.1 *Context Query*

Receiving information about a context entity is possible due to the implementation of the "Query Context Operation" defined in the NGSI specification.

5.6.2.2 *Context Update*

Updating information about a context entity is possible due to the implementation of the "Update Context Operation" defined in the NGSI specification. This may trigger context notifications to be sent to interested network participants.

5.6.2.3 *Subscription Request & Notification*

The NGSI server is capable to receive "subscribeContextRequests" according to the NGSI specification. These express the wish to be informed when the state of another context entity changes. In effect, notifications containing the information are sent to other participant's "notifyContext" interface.

5.7 Basic Design Principles

- **No knowledge about the NGSI domain required:** Template Handler a fully integrated toolkit so users can use BPMN and NGSI together at a high level of abstraction. All necessary steps happen only based on the graphical web interface. They can be simply executed through graphical interface, without any understanding of the programming behind it.
- **All in one:** Template Handler is a standalone toolkit. It requires nothing but a Java/Web server environment and does not depend on other software. So

modeling and executing BPMN processes integrates smoothly without compatibility issues.

5.8 Re-utilised Technologies/Specifications

The interfaces of the IoT Template Handler are based on RESTful Design Principles. The technologies and specifications used in this GE are:

- RESTful web services
- HTTP/1.1 (RFC2616)
- XML data serialization format.

5.9 Detailed Open Specifications

FIWARE.OpenSpecification.Details.IoT.Backend.TemplateHandler

5.10 Terms and definitions

This section comprises a summary of terms and definitions introduced during the previous sections. It intends to establish a vocabulary that will be help to carry out discussions internally and with third parties (e.g., Use Case projects in the EU FP7 Future Internet PPP). For a summary of terms and definitions managed at overall FI-WARE level, please refer to FIWARE Global Terms and Definitions

- **Thing**. One instance of a physical object, living organism, person or concept interesting to a person or an application from the perspective of a FI-WARE instance, one particular usage area or to several usage areas. Examples of physical objects are: building, table, bridge (classes), the Kreml, the tennis table from John's garden, the Tower bridge (instances). Examples of living organisms are: frog, tree, person (classes), the green frog from the garden, the oak tree in front of this building, Dr. Green.
- **Class of thing**. Defines the type of a thing in the style of object-oriented programming: a construct that is used as a blueprint to create instances, defining constituent members which enable class instances to have state and behavior. An example of class of thing is "person", whereas an instance of a thing is "Dr. Green", with all the static/dynamically changing properties of this particular person.
- **Group of things**. A physical/logical group of things. Examples are all office buildings from Munich, all bridges above the Thames, all screws from a drawer, the planes of Lufthansa currently above Germany, all cars inside a traffic jam driven by a female driver, the trees from the Amazonian forest, the squids from the Mediterranean see the mushrooms from Alice's garden, all the fish from the aquarium of John., the soccer team of Manchester, the colleagues from OrangeLabs currently working *abroad* (from the perspective of France), all patients above 60 years of age of Dr. Green, the traffic jams from Budapest, the wheat crop in France in the year 2011, the Research and Development Department of the company Telefónica.
- **Device**. Hardware entity, component or system that may be in a relationship with a *thing* or a *group of things* called *association*. A device has the means

either to measure properties of a thing/group of things and convert it to an analog or digital signal that can be read by a program or user or to potentially influence the properties of a thing/group of things or both to measure/influence. In case when it can only measure the properties, then we call it a sensor. In case when it can potentially influence the properties of a thing, we call it an actuator. Sensors and actuators may be elementary or composed from a set of elementary sensors/actuators. The simplest elementary sensor is a piece of hardware measuring a simple quantity, such as speed of the wind, and displaying it in a mechanical fashion. Sensors may be the combination of software and hardware components, such as a vehicle on-board unit, that consists of simple sensors measuring various quantities such as the speed of the car, fuel consumption etc and a wireless module transmitting the measurement result to an application platform. The simplest elementary actuator is a light switch. We have emphasized *potentially* because as a result of the light switch being switched on it is not sure that the effect will be that light bulb goes on: only an *associated* sensor will be able to determine whether it went on or not. Other examples of devices are smart meters, mobile POS devices. More sophisticated devices may have a unique identifier, embedded computing capabilities and local data storage utilities, as well as embedded communication capabilities over short and/or long distances to communicate with IoT backend. Simple devices may not have all these capabilities and they can for instance only disclose the measurement results via mechanical means. In this latter case the further disclosure of the measurement result towards IoT backend requires another kind of device, called IoT Gateway.

- **Association**. It is a physical/logical relationship between a device and a thing or a device and a group of things or a group of devices and a group of things from the perspective of a FI-WARE instance, an application within a usage area project or other stakeholder. A device is associated with a thing if it can sense or (potentially) influence at least one property of the thing, property capturing an aspect of the thing interesting to a person or an application from the perspective of a FI-WARE instance, one particular usage area or to several usage areas. Devices are associated with things in *fully dynamic* or *mainly static* or *fully static* manner. The association may have several different embodiments: *physical attachment*, *physical embedding*, *physical neighborhood, logical relation* etc. Physical attachment means that the device is physically attached to the thing in order to monitor and interact with it, enabling the thing to be connected to the Internet. An example is the on-board device installed inside the vehicle to allow sending sensor data from the car to the FI-WARE instances. Physical embedding means that the device is deeply built inside the thing or almost part of the thing. An example of physical embedding is the GPS sensor inside a mobile phone. Physical neighborhood means that the device is in the physical neighborhood of the thing, but not in physical contact with it. An example of physical neighborhood is the association of the mobile phone of a subscriber who is caught in a traffic jam with the traffic jam itself: the mobile phone is the device, the traffic jam is the thing and the association means that the mobile phone is in the physical neighborhood of the area inside which the traffic jam is constrained (e.g. within 100 m from the region where the average speed of cars is below 5 km/h). Logical association means that there is a relationship between the device and the thing which is neither fully physical in the sense of attachment or embedding, nor fully physical proximity related. An example of logical association is between the car and the garage door opener of the garage where the car usually parks during the night.
- **IoT gateway**. A device that additionally to or instead of sensing/actuating provides inter-networking and protocol conversion functionalities between

devices and IoT backend potentially in any combination of these hosts 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. 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 devices such as the available computing, power, storage and energy consumption. The level of functional split between the IoT backend and the IoT gateway will also depend on the available resources on the IoT gateway, the cost and quality of connectivity and the desired level for the distribution of intelligence and service abstraction.

- **IoT resource**. Computational elements (software) that provide the technical means to perform sensing and/or actuation on the device. There may be one-to-one or one-to-many relationship between a device and its IoT resource. Actuation capabilities exposed by the IoT resource may comprise configuration of the management/application features of the device, such as connectivity, access control, information, while sensing may comprise the gathering of faults, performance metrics, accounting/administration data from the device, as well as application data about the properties of the thing with which the device is associated. The resource is usually hosted on the device.
- **Management service**. It is the feature of the IoT resource providing programmatic access to readable and/or writable data belonging to the functioning of the device, comprising a subset of the FCAPS categories, that is, configuration management, fault management, accounting /access management/administration, performance management/provisioning and security management. The extent of the subset depends on the usage area. Example of configuration management data is the IP endpoint of the IoT backend instance to which the device communicates. Example of fault management is an error given as a result of the overheating of the device because of extensive exposure to the sun. Example of accounting/administration is setting the link quota for applications using data from a certain sensor. Example of security management is the provisioning of a list of device ID-s of peer devices with which the device may directly communicate.
- **Application service**. It is the feature of the IoT resource providing programmatic access to readable or writable data in connection with the thing which is associated with the device hosting the resource. The application service exchanges application data with another device (including IoT gateway) and/or the IoT backend. Measured sensory data are example of application data flowing from devices to sensors. Setting the steering angle of a security camera or sending a “start wetting” command to the irrigation system are examples of application data flowing from the application towards the sensor.
- **Event**. An event can be defined as an activity that happens, occurs in a device, gateway, IoT backend or is created by a software component inside the IoT service enablement. The digital representation of this activity in a device, IoT resource, IoT gateway or IoT backend instance, or more generally, in a FI-WARE instance, is also called an event. Events may be simple and complex. A simple event is an event that is not an abstraction or composition of other events. An example of a simple event is that “smart meter X got broken”. A complex event is an abstraction of other events called member events. For example a stock market crash or a cellular network blackout is an abstraction denoting many thousand of member events. In many cases a complex event references the set of its members, the implication being that the event contains a reference. For example, the cellular network blackout may be caused by a

power failure of the air conditioning in the operator's data center. In other cases such a reference may not exist. For example there is no accepted agreement as to which events are members of a stock market crash complex event. Complex events may be created of simple events or other complex events by an IoT resource or the IoT backend instance.

- **IoT Backend.** Provides management functionalities for the devices and IoT domain-specific support for the applications. Integrant component of FI-WARE instances.
- **IoT application**. An application that uses the application programming interface of the IoT service enablement component. May have parts running on one/set of devices, one/set of IoT gateways and one/set of IoT backends.
- **Virtual thing**. It is the digital representation of a thing inside the IoT service enablement component. Consists of a set of properties which are interesting to a person or an application from the perspective of a FI-WARE instance, one particular usage area or to several usage areas. Classes of things have the same set of properties, only the value of the properties change from instance to instance.

6 FIWARE OpenSpecification IoT Gateway DeviceManagement

Name	FIWARE.OpenSpecification.IoT.GatewayDeviceManagement		
Chapter	IoT Services Enablement,		
Catalogue-Link to Implementation	Gateway Device Management GE - Ericsson IoT Gateway	**Owner**	Fraunhofer FOKUS, Konrad Campowsky

6.1 Preface

Within this document you find a self-contained open specification of a FI-WARE generic enabler, please consult as well the FI-WARE Product Vision, the website on http://www.fi-ware.eu and similar pages in order to understand the complete context of the FI-WARE project.

6.2 Copyright

- Copyright © 2012 by Fraunhofer FOKUS

6.3 Legal Notice

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

6.4 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 (marshal/unmarshal). 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 is 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.

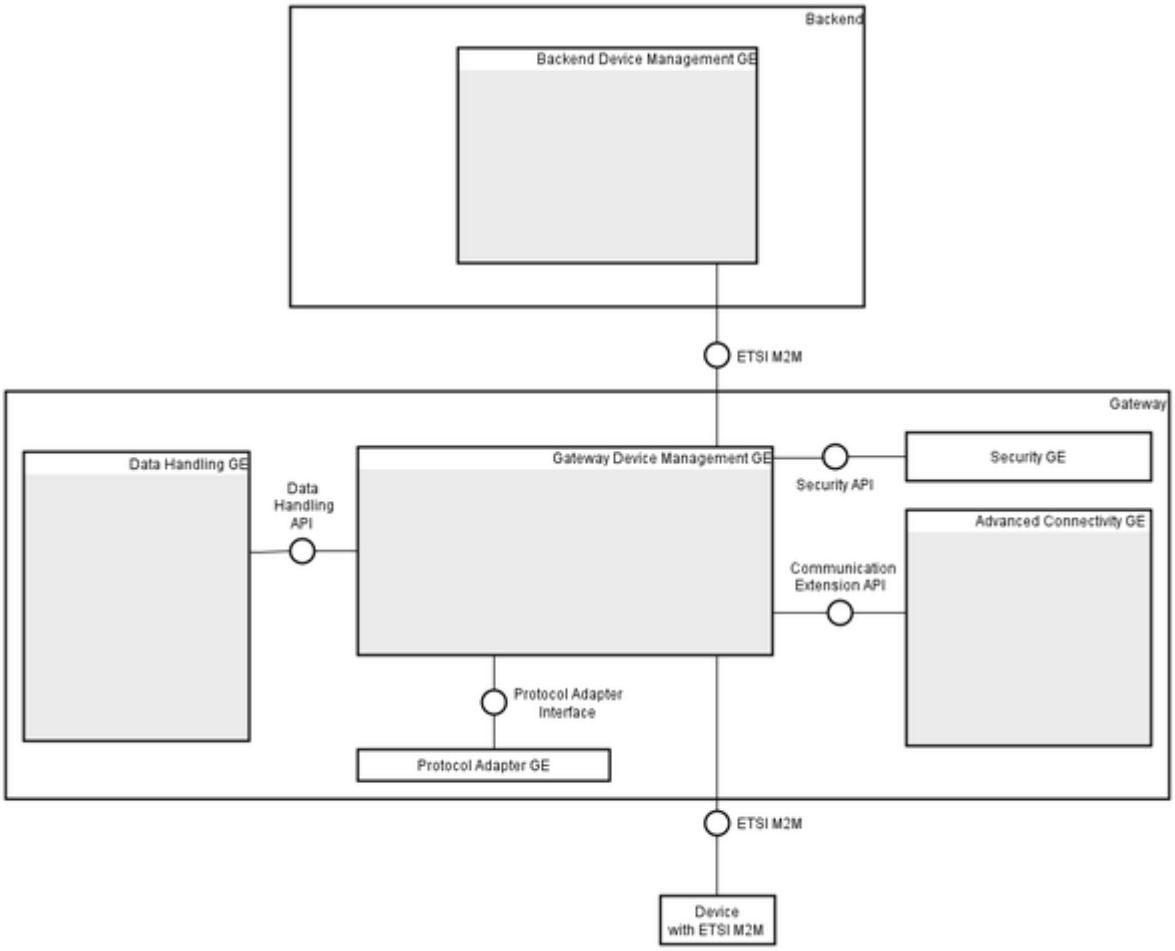


Figure 1: Gateway architecture

6.5 Basic Concepts

The IoT Gateway Device Management GE is based on the OMA NGSI context data model [1].

The core functionality of the GE is provided by the GSCL component of the OpenMTC [1] software developed at Fraunhofer FOKUS. This component natively provides the RESTful M2M interfaces as specified by ETSI [2] - specifically those described in TS 102.921 [3] and TS 102.690 [4]. In addition, an NGSI interface component has been developed according to FI-Ware specs.

6.5.1 FI-WARE NGSI

The IoT Gateway Device Management GE implements FI-WARE's RESTful binding specification of the OMA NGSI context management interface (FI-WARE NGSI specifications for short). More specifically, the GE implements the full set of NGSI-9 and NGSI-10 operations, where it acts both as a client and as a server.

FI-WARE NGSI Context Management specifications are based on the NGSI Context Management specifications defined by OMA (Open Mobile Alliance) [2]. 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 [3] and NGSI-10 [4].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 [5] (currently under construction) to learn the main concepts underlying FI-WARE NGSI Context Management specifications.

6.5.2 Architecture

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.

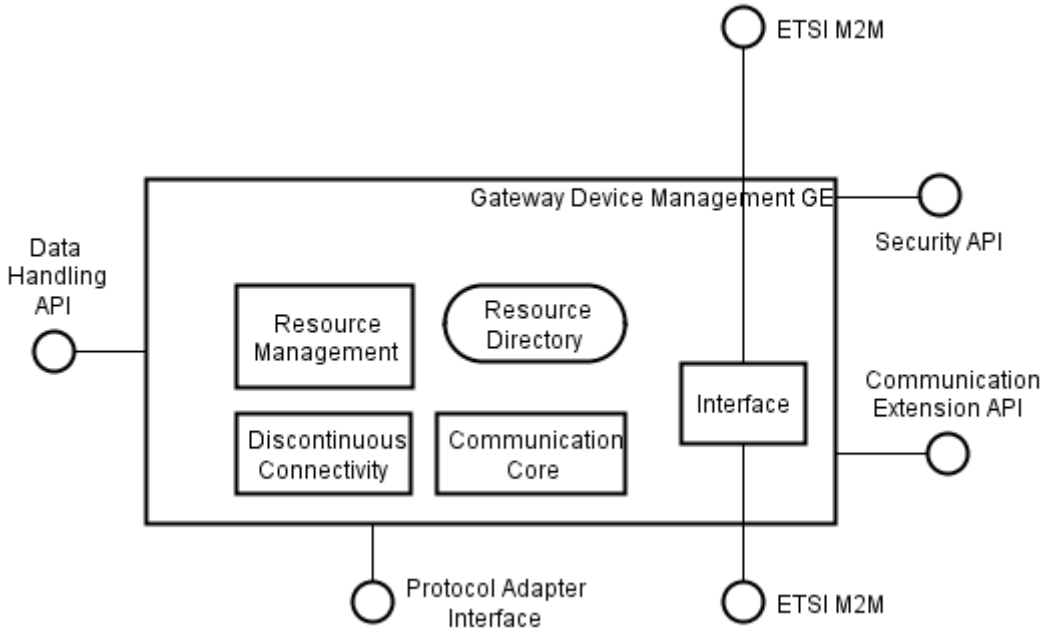


Figure 2: Gateway Device Management GE internal architecture

6.5.3 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.

6.5.4 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 a 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 a 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 then 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

6.5.5 Communication Core

The Communication Core contains the basic communication capabilities of the gateway to setup communications to the IoT Backend and IoT Devices.

6.6 Main interactions

6.6.1 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 communicating 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 of devices. Notable in this figure is that the Discontinuous 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.

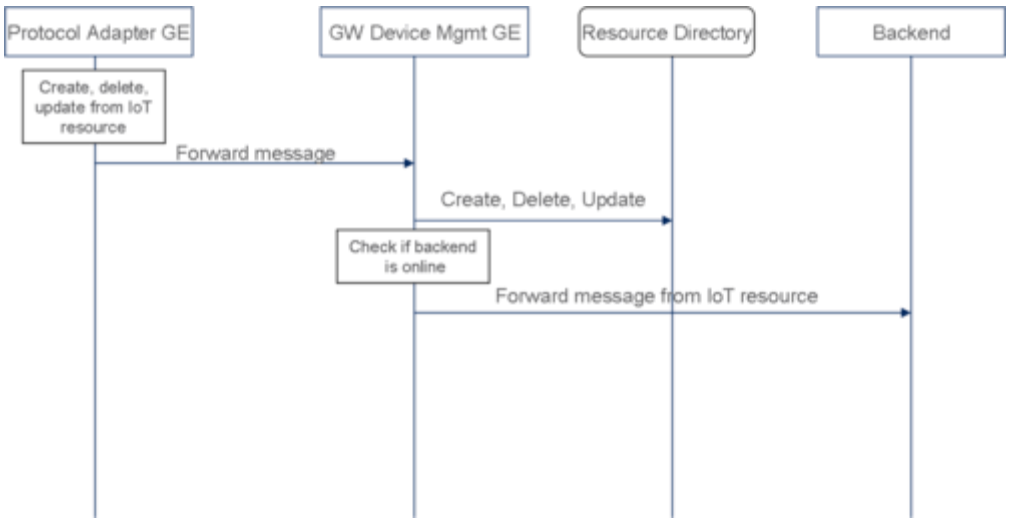


Figure 3: Resource Management (external resources)

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.

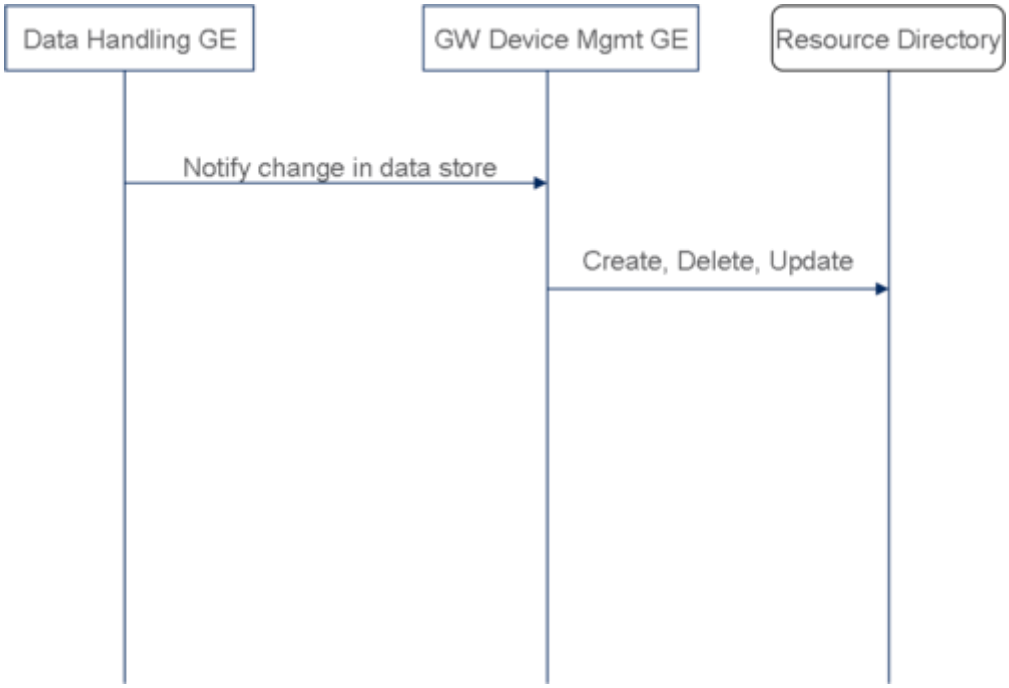


Figure 4: Resource Management (internal resources)

The Gateway Device Management GE exposes an API so that it is possible for the backend to read entries in the Resource Directory, i.e. discover which IoT resources that are exposed by the gateway.

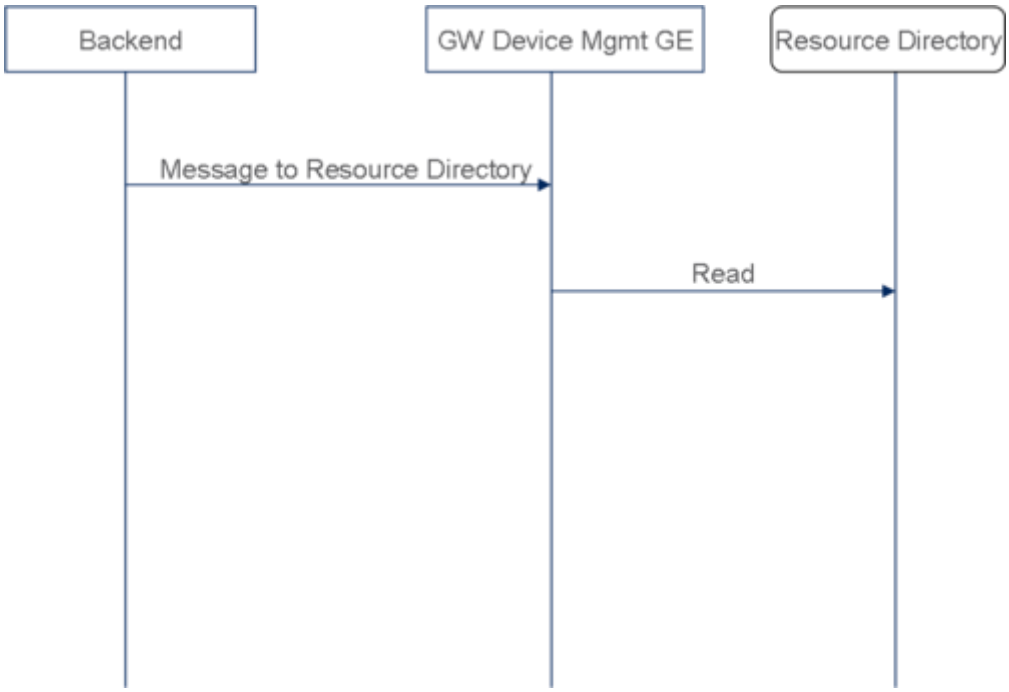


Figure 5: Lookup resources

6.6.2 Accessing resources

The Gateway Device Management GE enables 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 Discontinuous 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.

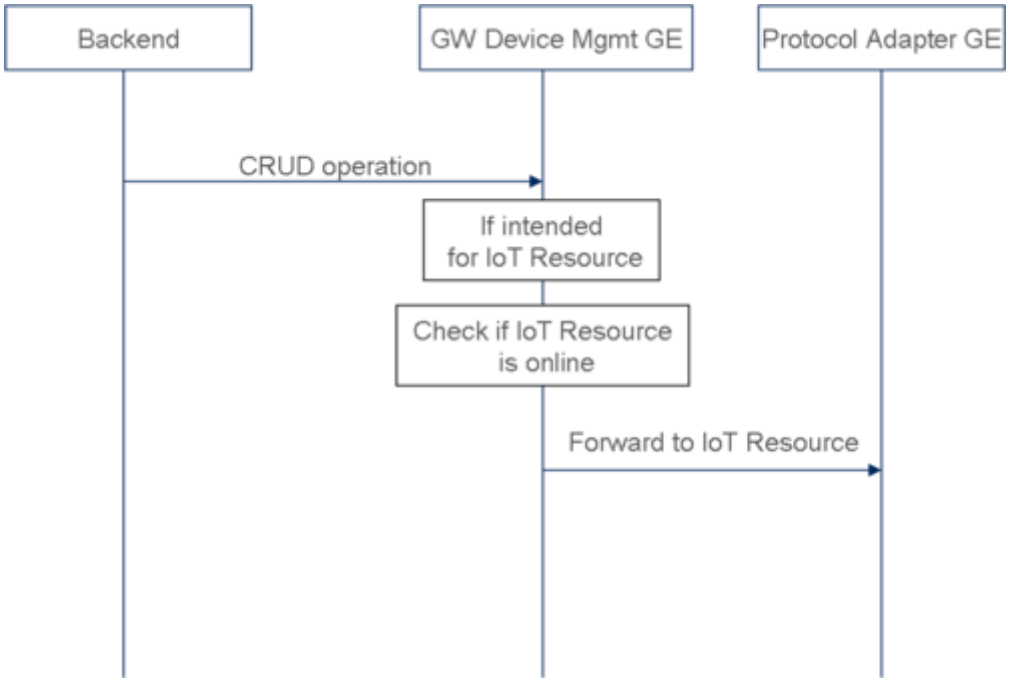


Figure 6: Access to resource hosted by device

Figure 7 shows the corresponding Create, Read, Update, Delete operations on a resource hosted by the gateway.

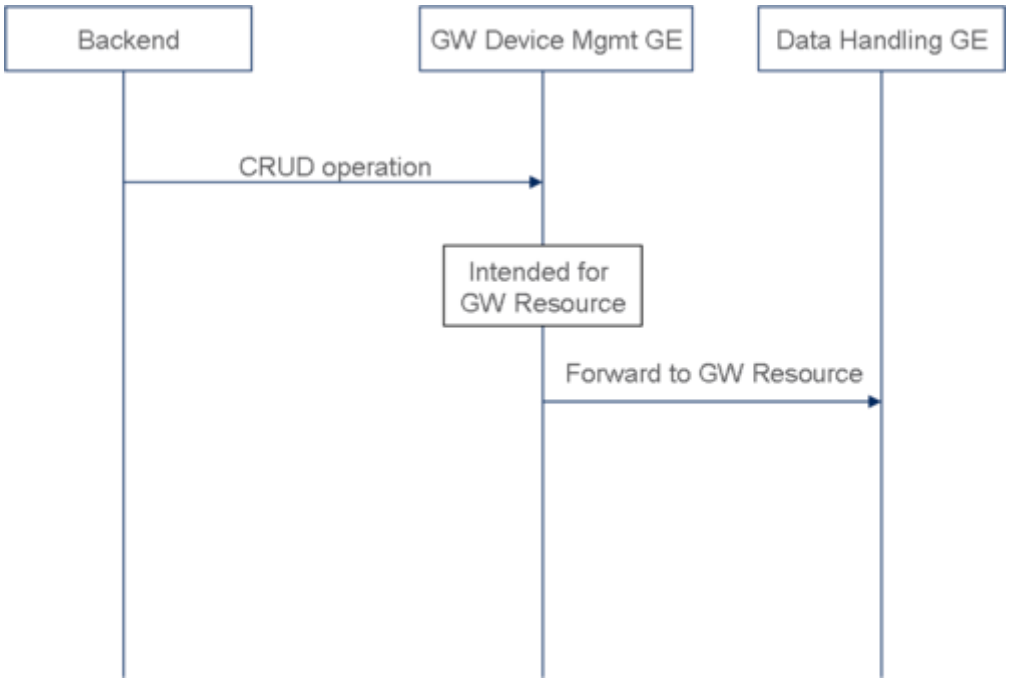


Figure 7: Access to resource hosted by the gateway

6.7 Re-utilised Technologies/Specifications

The Gateway Device Management GE is based on RESTful Design Principles. The technologies and specifications used in this GE are:

- RESTful web services
- HTTP/1.1 (RFC2616)

6.8 Detailed Open Specifications

The OpenMTC platform has been designed to act as a horizontal convergence layer in terms of a Machine-2-Machine (M2M) middleware for machine type communication that supports multiple vertical application domains. Those domains are usually the classic M2M verticals (market segments) such as transport and logistics, utilities, automotive, eHealth, etc. which can be deployed independently or as part of a common platform.

OpenMTC fosters the development of M2M applications on the gateways or over the top of the backend server though abstract high-level APIs by hiding the complexity of device resource structure and enabling more functionality. The APIs are classified in three categories:

- Data APIs handle functionalities related to accessing/manipulating data collected from devices/sensors.
- Network APIs deal with the roles related to the network applications and its session control with the M2M core.
- Device APIs find appropriate devices and gateway resources to fetch information from them.

6.9 Terms and definitions

This section comprises a summary of terms and definitions introduced during the previous sections. It intends to establish a vocabulary that will be help to carry out discussions internally and with third parties (e.g., Use Case projects in the EU FP7 Future Internet PPP). For a summary of terms and definitions managed at overall FI-WARE level, please refer to FIWARE Global Terms and Definitions

- **Thing**. One instance of a physical object, living organism, person or concept interesting to a person or an application from the perspective of a FI-WARE instance, one particular usage area or to several usage areas. Examples of physical objects are: building, table, bridge (classes), the Kreml, the tennis table from John's garden, the Tower bridge (instances). Examples of living organisms are: frog, tree, person (classes), the green frog from the garden, the oak tree in front of this building, Dr. Green.
- **Class of thing**. Defines the type of a thing in the style of object-oriented programming: a construct that is used as a blueprint to create instances, defining constituent members which enable class instances to have state and behavior. An example of class of thing is "person", whereas an instance of a thing is "Dr. Green", with all the static/dynamically changing properties of this particular person.
- **Group of things**. A physical/logical group of things. Examples are all office buildings from Munich, all bridges above the Thames, all screws from a drawer, the planes of Lufthansa currently above Germany, all cars inside a traffic jam driven by a female driver, the trees from the Amazonian forest, the squids from the Mediterranean see the mushrooms from Alice's garden, all the fish from the aquarium of John., the soccer team of Manchester, the colleagues from OrangeLabs currently working *abroad* (from the perspective of France), all patients above 60 years of age of Dr. Green, the traffic jams from Budapest, the wheat crop in France in the year 2011, the Research and Development Department of the company Telefónica.
- **Device**. Hardware entity, component or system that may be in a relationship with a *thing* or a *group of things* called *association*. A device has the means either to measure properties of a thing/group of things and convert it to an analog or digital signal that can be read by a program or user or to potentially influence the properties of a thing/group of things or both to measure/influence. In case when it can only measure the properties, then we call it a sensor. In case when it can potentially influence the properties of a thing, we call it an actuator. Sensors and actuators may be elementary or composed from a set of elementary sensors/actuators. The simplest elementary sensor is a piece of hardware measuring a simple quantity, such as speed of the wind, and displaying it in a mechanical fashion. Sensors may be the combination of software and hardware components, such as a vehicle on-board unit, that consists of simple sensors measuring various quantities such as the speed of the car, fuel consumption etc and a wireless module transmitting the measurement result to an application platform. The simplest elementary actuator is a light switch. We have emphasized *potentially* because as a result of the light switch being switched on it is not sure that the effect will be that light bulb goes on: only an *associated* sensor will be able to determine whether it went on or not. Other examples of devices are smart meters, mobile POS devices. More sophisticated devices may have a unique identifier, embedded computing capabilities and local data storage utilities, as well as embedded communication capabilities over short and/or long distances to communicate with IoT backend. Simple devices may not have all these capabilities and they can for instance only disclose the measurement results via mechanical means.

In this latter case the further disclosure of the measurement result towards IoT backend requires another kind of device, called IoT Gateway.

- **Association**. It is a physical/logical relationship between a device and a thing or a device and a group of things or a group of devices and a group of things from the perspective of a FI-WARE instance, an application within a usage area project or other stakeholder. A device is associated with a thing if it can sense or (potentially) influence at least one property of the thing, property capturing an aspect of the thing interesting to a person or an application from the perspective of a FI-WARE instance, one particular usage area or to several usage areas. Devices are associated with things in *fully dynamic* or *mainly static* or *fully static* manner. The association may have several different embodiments: *physical attachment*, *physical embedding*, *physical neighborhood, logical relation* etc. Physical attachment means that the device is physically attached to the thing in order to monitor and interact with it, enabling the thing to be connected to the Internet. An example is the on-board device installed inside the vehicle to allow sending sensor data from the car to the FI-WARE instances. Physical embedding means that the device is deeply built inside the thing or almost part of the thing. An example of physical embedding is the GPS sensor inside a mobile phone. Physical neighborhood means that the device is in the physical neighborhood of the thing, but not in physical contact with it. An example of physical neighborhood is the association of the mobile phone of a subscriber who is caught in a traffic jam with the traffic jam itself: the mobile phone is the device, the traffic jam is the thing and the association means that the mobile phone is in the physical neighborhood of the area inside which the traffic jam is constrained (e.g. within 100 m from the region where the average speed of cars is below 5 km/h). Logical association means that there is a relationship between the device and the thing which is neither fully physical in the sense of attachment nor embedding, nor fully physical proximity related. An example of logical association is between the car and the garage door opener of the garage where the car usually parks during the night.
- **IoT gateway**. A device that additionally to or instead of sensing/actuating provides inter-networking and protocol conversion functionalities between devices and IoT backend potentially in any combination of these hosts 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. 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 devices such as the available computing, power, storage and energy consumption. The level of functional split between the IoT backend and the IoT gateway will also depend on the available resources on the IoT gateway, the cost and quality of connectivity and the desired level for the distribution of intelligence and service abstraction.
- **IoT resource**. Computational elements (software) that provide the technical means to perform sensing and/or actuation on the device. There may be one-to-one or one-to-many relationship between a device and its IoT resource. Actuation capabilities exposed by the IoT resource may comprise configuration of the management/application features of the device, such as connectivity, access control, information, while sensing may comprise the gathering of faults, performance metrics, accounting/administration data from the device, as well as application data about the properties of the thing with which the device is associated. The resource is usually hosted on the device.
- **Management service**. It is the feature of the IoT resource providing

programmatic access to readable and/or writable data belonging to the functioning of the device, comprising a subset of the FCAPS categories, that is, configuration management, fault management, accounting /access management/administration, performance management/provisioning and security management. The extent of the subset depends on the usage area. Example of configuration management data is the IP endpoint of the IoT backend instance to which the device communicates. Example of fault management is an error given as a result of the overheating of the device because of extensive exposure to the sun. Example of accounting/administration is setting the link quota for applications using data from a certain sensor. Example of security management is the provisioning of a list of device ID-s of peer devices with which the device may directly communicate.

- **Application service**. It is the feature of the IoT resource providing programmatic access to readable or writable data in connection with the thing which is associated with the device hosting the resource. The application service exchanges application data with another device (including IoT gateway) and/or the IoT backend. Measured sensory data are example of application data flowing from devices to sensors. Setting the steering angle of a security camera or sending a “start wetting” command to the irrigation system are examples of application data flowing from the application towards the sensor.
- **Event**. An event can be defined as an activity that happens, occurs in a device, gateway, IoT backend or is created by a software component inside the IoT service enablement. The digital representation of this activity in a device, IoT resource, IoT gateway or IoT backend instance, or more generally, in a FI-WARE instance, is also called an event. Events may be simple and complex. A simple event is an event that is not an abstraction or composition of other events. An example of a simple event is that “smart meter X got broken”. A complex event is an abstraction of other events called member events. For example a stock market crash or a cellular network blackout is an abstraction denoting many thousand of member events. In many cases a complex event references the set of its members, the implication being that the event contains a reference. For example, the cellular network blackout may be caused by a power failure of the air conditioning in the operator’s data center. In other cases such a reference may not exist. For example there is no accepted agreement as to which events are members of a stock market crash complex event. Complex events may be created of simple events or other complex events by an IoT resource or the IoT backend instance.
- **IoT Backend.** Provides management functionalities for the devices and IoT domain-specific support for the applications. Integrant component of FI-WARE instances.
- **IoT application**. An application that uses the application programming interface of the IoT service enablement component. May have parts running on one/set of devices, one/set of IoT gateways and one/set of IoT backends.
- **Virtual thing**. It is the digital representation of a thing inside the IoT service enablement component. Consists of a set of properties which are interesting to a person or an application from the perspective of a FI-WARE instance, one particular usage area or to several usage areas. Classes of things have the same set of properties, only the value of the properties change from instance to instance.

6.10 References

[1] https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/NGSI_9/10_information_model

[2] http://technical.openmobilealliance.org/Technical/release_program/docs/CopyrightClick.aspx?pck=NGSI&file=V1_0-20101207-C/OMA-TS-NGSI_Context_Management-V1_0-20100803-C.pdf

[3] https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI-9_Open_RESTful_API_Specification_%28PRELIMINARY%29

[4] https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI-10_Open_RESTful_API_Specification_%28PRELIMINARY%29

[5] https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI_Context_Management_tutorial
FIWARE.OpenSpecification.Details.IoT.Gateway.DeviceManagement

7 FIWARE OpenSpecification IoT Gateway DataHandling

Name	FIWARE.ArchitectureDescription.IoT.Gateway.DataHandling		
Chapter	IoT Services Enablement,		
Catalogue-Link to Implementation	Gateway Datahandling GE	**Owner**	Orange, Laurent ARTUSIO

7.1 Preface

Within this document you find a self-contained open specification of a FI-WARE generic enabler, please consult as well the FI-WARE Product Vision, the website on http://www.fi-ware.eu and similar pages in order to understand the complete context of the FI-WARE project.

7.2 Copyright

7.3 Legal Notice

7.4 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 real-time data from different sources. A lot of applications expect to receive value-added data that are relevant to their needs, and this can be easily achieved in a decoupled manner thanks to the Complex Event Processing technology (CEP). This is also referred to as event stream analysis, or real time event correlation. Events are published by Event Producers and subscribed to by Event Consumers. One can only subscribe to events that are allowed by its granted access rights.

Typical applications that require Data Handling GE are sensor network applications, RFID readings, supply chains, scheduling and control of fabrication lines, air traffic, smart buildings, home automation, and so on. These applications have in common 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 CEP Engine component of the Data Handling GE typically processes input data, in order to generate a smaller set of output data, which avoids upper level software overloading and network flooding.

The Data Handling GE handles data streams from IoT devices that cannot continuously be online, for various reasons. Among them are possible network disruptions and power resource constraints. The latter typically requires optimizing and limiting the communications with the IoT resources. For this purpose, local storage is desirable, in order to cache the last available processed data and contexts.

The natural surrounding GEs for Data Handling are the IoT Broker, the Configuration Management, and the Protocol Adapter GE.

Although it can be operated standalone, the Data Handling GE typically receives events from the Gateway Protocol Adapter GE, and propagates the processed data towards the IoT Broker GE. Prior to propagating any data, the Data Handling GE must be registered to the Configuration Management GE, in order to be identified as a context producer.

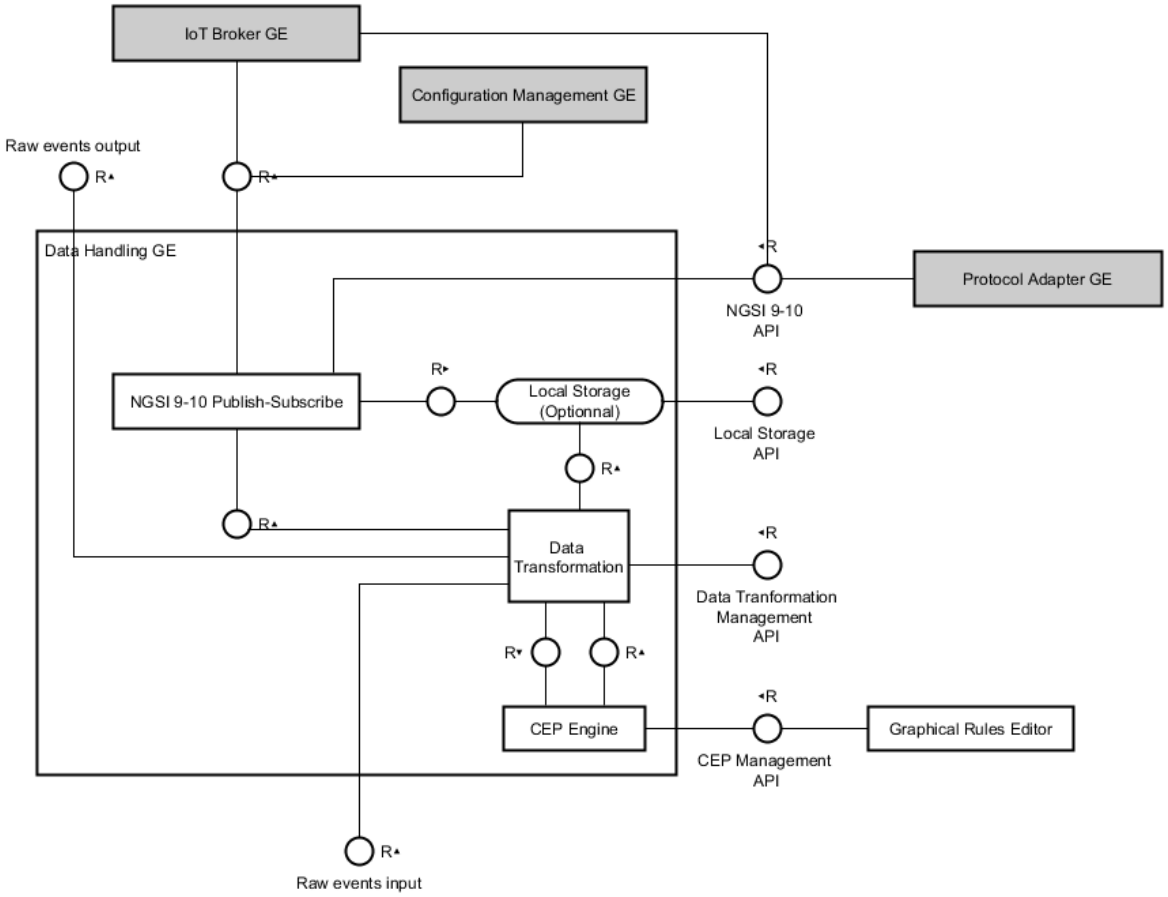
It should be noted that Publish-Subscribe and CEP features are also available in other GEs. This should not be seen as useless redundancy: some features are deliberately distributed at different level of granularity, in order to provide reliability to the whole architecture, and also to appropriately dispatch the load at different levels.

The Data Handling GE comes in two configurations: Esper4FastData and SOL/CEP. Esper4FastData targets both mobile and fixed IoT gateways: it supports both local storage and data transformation. Despite its gateway orientation, Esper4FastData could also be used server-side, whereas SOL/CEP is aimed at more constrained environments, which lack both the CPU resources and memory size for storage, or data transformation mechanisms. It targets small gateways, various devices and sensors that still have enough CPU power to run it.

Both configurations are equipped with a CEP Engine component, but they operate in distinct ways. The differences reflect the requirements for the underlying host hardware, on which the Data Handling GE is deployed and executed.

The next sections discuss the overall architecture reflecting the different configuration options. Then each component is described, if necessary in subsections according to the differences.

7.4.1 Internal architecture components diagram



Data Handling GE architecture

The interfaces for this GE are compliant with the FI-WARE implementation of the Next Generation Services Interface (NGSI), and more precisely, NGSI-9 and NGSI-10 subsections. It should be noted that the official OMA NGSI Context Management standard neither specifies nor imposes any particular implementation regarding the exchange format, but simply describes NGSI data types as belonging to the "XML Schema Part 2" W3C recommendation. For the time being, FI-WARE has chosen to support an XML implementation of the NGSI standard: more information can be found here.

7.4.2 Main interfaces

7.4.2.1 *NGSI-9/10 API*

The Data Handling API is NGSI compliant. It accepts events from any NGSI compliant The Data Handling API is NGSI-9/10 compliant. It accepts events from any NGSI compliant Event Producer. In the IoT Service Enablement architecture, the Gateway Device Management GE publishes device events and data towards the Data Handling GE.

In this setup, the Gateway Protocol Adapter GE registers as an NGSI context provider, by calling the NGSI-9 *registerContext* method exposed by the Data Handling GE.

After context registration, the Gateway Protocol Adapter GE can send events by calling the NGSI-10 *updateContext* method of the Data Handling GE.

The Data Handling GE also registers as an NGSI context provider, by calling the NGSI-9 *registerContext* method of the upper level GE that acts as an NGSI registrar. In the context of the IoT Services Enablement architecture, this GE is the IoT Backend Configuration Management. After registration, Data Handling GE calls the NGSI-10 *updateContext* method of the IoT Broker GE, for each propagated event. The IoT broker acts as an event consumer.

Third party software and other external GEs can subscribe to Data Handling GE output events by calling its NGSI-10 *subscribeContext* methods. For every subscriber, events are propagated by calling their NGSI-10 *notifyContext* method, which means subscribers must implement this method, in order for them to receive events.

7.4.2.2 *CEP Management API*

This API allows the configuration and operation of the Complex Event Processor: it provides features such as setting rules, defining event types, and configuring output event recipients.

In the standard configuration, it takes place as follows.

For more hardware-constrained configurations, it takes place as follows.

However, the meaning behind "standard" and "constrained" is left up to common sense and to the one who's in charge of implementing a solution. There are no strong guidelines regarding how much a system is embedded-oriented, hardware constrained, or server-oriented. For example, a PC-style gateway might run Esper4FastData, whereas a high-end sensor might better run SOL/CEP.

It should be noted that the CEP management API methods should ideally be consistent, whether the implementation concerns a constrained device, a mobile phone, or a powerful gateway.

7.4.2.3 *Local Storage API*

This API provides features to manage the local database at a low-level, and to query historical data.

This is not available in the SOL/CEP constrained configuration.

7.4.2.4 *Data Transformation Management API*

This API allows configuring transformers on both CEP input and output. Transformers define how to update event properties and structure on the fly, before and after event processing.

7.4.3 Main components

7.4.3.1 *NGSI-9/10 Publish-Subscribe component*

This component is the core implementation of the Data Handling GE NGSI functionality. It allows the Generic Enabler to communicate with the outside world and the other GEs, in an ingoing and outgoing way, and also features publish-subscribe functions. It implements the necessary subset of NGSI-9 and NGSI-10 methods for the purpose. It also allows the GE to register itself to the Context Broker GE, which is a prerequisite to the propagation of events towards the IoT Broker GE.

This organization enables the creation of a Publish/Subscribe network of NGSI compliant GEs, that expose their services to each others.

It should be noted that NGSI compliance does not imply the implementation of the full set of NGSI methods.

7.4.3.2 *CEP Engine Component*

This is the core functionality of the Data Handling GE.

The role of the CEP Engine component is to process events in real time, which mainly involves filtering, aggregating and merging data. This processing is based on a set of rules that can be defined by the CEP administrator.

The CEP Engine component can process XML data structures on its input. It is fed by the Data Transformation component, which is in charge of XSLT event transformations before processing. Events don't directly reach the CEP Engine component from outside.

Once processed, events pass through Data Transformation for optional output XSLT transformation purpose, before being finally sent to their recipients, through the Publish-Subscribe component.

7.4.3.3 *Data Transformation component*

This component performs XSLT transformations on both input and output XML events.

The Data Handling GE supports XML events. It might also supports other lower level formats, when configured for constrained devices with SOL/CEP.

When a NGSI-compliant event arrives, it first flows through the NGSI 9-10 Publish-Subscribe component before being pushed towards Data Transformation and thus transformed or not, depending on configured transformations. Other XML non-NGSI events come directly to the input of Data Transformation.

In all cases, NGSI input events are finally translated into a simplified CEP-friendly XML format, that is easily understood by the CEP Engine component, and easier to address with CEP rules scripting language. Events are then pushed towards CEP Engine.

Data Tranformation also receives processed events from CEP Engine output, for optional XSLT transformation purpose, before the processed and transformed events are finally propagated towards external configured recipients.

It should be noted that in the case of the SOL/CEP implementation for constrained environments, the Data Transformation component is not available.

7.4.3.4 *Local Storage component*

This component stores output events from the Publish-Subscribe component and other XML output events. For example, it could retain historical records of sensor readings. Compulsory metadata are locally added to the actual stored data, such as source identification and local time stamps.

The amount of stored events can be configured, but of course depends on the available memory, which may vary greatly among devices and gateways.

The Local Storage component provides methods to access and manage historical event data.

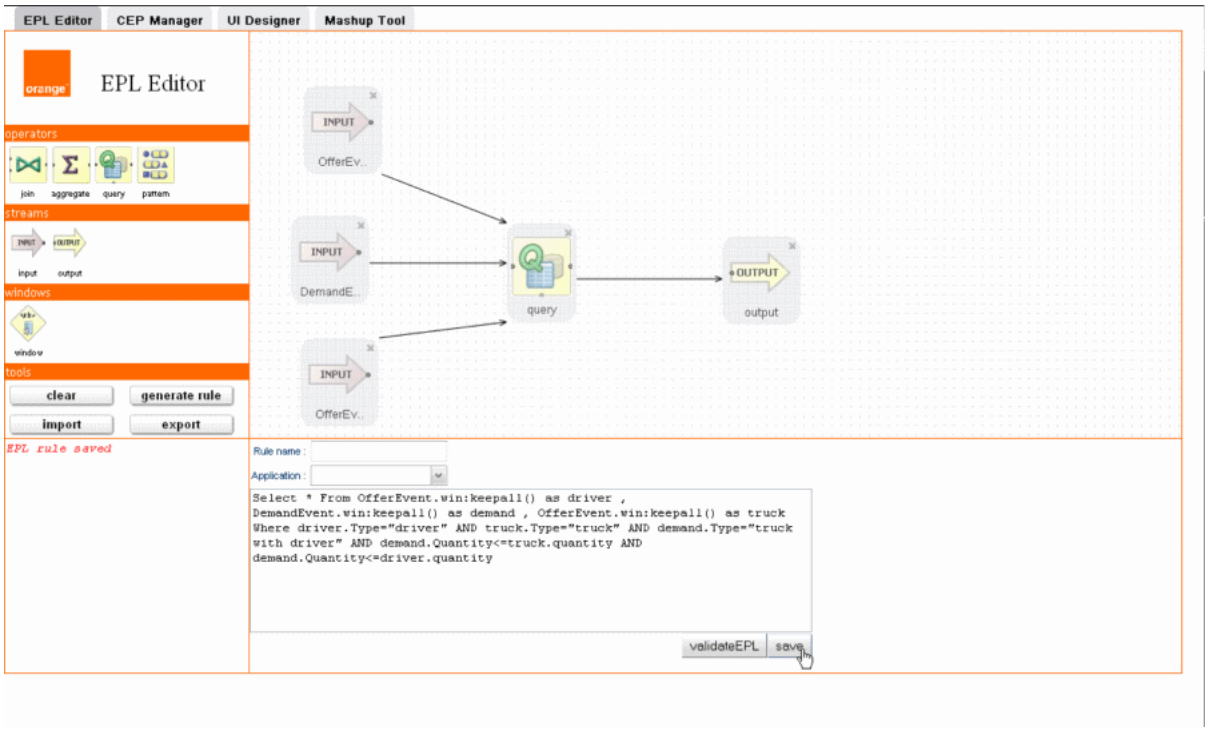
SOL/CEP implementation does not feature a Local Storage component.

7.4.3.5 *Graphical rules editor (EPL Editor)*

This optional component allows to graphically manage CEP rules, through the CEP Management API.

It allows graphical CEP rules building, by placing and biding blocks together. The resulting diagram is then converted into EPL syntax, which is understood by Esper4FastData.

The Graphical rules editor is useful to manage CEP rules in a user-friendly environment. It is not available for the SOL/CEP configuration.



Graphical rules editor

7.5 Basic concepts

7.5.1 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, an event is something notable that happens in the real world or in a system. But in the computer world, an event is understood as a computer representation of such an occurrence, usually for the purpose of computer processing. Events can contain data, are immutable, but more than one event may record the same activity.

Before the actual event processing occurs, optional event preprocessing can be implemented: it 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 CEP solution. Heterogeneous, distributed event processing applications normally require some type of event preprocessing for data normalization, validation and transformation.

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.

CEP also refers to state processing, which is the change of state occurring when a defined threshold is exceeded. It requires event monitoring, event reporting, event recording, and event filtering. Any 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 change usually comes with an attached time stamp

defining the order of occurrence, and a topology mark that can define the location of the occurrence.

Regarding the Data Handling GE, the choice for the reference implementation of the Complex Event Processor depends on the constraint level of the hardware that hosts and runs the enabler.

For constrained environment, SOL/CEP is the right choice, whereas for usual gateways, the Esper engine is appropriate.

7.5.2 Esper Complex Event Processing engine

Esper is a CEP library that is available for both Java and .Net. It features a dedicated CEP rule (EPL) language. The Java version has been chosen for the FI-WARE project.

It is an open-source software under the GNU General Public License GPL.

Esper CEP library is mainly used for:

- Complex computations: applications that detect event correlation, filter events, merge event streams, trigger specific actions when particular conditions are satisfied, 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

Native supported event formats are plain Java objects, XML, and map objects. XML is the one implemented in FI-WARE.

Esper provides an dedicated Event Processing Language (EPL) which allows rules definition. EPL is an SQL-like language, but is specifically optimized for dealing with high frequency event data.

The EPL language allows grouping, aggregating, sorting, filtering, merging, splitting and duplicating event streams, defining time and length sliding windows, matching event pattern, and so on...

Some important concepts drive the CEP Engine component structure:

- The event types describe the internal structure of incoming events
- The rules define the way event are processed. Rule triggering occurs when event data match the rule criteria.
- Sliding windows store event data in a First-In First-Out (FIFO) manner. They can be sized in length or duration.
- A listener is executed when a rule is triggered. A listener is always associated to a rule.

7.5.2.1 *EPL statement structure*

A typical statement looks like the following. The brackets indicate optional features.

```
[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]
```

7.5.2.2 *Examples of rules statement:*

- Simply select fields from a single event source:

```
INSERT INTO StoredTaxiPresenceEventType
SELECT phoneNumber,status
FROM PresenceEvent
WHERE userType='Drive'
```

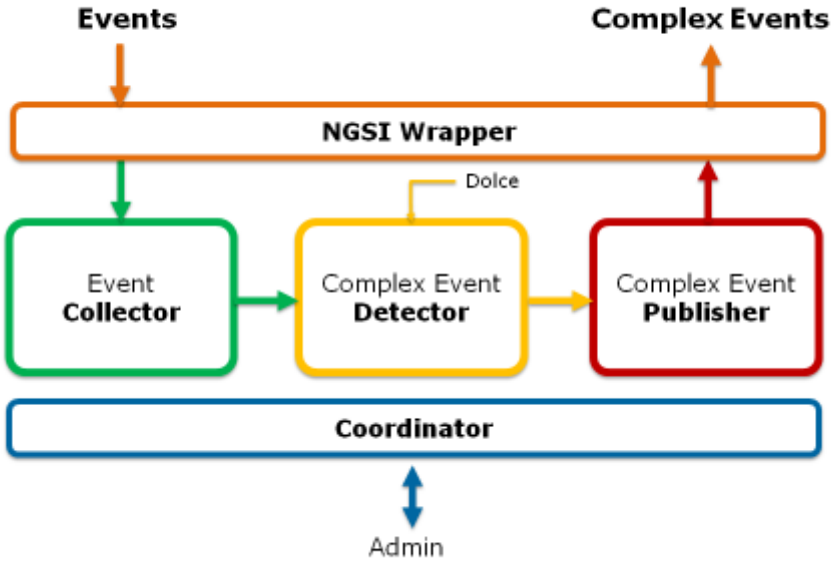
- Merge event sources:

```
INSERT INTO FraudEventType

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
```

7.5.3 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 the domain specific Dolce Language

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 adapters and processed. Likewise, Complex Events can be published towards different channels in a variety of formats.

For the FI-WARE Gateway Data Handling GE it will interact using NGSI.

7.5.3.1 *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

7.5.3.2 *Technical aspects*

The SOL/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).

7.5.3.3 *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 100 mm 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.

7.5.4 NGSI

NGSI Context Management specifications defined by OMA (Open Mobile Alliance)

NGSI 9/10 information model for FI-WARE

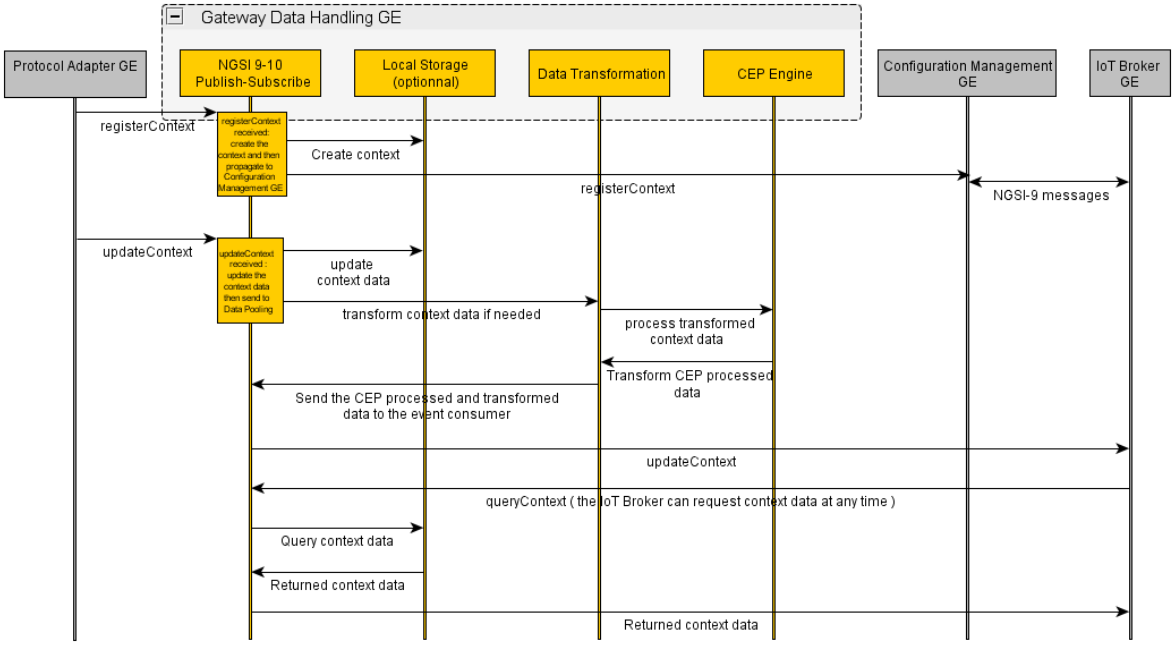
FI-WARE NGSI Open RESTful API Specification

7.5.5 ETSI M2M

[2] ETSI M2M communications

7.6 Main Interactions

- The NGSI 9-10 Publish-Subscribe component registers the Data Handling GE towards the Configuration Management GE by calling its *registerContext* method.
- The NGSI 9-10 Publish-Subscribe component sends events to the IoT Broker by calling its *updateContext* NGSI method.
- When needed, the IoT broker can call the *queryContext* method to retrieve context data from the Data Handling NGSI Publish-Subscribe component.
- The Protocol Adapter GE registers itself towards the Data Handling GE by calling its *registerContext* method.
- The Protocol Adapter GE sends events towards the Data Handling GE by calling its *updateContext* method.



Data Handling GE main interactions with surrounding GEs

7.7 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 resource 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.

7.8 Re-utilised Technologies/Specifications

The Gateway Data Handling GE is based on RESTful Design Principles. The technologies and specifications used in this GE are:

- RESTful web services
- HTTP/1.1 (RFC2616)
- XML data serialization format.

Other technology which is re-used

- Esper documentation

7.9 Detailed Open Specifications

Following is a list of Open Specifications linked to this Generic Enabler.

7.9.1 Open Restful API Specifications

Applicable only to the Esper4FastData configuration:

- Full Data Handling GE API

Applicable only to the SOL/CEP configuration:

- Administrative API for SOL/CEP (constrained environments)

Applicable to both Esper4FastData and SOL/CEP configurations:

- OMA NGSI-10
- OMA NGSI-9

7.9.2 Other Open Specifications

Non Applicable up to now.

7.10 Terms and definitions

This section comprises a summary of terms and definitions introduced during the previous sections. It intends to establish a vocabulary that will be help to carry out discussions internally and with third parties (e.g., Use Case projects in the EU FP7 Future Internet PPP). For a summary of terms and definitions managed at overall FI-WARE level, please refer to FIWARE Global Terms and Definitions.

- **Thing**. One instance of a physical object, living organism, person or concept interesting to a person or an application from the perspective of a FI-WARE instance, one particular usage area or to several usage areas. Examples of physical objects are: building, table, bridge (classes), the Kremlin, the tennis table from John's garden, the Tower bridge (instances). Examples of living organisms are: frog, tree, person (classes), the green frog from the garden, the oak tree in front of this building, Dr. Green.
- **Class of thing**. Defines the type of a thing in the style of object-oriented programming: a construct that is used as a blueprint to create instances, defining constituent members which enable class instances to have state and behavior. An example of class of thing is "person", whereas an instance of a thing is "Dr. Green", with all the static/dynamically changing properties of this particular person.
- **Group of things**. A physical/logical group of things. Examples are all office buildings from Munich, all bridges above the Thames, all screws from a drawer, the planes of Lufthansa currently above Germany, all cars inside a traffic jam driven by a female driver, the trees from the Amazonian forest, the squids from the Mediterranean see the mushrooms from Alice's garden, all the fish from the aquarium of John., the soccer team of Manchester, the colleagues from OrangeLabs currently working *abroad* (from the perspective of France), all patients above 60 years of age of Dr. Green, the traffic jams from Budapest, the wheat crop in France in the year 2011, the Research and Development Department of the company Telefónica.
- **Device**. Hardware entity, component or system that may be in a relationship with a *thing* or a *group of things* called *association*. A device has the means either to measure properties of a thing/group of things and convert it to an analog or digital signal that can be read by a program or user or to potentially influence the properties of a thing/group of things or both to measure/influence. In case when it can only measure the properties, then we call it a sensor. In

case when it can potentially influence the properties of a thing, we call it an actuator. Sensors and actuators may be elementary or composed from a set of elementary sensors/actuators. The simplest elementary sensor is a piece of hardware measuring a simple quantity, such as speed of the wind, and displaying it in a mechanical fashion. Sensors may be the combination of software and hardware components, such as a vehicle on-board unit, that consists of simple sensors measuring various quantities such as the speed of the car, fuel consumption etc. and a wireless module transmitting the measurement result to an application platform. The simplest elementary actuator is a light switch. We have emphasized *potentially* because as a result of the light switch being switched on it is not sure that the effect will be that light bulb goes on: only an *associated* sensor will be able to determine whether it went on or not. Other examples of devices are smart meters, mobile POS devices. More sophisticated devices may have a unique identifier, embedded computing capabilities and local data storage utilities, as well as embedded communication capabilities over short and/or long distances to communicate with IoT backend. Simple devices may not have all these capabilities and they can for instance only disclose the measurement results via mechanical means. In this latter case the further disclosure of the measurement result towards IoT backend requires another kind of device, called IoT Gateway.

- **Association**. It is a physical/logical relationship between a device and a thing or a device and a group of things or a group of devices and a group of things from the perspective of a FI-WARE instance, an application within a usage area project or other stakeholder. A device is associated with a thing if it can sense or (potentially) influence at least one property of the thing, property capturing an aspect of the thing interesting to a person or an application from the perspective of a FI-WARE instance, one particular usage area or to several usage areas. Devices are associated with things in *fully dynamic* or *mainly static* or *fully static* manner. The association may have several different embodiments: *physical attachment*, *physical embedding*, *physical neighborhood, logical relation* etc. Physical attachment means that the device is physically attached to the thing in order to monitor and interact with it, enabling the thing to be connected to the Internet. An example is the on-board device installed inside the vehicle to allow sending sensor data from the car to the FI-WARE instances. Physical embedding means that the device is deeply built inside the thing or almost part of the thing. An example of physical embedding is the GPS sensor inside a mobile phone. Physical neighborhood means that the device is in the physical neighborhood of the thing, but not in physical contact with it. An example of physical neighborhood is the association of the mobile phone of a subscriber who is caught in a traffic jam with the traffic jam itself: the mobile phone is the device, the traffic jam is the thing and the association means that the mobile phone is in the physical neighborhood of the area inside which the traffic jam is constrained (e.g. within 100 m from the region where the average speed of cars is below 5 km/h). Logical association means that there is a relationship between the device and the thing which is neither fully physical, in the sense of attachment or embedding, nor fully physical proximity related. An example of logical association is between the car and the garage door opener of the garage where the car usually parks during the night.
- **IoT gateway**. A device that additionally to or instead of sensing/actuating provides inter-networking and protocol conversion functionalities between devices and IoT backend potentially in any combination of these hosts 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. 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 devices such as the available computing, power, storage and energy consumption. The level of functional split between the IoT backend and the IoT gateway will also depend on the available resources on the IoT gateway, the cost and quality of connectivity and the desired level for the distribution of intelligence and service abstraction.

- **IoT resource**. Computational elements (software) that provide the technical means to perform sensing and/or actuation on the device. There may be one-to-one or one-to-many relationship between a device and its IoT resource. Actuation capabilities exposed by the IoT resource may comprise configuration of the management/application features of the device, such as connectivity, access control, information, while sensing may comprise the gathering of faults, performance metrics, accounting/administration data from the device, as well as application data about the properties of the thing with which the device is associated. The resource is usually hosted on the device.
- **Management service**. It is the feature of the IoT resource providing programmatic access to readable and/or writable data belonging to the functioning of the device, comprising a subset of the FCAPS categories, that is, configuration management, fault management, accounting /access management/administration, performance management/provisioning and security management. The extent of the subset depends on the usage area. Example of configuration management data is the IP endpoint of the IoT backend instance to which the device communicates. Example of fault management is an error given as a result of the overheating of the device because of extensive exposure to the sun. Example of accounting/administration is setting the link quota for applications using data from a certain sensor. Example of security management is the provisioning of a list of device ID-s of peer devices with which the device may directly communicate.
- **Application service**. It is the feature of the IoT resource providing programmatic access to readable or writable data in connection with the thing which is associated with the device hosting the resource. The application service exchanges application data with another device (including IoT gateway) and/or the IoT backend. Measured sensory data are example of application data flowing from devices to sensors. Setting the steering angle of a security camera or sending a “start wetting” command to the irrigation system are examples of application data flowing from the application towards the sensor.
- **Event**. An event can be defined as an activity that happens, occurs in a device, gateway, IoT backend or is created by a software component inside the IoT service enablement. The digital representation of this activity in a device, IoT resource, IoT gateway or IoT backend instance, or more generally, in a FI-WARE instance, is also called an event. Events may be simple and complex. A simple event is an event that is not an abstraction or composition of other events. An example of a simple event is that “smart meter X got broken”. A complex event is an abstraction of other events called member events. For example a stock market crash or a cellular network blackout is an abstraction denoting many thousand of member events. In many cases a complex event references the set of its members, the implication being that the event contains a reference. For example, the cellular network blackout may be caused by a power failure of the air conditioning in the operator’s data center. In other cases such a reference may not exist. For example there is no accepted agreement as to which events are members of a stock market crash complex event. Complex events may be created of simple events or other complex events by

an IoT resource or the IoT backend instance.

- **IoT Backend.** Provides management functionalities for the devices and IoT domain-specific support for the applications. Integrant component of FI-WARE instances.
- **IoT application**. An application that uses the application programming interface of the IoT service enablement component. May have parts running on one/set of devices, one/set of IoT gateways and one/set of IoT backends.
- **Virtual thing**. It is the digital representation of a thing inside the IoT service enablement component. Consists of a set of properties which are interesting to a person or an application from the perspective of a FI-WARE instance, one particular usage area or to several usage areas. Classes of things have the same set of properties, only the value of the properties change from instance to instance.

8 FIWARE OpenSpecification IoT Gateway ProtocolAdapter

Name	FIWARE.OpenSpecification.IoT.Gateway.ProtocolAdapter		
Chapter	IoT Services Enablement,		
Catalogue-Link to Implementation	Gateway Protocol Adapter - ZPA	**Owner**	Orange, Telecom Italia, SAP,

8.1 Preface

Within this document you find a self-contained open specification of a FI-WARE generic enabler, please consult as well the FI-WARE_Product_Vision, the website on http://www.fi-ware.eu and similar pages in order to understand the complete context of the FI-WARE project.

8.2 Copyright

8.3 Legal Notice

8.4 Overview

The Protocol Adapter GE deals with the incoming and outgoing traffic and messages between the IoT Gateway and registered devices, to be served by either the Gateway Device Management GE or the Data Handling GE. There may be multiple instances of Protocol Adapter GEs capable of serving not fully IoT compliant devices, i.e. devices that do not support ETSI M2M (the specifications may be found at the following link ETSI M2M Latest Drafts). These devices can be IP-based devices, that communicates using the IP stack (IPv4 or IPv6), or "legacy devices", meaning devices communicating using non-IP based protocols, for instance ZigBee, or Z-Wave.

The Protocol Adapter GE receives these device specific protocols and translates them to a uniform internal API. The exposed API handles 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 ZigBee PAN(s) (Personal Area Network). It supports a mechanism whereby a 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.

8.5 Basic Concepts

An overview of the Protocol Adapter GE is provided below, and is followed by an identification of the interfaces.

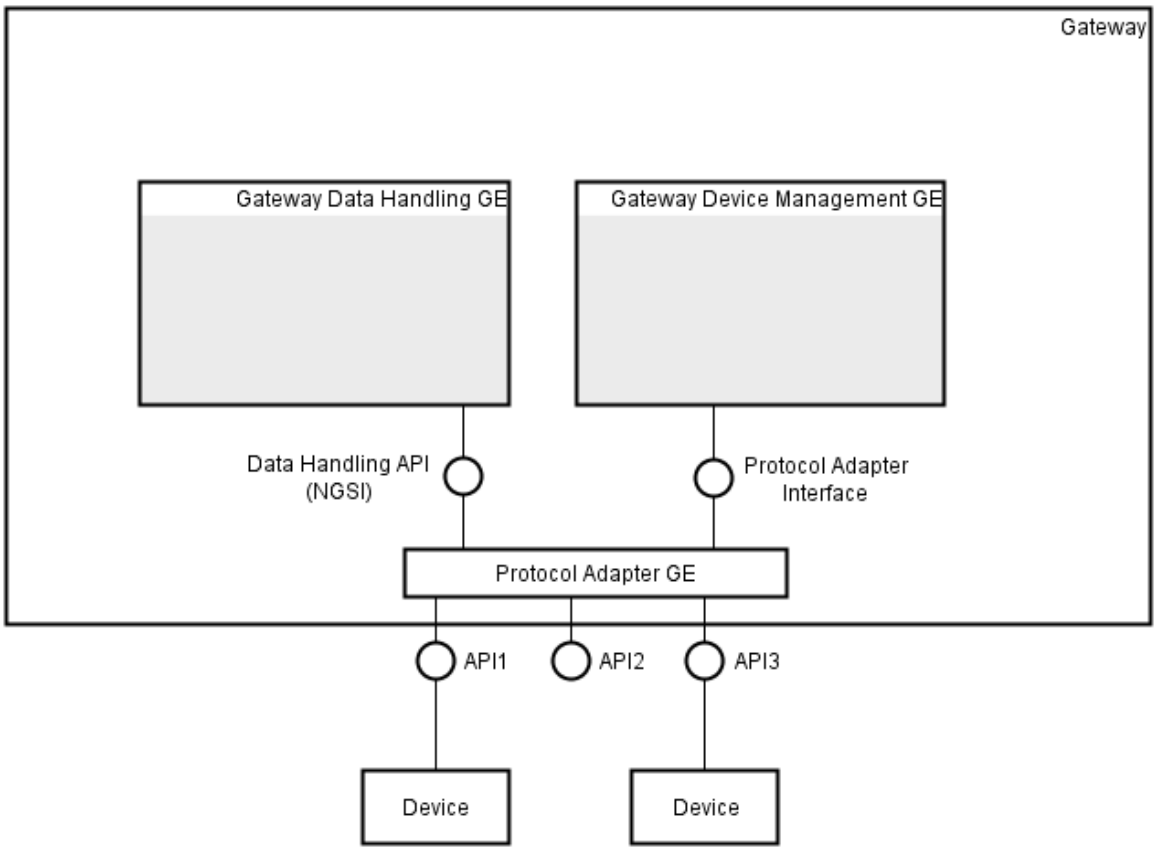


Figure 1: IoT Gateway Architecture

There are three interfaces to the Protocol Adapter GE. The southbound APIs provide the gateway external interface to non-ETSI M2M devices hosting sensor and actuator resources. The currently supported interfaces are IETF CoRE (provided by Ericsson and no more supported) and ZigBee (the specifications may be found at the following link ZigBee Network Devices Standard Overview).

On the northbound side there are two interfaces:

- the first interface is the Protocol Adapter Interface that is a communications protocol to the Gateway Device Management GE, based on the Generic Device Access API. 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 their resources.
- the second one is the Gateway Data Handling API, NGSI compliant, that allows to interact with the Gateway Data Handling GE. Through this interface, device events and data are published towards the Gateway Data Handling GE.

Figure 2 introduces the architecture of the FI-WARE implementation of the Protocol Adapter GE.

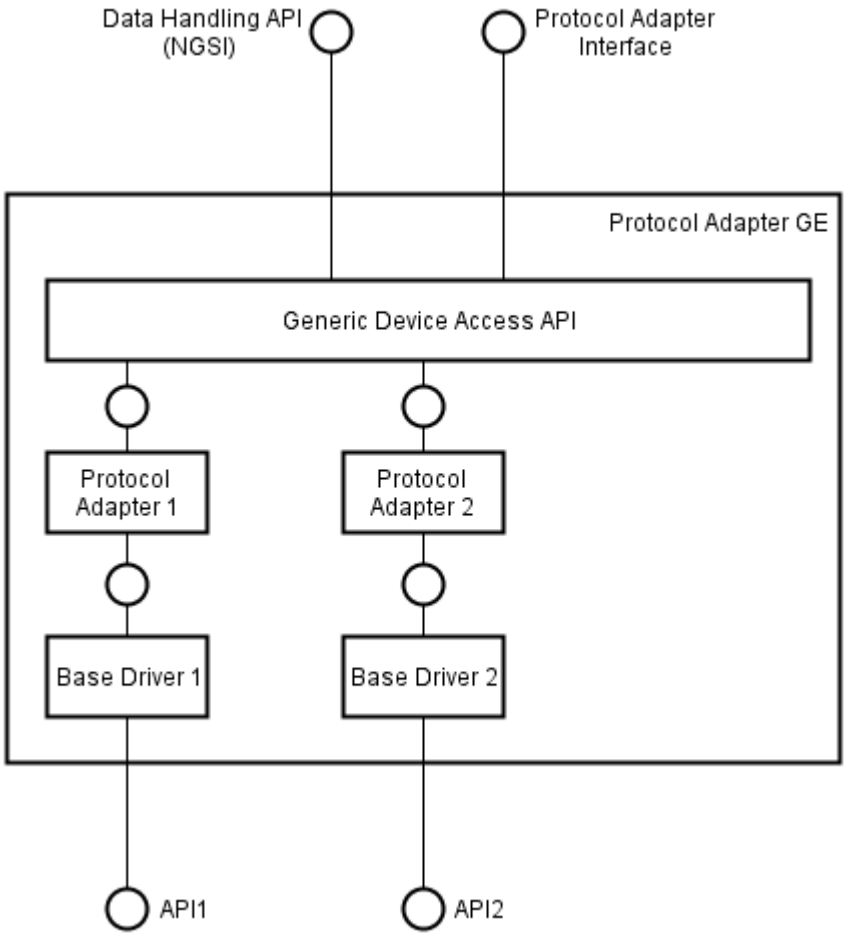


Figure 2: Protocol Adapter GE internal architecture

8.5.1.1 *Base Driver*

The Base Driver is the low-level API for legacy devices (i.e. an implementation of the device specific protocol stack). Base Drivers handle device discovery and access to sensor and actuator resources in a protocol specific way.

For instance, the ZigBee Base Driver is based on the Network Device Gateway Specification defined by the ZigBee Alliance (ZigBee document 075468r35 may be found at this link ZigBee Network Devices Standard Overview). 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

8.5.1.2 *Protocol Adaptation*

A Protocol Adapter is the glue between a base driver and the Generic Device Access API or the Gateway Data Handling API. Via the base driver, it discovers devices, tracks events occurred on them and executes commands to actuate them. Therefore, a Protocol Adapter is necessary for each protocol that the Base Drivers support. Whether the protocol is standardized or proprietary does not matter. As far as the Base Driver is available and the Protocol Adapter is implemented on top, the Generic Device Access API or the Gateway Data Handling API are able to support the protocol and provide a unified way to access devices with the protocol. The Protocol Adapter is 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 measurement 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 the device service that is registered in the Protocol Adapter framework.
- Device event and data. Its behavior is like an Event Producer and for each device connected it shall send events and data

8.5.1.3 *Generic Device Access API*

The Generic Device Access API (GDA) exposes a high-level, protocol agnostic API towards the Gateway Data Handling and Gateway Device Management. GDA uses service schemas which are XML-files that describe the supported resources, i.e. the application profiles. This schema based approach makes it possible to cover a wide range of applications spanning from home automation, to media, to health care.

The GDA defines two main data structures:

- Device: it represents the sensor/actuator,
- Service: it represents a set of functionalities provided by the Device.

The GDA main methods, which has to be used by the Gateway Data Handling and Gateway Device Management are:

- device.getService(<name of service>) – used to get the services associated to a specific device,
- service.getProperties() – used to get the list of properties (i.e. attributes) implemented by a specific service implementation,
- service.getAction(<name of action>) – used to get a single action (i.e. command) implemented by a specific service implementation.

Gateway Data Handling and Gateway Device Management can get sensor data or configure a device by reading/writing a Service property, and can command an actuator by calling a Service action.

8.6 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 for new devices with the gateway framework (OSGi service bus). 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.

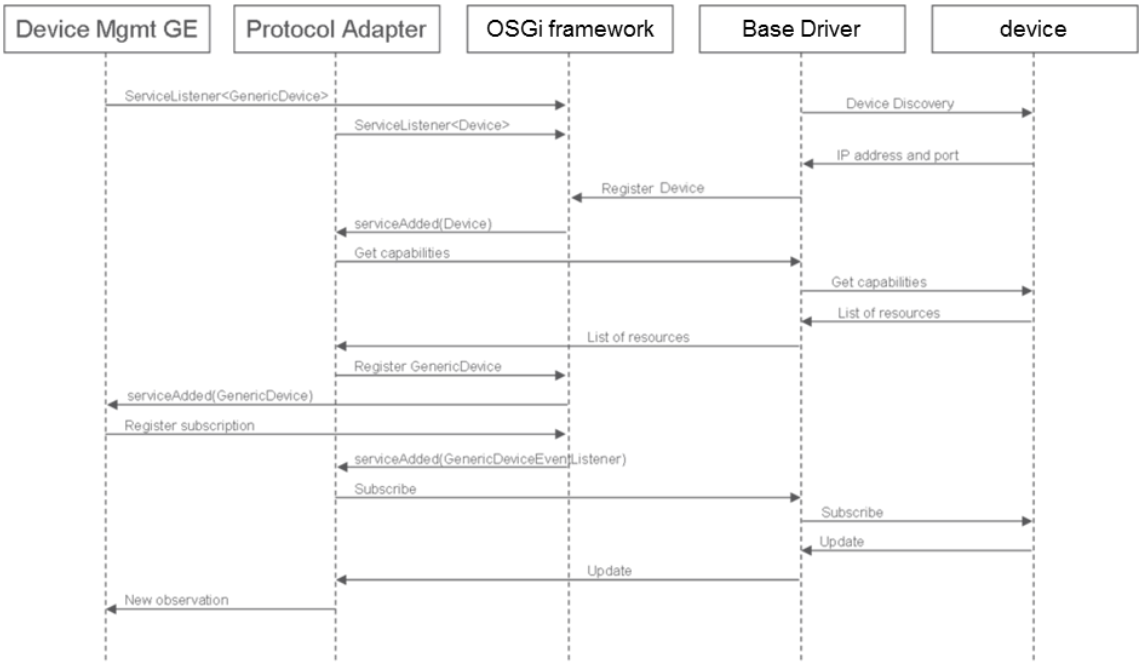


Figure 3: Device discovery and subscribe

Moreover the Protocol Adapter GE is an NGSI compliant Event Producer towards the Data Handling GE. In the IoT Service Enablement architecture, the Protocol Adapter GE publishes device events and data towards the Data Handling GE. In this setup, the Protocol Adapter GE registers itself as an NGSI Context Provider, by calling the NGSI-9 *registerContext* method exposed by the Data Handling GE. After this context registration, the Protocol Adapter GE sends events, related to the connected devices, by calling the NGSI-10 *updateContext* method of the Data Handling GE.

8.7 Basic Design Principles

Projects deciding to implement support for additional protocols should make sure that their implementations provide the following functionality:

- Device discovery
- Device measurement update
- Device actuation
- Device events and data support

8.8 Re-utilised Technologies/Specifications

The Gateway Protocol Adpater GE is based on RESTful Design Principles. The technologies and specifications used in this GE are:

- RESTful web services
- HTTP/1.1 (RFC2616)
- XML data serialization format.

8.9 Terms and definitions

This section comprises a summary of terms and definitions introduced during the previous sections. It intends to establish a vocabulary that will be help to carry out discussions internally and with third parties (e.g., Use Case projects in the EU FP7 Future Internet PPP). For a summary of terms and definitions managed at overall FI-WARE level, please refer to FIWARE Global Terms and Definitions

- **Thing**. One instance of a physical object, living organism, person or concept interesting to a person or an application from the perspective of a FI-WARE instance, one particular usage area or to several usage areas. Examples of physical objects are: building, table, bridge (classes), the Kremlin, the tennis table from John's garden, the Tower bridge (instances). Examples of living organisms are: frog, tree, person (classes), the green frog from the garden, the oak tree in front of this building, Dr. Green.
- **Class of thing**. Defines the type of a thing in the style of object-oriented programming: a construct that is used as a blueprint to create instances, defining constituent members which enable class instances to have state and behavior. An example of class of thing is "person", whereas an instance of a thing is "Dr. Green", with all the static/dynamically changing properties of this particular person.
- **Group of things**. A physical/logical group of things. Examples are all office buildings from Munich, all bridges above the Thames, all screws from a drawer, the planes of Lufthansa currently above Germany, all cars inside a traffic jam driven by a female driver, the trees from the Amazonian forest, the squids from the Mediterranean see the mushrooms from Alice's garden, all the fish from the aquarium of John., the soccer team of Manchester, the colleagues from OrangeLabs currently working *abroad* (from the perspective of France), all patients above 60 years of age of Dr. Green, the traffic jams from Budapest, the wheat crop in France in the year 2011, the Research and Development Department of the company Telefónica.
- **Device**. Hardware entity, component or system that may be in a relationship with a *thing* or a *group of things* called *association*. A device has the means

either to measure properties of a thing/group of things and convert it to an analog or digital signal that can be read by a program or user or to potentially influence the properties of a thing/group of things or both to measure/influence. In case when it can only measure the properties, then we call it a sensor. In case when it can potentially influence the properties of a thing, we call it an actuator. Sensors and actuators may be elementary or composed from a set of elementary sensors/actuators. The simplest elementary sensor is a piece of hardware measuring a simple quantity, such as speed of the wind, and displaying it in a mechanical fashion. Sensors may be the combination of software and hardware components, such as a vehicle on-board unit, that consists of simple sensors measuring various quantities such as the speed of the car, fuel consumption etc and a wireless module transmitting the measurement result to an application platform. The simplest elementary actuator is a light switch. We have emphasized *potentially* because as a result of the light switch being switched on it is not sure that the effect will be that light bulb goes on: only an *associated* sensor will be able to determine whether it went on or not. Other examples of devices are smart meters, mobile POS devices. More sophisticated devices may have a unique identifier, embedded computing capabilities and local data storage utilities, as well as embedded communication capabilities over short and/or long distances to communicate with IoT backend. Simple devices may not have all these capabilities and they can for instance only disclose the measurement results via mechanical means. In this latter case the further disclosure of the measurement result towards IoT backend requires another kind of device, called IoT Gateway.

- **Association**. It is a physical/logical relationship between a device and a thing or a device and a group of things or a group of devices and a group of things from the perspective of a FI-WARE instance, an application within a usage area project or other stakeholder. A device is associated with a thing if it can sense or (potentially) influence at least one property of the thing, property capturing an aspect of the thing interesting to a person or an application from the perspective of a FI-WARE instance, one particular usage area or to several usage areas. Devices are associated with things in *fully dynamic* or *mainly static* or *fully static* manner. The association may have several different embodiments: *physical attachment*, *physical embedding*, *physical neighborhood, logical relation* etc. Physical attachment means that the device is physically attached to the thing in order to monitor and interact with it, enabling the thing to be connected to the Internet. An example is the on-board device installed inside the vehicle to allow sending sensor data from the car to the FI-WARE instances. Physical embedding means that the device is deeply built inside the thing or almost part of the thing. An example of physical embedding is the GPS sensor inside a mobile phone. Physical neighborhood means that the device is in the physical neighborhood of the thing, but not in physical contact with it. An example of physical neighborhood is the association of the mobile phone of a subscriber who is caught in a traffic jam with the traffic jam itself: the mobile phone is the device, the traffic jam is the thing and the association means that the mobile phone is in the physical neighborhood of the area inside which the traffic jam is constrained (e.g. within 100 m from the region where the average speed of cars is below 5 km/h). Logical association means that there is a relationship between the device and the thing which is neither fully physical, in the sense of attachment or embedding, nor fully physical proximity related. An example of logical association is between the car and the garage door opener of the garage where the car usually parks during the night.
- **IoT gateway**. A device that additionally to or instead of sensing/actuating provides inter-networking and protocol conversion functionalities between

devices and IoT backend potentially in any combination of these hosts 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. 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 devices such as the available computing, power, storage and energy consumption. The level of functional split between the IoT backend and the IoT gateway will also depend on the available resources on the IoT gateway, the cost and quality of connectivity and the desired level for the distribution of intelligence and service abstraction.

- **IoT resource**. Computational elements (software) that provide the technical means to perform sensing and/or actuation on the device. There may be one-to-one or one-to-many relationship between a device and its IoT resource. Actuation capabilities exposed by the IoT resource may comprise configuration of the management/application features of the device, such as connectivity, access control, information, while sensing may comprise the gathering of faults, performance metrics, accounting/administration data from the device, as well as application data about the properties of the thing with which the device is associated. The resource is usually hosted on the device.
- **Management service**. It is the feature of the IoT resource providing programmatic access to readable and/or writable data belonging to the functioning of the device, comprising a subset of the FCAPS categories, that is, configuration management, fault management, accounting /access management/administration, performance management/provisioning and security management. The extent of the subset depends on the usage area. Example of configuration management data is the IP endpoint of the IoT backend instance to which the device communicates. Example of fault management is an error given as a result of the overheating of the device because of extensive exposure to the sun. Example of accounting/administration is setting the link quota for applications using data from a certain sensor. Example of security management is the provisioning of a list of device ID-s of peer devices with which the device may directly communicate.
- **Application service**. It is the feature of the IoT resource providing programmatic access to readable or writable data in connection with the thing which is associated with the device hosting the resource. The application service exchanges application data with another device (including IoT gateway) and/or the IoT backend. Measured sensory data are example of application data flowing from devices to sensors. Setting the steering angle of a security camera or sending a “start wetting” command to the irrigation system are examples of application data flowing from the application towards the sensor.
- **Event**. An event can be defined as an activity that happens, occurs in a device, gateway, IoT backend or is created by a software component inside the IoT service enablement. The digital representation of this activity in a device, IoT resource, IoT gateway or IoT backend instance, or more generally, in a FI-WARE instance, is also called an event. Events may be simple and complex. A simple event is an event that is not an abstraction or composition of other events. An example of a simple event is that “smart meter X got broken”. A complex event is an abstraction of other events called member events. For example a stock market crash or a cellular network blackout is an abstraction denoting many thousand of member events. In many cases a complex event references the set of its members, the implication being that the event contains a reference. For example, the cellular network blackout may be caused by a

power failure of the air conditioning in the operator's data center. In other cases such a reference may not exist. For example there is no accepted agreement as to which events are members of a stock market crash complex event. Complex events may be created of simple events or other complex events by an IoT resource or the IoT backend instance.

- **IoT Backend.** Provides management functionalities for the devices and IoT domain-specific support for the applications. Integrant component of FI-WARE instances.
- **IoT application**. An application that uses the application programming interface of the IoT service enablement component. May have parts running on one/set of devices, one/set of IoT gateways and one/set of IoT backends.
- **Virtual thing**. It is the digital representation of a thing inside the IoT service enablement component. Consists of a set of properties which are interesting to a person or an application from the perspective of a FI-WARE instance, one particular usage area or to several usage areas. Classes of things have the same set of properties, only the value of the properties change from instance to instance.

9 FI-WARE NGSI Open RESTful API Specification

In compliance with OMA NGSI specifications, FI-WARE NGSI specifications comprise the interface of related interface specifications:

- FI-WARE NGSI-9 Open RESTful API Specification
- FI-WARE NGSI-10 Open RESTful API Specification
- NGSI association
- FI-WARE NGSI: publicly available documents

9.1 NGSI 9/10 information model

This page outlines the information model of OMA NGSI 9 and OMA NGSI 10. More detail can be found in the OMA specification document.

9.1.1 Central Concepts

9.1.1.1 *Entities*

The central aspect of the NGSI 9/10 information model is the concept of **entity**. Entities are the virtual representation of all kinds of physical objects in the real world. Examples for physical entities are tables, rooms, or persons. Virtual entities have an identifier and a type. For example, a virtual entity representing a person named “John” could have the identifier “John” and the type “person”.

9.1.1.2 *Attributes*

Any available information about physical entities is expressed in the form of **attributes** of virtual entities. Attributes have a name and a type as well. For example, the body temperature of John would be represented as an attribute having the name “body_temperature” and the type “temperature”. Values of such attributes are contained in value containers. This kind of container does not only consist of the actual attribute value, but also contains a set of metadata. Metadata is data about data; in the body temperature example, this metadata could represent the time of measurement, the measurement unit, and other information about the attribute value.

9.1.1.3 *Attribute Domains*

There also is a concept of **attribute domains** in OMA NGSI 9/10. An attribute domain logically groups together a set of attributes. For example, the attribute domain "health_status" could be comprised of the attributes "body_temperature" and "blood_pressure".

9.1.1.4 *Context Elements*

The container used for exchanging information about entities is the '*context element'*. A context element can contain information about multiple attributes of one entity. The domain of these attributes can also be specified inside the context element; in this case all provided attribute values have to belong to that domain.

Formally, a context element contains the following information:

- an entity ID including the name and the type of the entity
- a list of attributes
- (optionally) the name of an attribute domain
- (optionally) a list of metadata that apply to all attribute values of the given domain

9.2 FI-WARE NGSI-9 Open RESTful API Specification

9.2.1 Introduction to the FI-WARE NGSI-9 API

9.2.1.1 *FI-WARE NGSI-9 API Core*

The FI-WARE version of the OMA NGSI-9 interface is a RESTful API via HTTP. Its purpose is to exchange information about the availability of context information. The three main interaction types are

- one-time queries for discovering hosts (agents) where certain context information is available
- subscriptions for context availability information updates (and the corresponding notifications)
- registration of context information, i.e. announcements that certain context information is available (invoked by context providers)

9.2.2 Intended Audience

This guide is intended for both developers of GE implementations and IoT Application programmers. For the former, this document specifies the API that has to be implemented in order to ensure interoperability with other GEs from the IoT Chapter of FI-WARE and the Publish/Subscribe Broker GE. For the latter, this document describes how to assemble instances of the FI-WARE IoT Platform.

Prerequisites: Throughout this specification it is assumed that the reader is familiar with:

- ReSTful web services
- HTTP/1.1
- XML data serialization formats

We also refer the reader to the NGSI-9/NGSI-10 specification and binding documents for details on the resource structure and message formats.

9.2.3 Change history

This version of the FI-WARE NGSI-9 Open RESTful API Specification replaces and obsoletes all previous versions. The most recent changes are described in the table below:

Revision Date	Changes Summary
July 14, 2012	• 1st stable version

9.2.4 Additional Resources

This document is to be considered as a guide to the NGSI-9 API. The formal specification of NGSI-9 can be downloaded from the website of the Open Mobile Alliance.

The RESTful binding of OMA NGSI-9 described on this page has been defined by the FI-WARE project. It can be accessed in the [1]. Note that also the schema files are part of the binding.

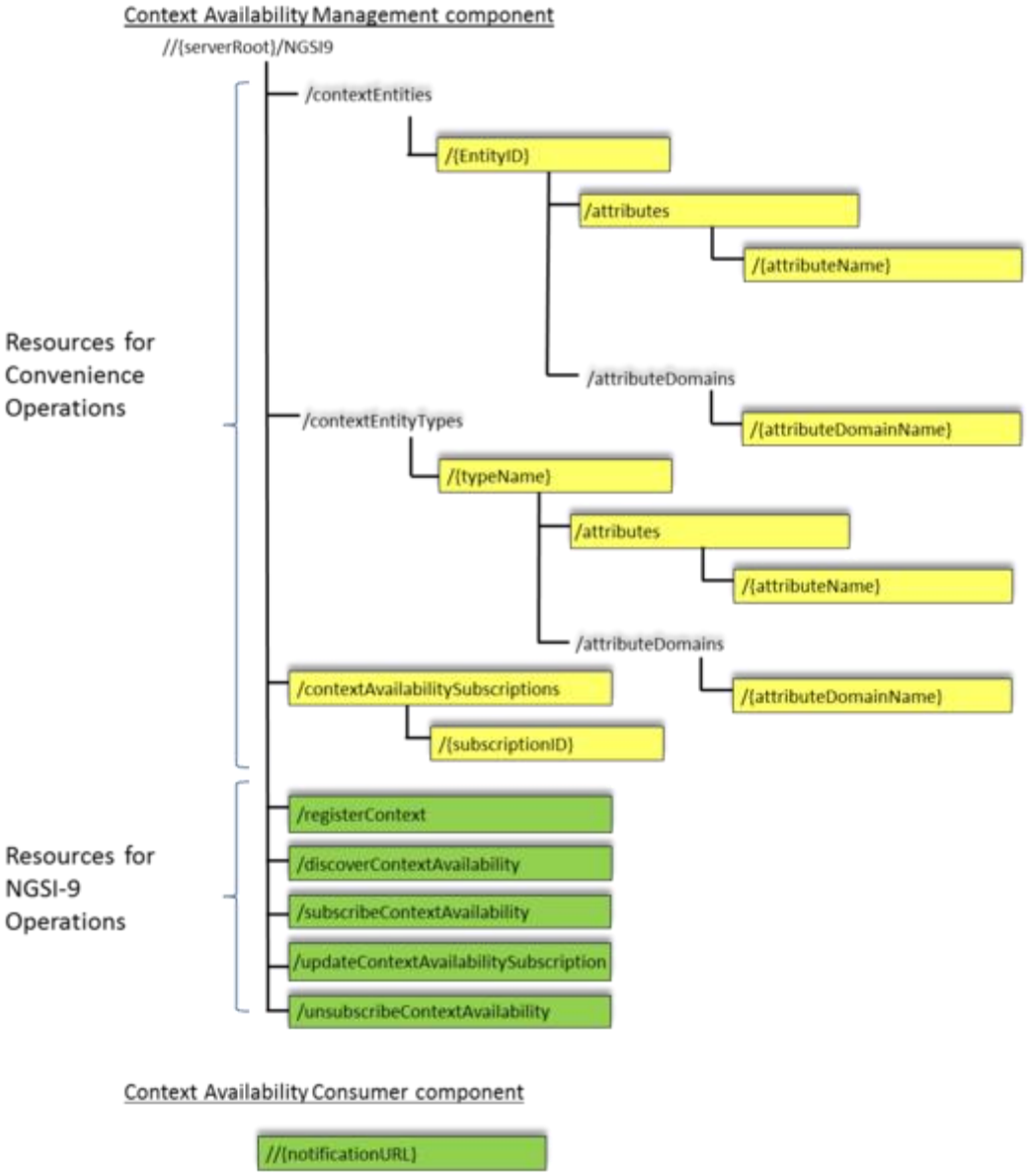
OMA NGSI-10 and OMA NGSI-9 share the same NGSI-9/NGSI-10 information model. Be sure to have read it before continuing on this page.

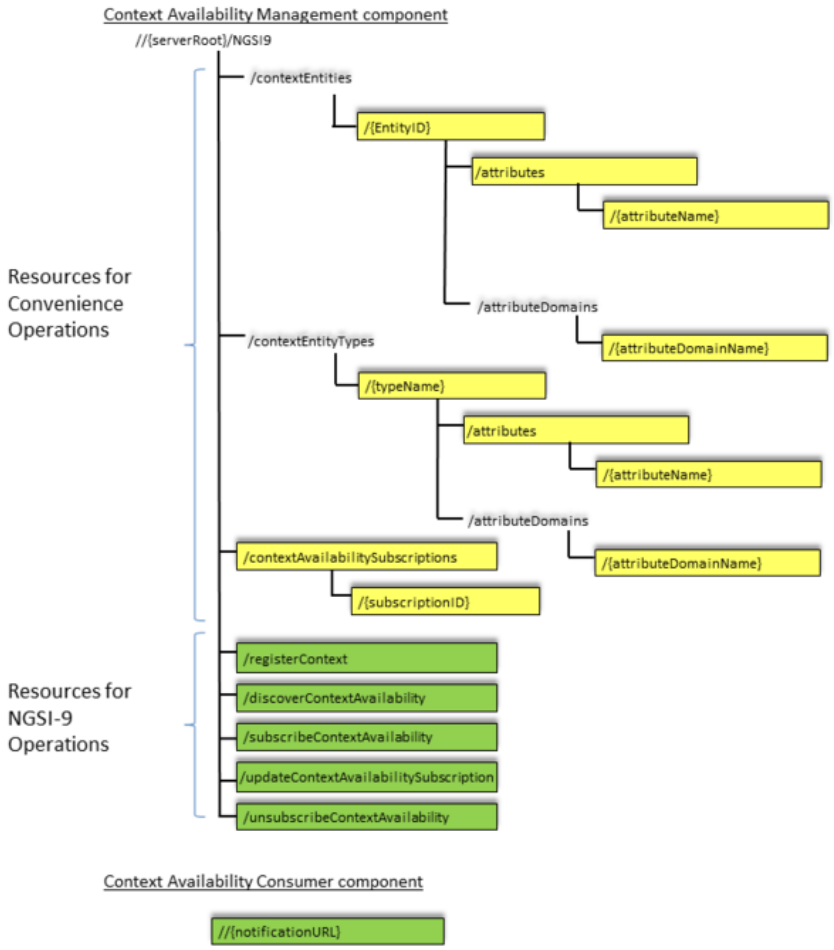
9.2.5 Legal Notice

Please check the following FI-WARE Open Specification Legal Notice (implicit patents license) and FI-WARE Open Specification Legal Notice (essential patents license) to understand the rights to use these specifications.

9.3 General NGSI-9 API information

9.3.1 Resources Summary


Context Availability Management component
//{serverRoot}/NGSI9
/contextEntities
/{EntityID}
/attributes
/{attributeName}
/attributeDomains
/{attributeDomainName}
/contextEntityTypes
/{typeName}
/attributes
/{attributeName}
/attributeDomains
/{attributeDomainName}
/contextAvailabilitySubscriptions
/{subscriptionID}
Resources for Convenience Operations
/registerContext
/discoverContextAvailability
/subscribeContextAvailability
/updateContextAvailabilitySubscription
/unsubscribeContextAvailability
Resources for NGSI-9 Operations
Context Availability Consumer component
//{notificationURL}



The mapping of NGSI-9 functionality to a resource tree (see figure above) follows a twofold approach. On the one hand, there is one resource per NGSI-9 operation which supports the respective functionality by providing a POST operation (colored green in the picture). On the other hand, a number of additional resources support convenience functionality (colored yellow). The latter resource structure more closely follows the REST approach and typically supports more operations (GET PUT, POST, and DELETE).

The convenience functions typically only support a subset of the functionality of the corresponding NGSI operations. Nevertheless, they enable simpler and more straightforward access. All data structures, as well as the input and output messages are represented by xml types. The definition of these types can be found in the xml schema files.

9.3.2 Representation Format

The NGSI-9 API supports only XML as data serialization format.

9.3.3 Representation Transport

Resource representation is transmitted between client and server by using HTTP 1.1 protocol, as defined by IETF RFC-2616. Each time an HTTP request contains payload, a Content-Type header shall be used to specify the MIME type of wrapped

representation. In addition, both client and server may use as many HTTP headers as they consider necessary.

9.3.4 API Operations on Context Management Component

9.3.4.1 *Standard NGSI-9 Operation Resources*

The five resources listed in the table below represent the five operations offered by systems that implement the NGSI-9 Context Management role. Each of these resources allows interaction via http POST. All attempts to interact by other verbs shall result in an HTTP error status 405; the server should then also include the 'Allow: POST' field in the response.

Resource	**Base URI: http://{serverRoot}/NGSI9**	**HTTP verbs**
		POST
Context Registration Resource	/registerContext	Generic context registration. The expected request body is an instance of registerContextRequest; the response body is an instance of registerContextResponse.
Discovery resource	/discoverContextAvailability	Generic discovery of context information providers. The expected request body is an instance of discoverContextAvailabilityRequest; the response body is an instance of discoverContextAvailabilityResponse.
Availability subscription resource	/subscribeContextAvailability	Generic subscription to context availability information. The expected request body is an instance of subscribeContextAvailabilityRequest; the response body is an instance of subscribeContextAvailabilityResponse.
Availability subscription update resource	/updateContextAvailabilitySubscription	Generic update of context availability subscriptions. The expected request body is an instance of updateContextAvailabilitySubscriptionRequest; the response body is an instance of updateContextAvailabilitySubscriptionResponse.
Availability subscription deletion	/unsubscribeContextAvailability	Generic deletion of context availability subscriptions. The expected request body is an instance of

resource		unsubscribeContextAvailabilityRequest; the response body is an instance of unsubscribeContextAvailabilityResponse.

9.3.4.2 *Convenience Operation Resources*

The table below gives an overview of the resources for convenience operation and the effects of interacting with them via the standard HTTP verbs GET, PUT, POST, and DELETE.

Resource	**Base URI: http://{serverRoot}/NGSI9**	**HTTP verbs**			
		GET	**PUT**	**POST**	**DELETE**
Individual context entity	/contextEntities/{EntityID}	Retrieve information on providers of any information about the context entity	-	Register a provider of information about the entity	
Attribute container of individual context entity	/contextEntities/{EntityID}/attributes	Retrieve information on providers of any information about the context entity	-	Register a provider of information about the entity	
Attribute of individual context entity	/contextEntities/{EntityID}/attributes/{attributeName}	Retrieve information on providers of the attribute value	-	Register a provider of information about the attribute	
Attribute domain of individual context entity	/contextEntities/{EntityID}/attributeDomains/{attributeDomainName}	Retrieve information on providers of information about attribute	-	Register a provider of information about attributes from the	

		values from the domain		domain	
Context entity type	/contextEntityTypes/{tyeName}	Retrieve information on providers of any information about context entities of the type	-	Register a provider of information about context entitie of the type	
Attribute container of entity type	/contextEntityTypes/{typeName}/attributes	Retrieve information on providers of any information about context entities of the type	-	Register a provider of information about context entitie of the type	
Attribute of entity type	/contextEntityTypes/{typeName}/attributes/{attributeName}	Retrieve information on providers of values of this attribute of context entities of the type	-	Register a provider of information about this attribute of context entities of the type	
Attribute domain of entity type	/contextEntityTypes/{typeName}/attributeDomains/{attributeDomainName}	Retrieve information on providers of attribute values belonging to the specific domain, where the entity is of the specific type	-	Register a provider of information about attributes belonging to the specific domain, where the entity is of the specific type	
Availability	/contextAvailabilitySubscript	-	-	Create a	

subscription container	ions			new availability subscription	
Availability subscription	/contextAvailabilitySubscriptions/{subscriptionID}	-	Update subscription	-	Cancel subscription

9.3.5 API operation on Context Consumer Component

This section describes the resource that has to be provided by the context consumer in order to receive availability notifications. All attempts to interact with it by other verbs than POST shall result in an HTTP error status 405; the server should then also include the 'Allow: POST' field in the response.

Resource	URI	HTTP verbs
		POST
Notify context resource	//{notificationURI}	Generic availability notification.The expected request body is an instance of notifyContextAvailabilityRequest; the response body is an instance of notifyContextAvailabilityResponse.

9.4 FI-WARE NGSI-10 Open RESTful API Specification

9.4.1 Introduction to the FI-WARE NGSI 10 API

Please check the following FI-WARE Open Specification Legal Notice (essential patents license) to understand the rights to use these specifications.

9.4.1.1 *FI-WARE NGSI 10 API Core*

The FI-WARE version of the OMA NGSI 10 interface is a RESTful API via HTTP. Its purpose is to exchange context information. The three main interaction types are:

- one-time queries for context information
- subscriptions for context information updates (and the corresponding notifications)
- unsolicited updates (invoked by context providers)

9.4.1.2 *Intended Audience*

This guide is intended for both developers of GE implementations and IoT Application programmers. For the former, this document specifies the API that has to be implemented in order to ensure interoperability with other GEs from the IoT Chapter of FI-WARE and the Publish/Subscribe Broker GE. For the latter, this document describes how to assemble instances of the FI-WARE IoT Platform.

Prerequisites: Throughout this specification it is assumed that the reader is familiar with:

- ReSTful web services
- HTTP/1.1
- XML data serialization formats

We also refer the reader to the NGSI-9/10 specification and binding documents for details on the resource structure and message formats.

9.4.1.3 *Change history*

This version of the FI-WARE NGSI-10 Open RESTful API Specification replaces and obsoletes all previous versions. The most recent changes are described in the table below:

Revision Date	Changes Summary
May 14, 2012	• 1st stable version

9.4.1.4 *Additional Resources*

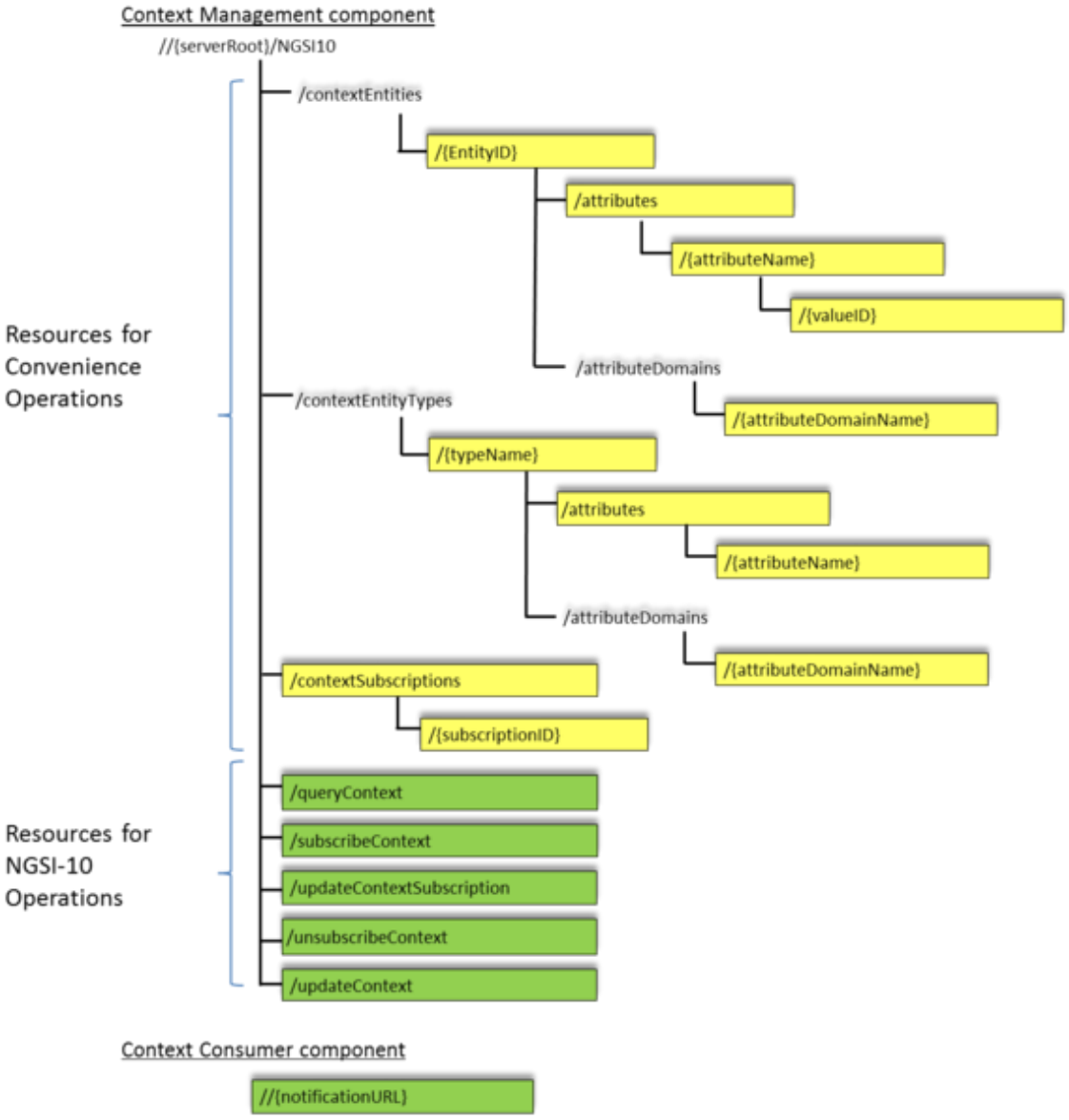
The formal specification of OMA NGSI 10 can be downloaded from the website of the Open Mobile Alliance.

The FI-WARE RESTful binding of OMA NGSI-10 described on this page has been defined by the FI-WARE project. It can be accessed in the svn. Note that also the XML schema files are part of the binding.

FI-WARE NGSI-10 and FI-WARE NGSI-9 share the same NGSI-9/10 information model. Be sure to have read it before continuing on this page.

9.4.2 General NGSI 10 API information

9.4.2.1 *Resources Summary*



The mapping of NGSI-10 functionality to a resource tree (see figure above) follows a twofold approach. On the one hand, there is one resource per NGSI-10 operation which supports the respective functionality by providing a POST operation (colored green in the picture). On the other hand, a number of additional resources support convenience functionality (colored yellow). The latter resource structure more closely follows the REST approach and typically supports more operations (GET PUT, POST, and DELETE). The operation scope of the GET operation on these resources can further be limited by a URI parameter.

The convenience functions typically only support a subset of the functionality of the corresponding NGSI operations. Nevertheless, they enable simpler and more straightforward access. All data structures, as well as the input and output messages are represented by xml types. The definition of these types can be found in the xml schema files, and some examples are shown below.

9.4.3 Representation Format

The NGSI 10 API supports only XML as data serialization format.

9.4.4 Representation Transport

Resource representation is transmitted between client and server by using HTTP 1.1 protocol, as defined by IETF RFC-2616. Each time an HTTP request contains payload, a Content-Type header shall be used to specify the MIME type of wrapped representation. In addition, both client and server may use as many HTTP headers as they consider necessary.

9.4.5 API Operations on Context Management Component

9.4.5.1 *Standard NGSI-10 Operation Resources*

The five resources listed in the table below represent the five operations offered by systems that implement the NGSI-10 Context Management role. Each of these resources allows interaction via http POST. All attempts to interact by other verbs shall result in an HTTP error status 405; the server should then also include the 'Allow: POST' field in the response.

Resource	**Base URI: http://{serverRoot}/NGSI10**	**HTTP verbs**
		POST
Context query resource	/queryContext	Generic queries for context information.The expected request body is an instance of queryContextRequest; the response body is an instance of queryContextResponse.
Subscribe context resource	/subscribeContext	Generic subscriptions for context information. The expected request body is an instance of subscribeContextRequest; the response body is an instance of subscribeContextResponse.
Update context subscription resource	/updateContextSubscription	Generic update of context subscriptions. The expected request body is an instance of updateContexSubscriptiontRequest; the response body is an instance of updateContextSubscriptionResponse.
Unsubscribe context resource	/unsubscribeContext	Generic unsubscribe operations. The expected request body is an instance of unsubscribeContextRequest; the response body is an instance of

		unsubscribeContextResponse.
Update context resource	/updateContext	Generic context updates. The expected request body is an instance of updateContextRequest; the response body is an instance of updateContextResponse.

9.4.5.2 *Convenience Operation Resources*

The table below gives an overview of the resources for convenience operation and the effects of interacting with them via the standard HTTP verbs GET, PUT, POST, and DELETE.

Resource	**Base URI: http://{serverRoot}/NGSI10**	**HTTP verbs**			
		GET	**PUT**	**POST**	**DELETE**
Individual context entity	/contextEntities/{EntityID}	Retrieve all available information about the context entity	Replace a number of attribute values	Append context attribute values	Delete all entity information
Attribute container of individual context entity	/contextEntities/{EntityID}/attributes	Retrieve all available information about context entity	Replace a number of attribute values	Append context attribute values	Delete all entity information
Attribute of individual context entity	/contextEntities/{EntityID}/attributes/{attributeName}	Retrieve attribute value(s) and associated metadata	-	Append context attribute value	Delete all attribute values
Specific attribute value of individu	/contextEntities/{EntityID}/attributes/{attributeName}/{attributeID}	Retrieve specific attribute value	Replace attribute value	-	Delete attribute value

al context entity					
Attribute domain of individual context entity	/contextEntities/{EntityID}/attributeDomains/{attributeDomainName}	Retrieve all attribute information belonging to attribute domain	-	-	-
Context entity type	/contextEntityTypes/{tyeName}	Retrieve all available information about all context entities having that entity type	-	-	-
Attribute container of entity type	/contextEntityTypes/{typeName}/attributes	Retrieve all available information about all context entities having that entity type	-	-	-
Attribute of entity type	/contextEntityTypes/{typeName}/attributes/{attributeName}	Retrieve all attribute values of the context entities of the specific entity type	-	-	-
Attribute domain of entity type	/contextEntityTypes/{typeName}/attributeDomains/{attributeDomainName}	For all context entities of the specific type, retrieve	-	-	-

		the values of all attributes belonging to the attribute domain.			
Subscriptions container	/contextSubscriptions	-	-	Create a new subscription	-
Subscription	/contextSubscriptions/{subscriptionID}	-	Update subscription	-	Cancel subscription

9.4.6 API operation on Context Consumer Component

This section describes the resource that has to be provided by the context consumer in order to receive notifications. All attempts to interact with it by other verbs than POST shall result in an HTTP error status 405; the server should then also include the 'Allow: POST' field in the response.

Resource	URI	HTTP verbs
		POST
Notify context resource	//{notificationURI}	Generic notification. The expected request body is an instance of notifyContextRequest; the response body is an instance of notifyContextResponse.

9.5 NGSI association

9.5.1 Associations in FI-WARE NGSI 9/10

The purpose of the association concept in FI-WARE NGSI is to enable a transition from device-level information to thing-level information and vice versa. Associations describe relationships between these two levels of abstraction.

In what follows we first describe associations as an abstract concept. Subsequently, the representation of associations in the FI-WARE NGSI information model is defined, then the registration and retrieval of associations with NGSI operations is described, and finally a set of examples is provided.

9.5.1.1 *Abstract description of associations between entities and attributes*

The information model of the OMA NGSI context interfaces, and therefore also of the FI-WARE NGSI binding, is based on the notion of entities and their attributes. Entities represent arbitrary physical objects, and attributes represent properties of these objects. To simplify the discussion, we will use the syntax <entityId>.<attributeName> to refer to the attribute named <attributeName> of the entity named <entityId>.

An *attribute association* is an ordered pair of Entity/Attribute combinations, written in the form <entityId_1>.<attributeName_1> --> <entityId2>.<attributeName_2>. The semantics of such an association is that any value of <entityId_1>.<attributeName_1> can be interpreted as a value of <entityId2>.<attributeName_2>. We emphasize that this is a unidirectional relation, i.e., values of <entityId2>.<attributeName_2> are not necessarily to be interpreted as values of <entityId_1>.<attributeName_1>.

An *entity association* is an ordered pair of entities, written in the form <entityId_1> --> <entityId_2>. The entity association implicitly represents all attribute associations of the form <entityId_1>.<attributeName> --> <entityId_2>.<attributeName>, for any possible attribute <attributeName>.

For associations <entityId_1>.<attributeName_1> --> <entityId2>.<attributeName_2>, <entityId_1> is called the *source entity* and <attributeName_1> is denoted *source attribute* of the association. Furthermore, <entityId_2> and <attributeName_2> is called *target entity* and *target attribute* of the association, respectively.

9.5.1.2 *Representation of associations in FI-WARE NGSI*

The NGSI data structure used to represent associations is the ContextRegistration structure.

ContextRegistration structure

Element name	Element type	Optional	Description
EntityIdList	EntityId[1..unbound]	Yes	List of identifiers for the Context Entities being registered
ContextRegistrationAttribute	ContextRegistrationAttribute[0..unbound]	Yes	List of ContextAttributes and AttributeDomains which are made available through this registration
RegistrationMetadata	ContextMetadata[1..unbound]	Yes	Metadata characterizing this registration
ProvidingApplication	xsd:anyURI	No	URI identifying the application that provides the values of the context attributes for the

			target Context Entities

Any association information is represented in the the RegistrationMetadata field. For this purpose, a special metadata type is defined below.(ContextMetadata)

The EntityIdList and ContextRegistrationAttribute fields can be omitted, as all necessary information on the association will be contained in the Metadata. If these fields are not empty, then they shall be interpreted in the standard way, i.e., that the URI specified in the ProvidingApplication points to a NGSI-10 context provider that can provide information about the specified entities and attributes.

OMA NGSI 9/10 defines ContextMetadata as follows.

ContextMetadata structure

Element name	Element type	Optional	Description
Name	xsd:string	No	Name of the metadata
Type	xsd:anyUri	Yes	Indicates the type of the value field
Value	xsd:any	No	The actual value of the metadata

Associations are represented by contextMetadata instances whose type is set to *association*, and whose value is an instance of the following data structure.

Association structure

Element name	Element type	Optional	Description
SourceEntityId	EntityId	No	The Id of this association's source entity
TargetEntityId	EntityId	No	The Id of this association's target entity
AttributeAssociationList	AttributeAssociation[]	Yes	List of pairs of source attribute and target attribute. If this field is not present, then the Association instance represents an entity association.

AttributeAssociation structure

Element name	Element type	Optional	Description
SourceAttribute	xsd:string	No	The name of this association's source attribute

TargetAttribute	xsd:string	No	The name of this association's target attribute

In case an Association instance contains no AttributeAssociationList field, it represents the entity association between the given source entity <sourceEntityId> and the given target entity <targetEntityId>. Otherwise the Association instance represents a set of attribute associations. For each pair of source attribute <sourceAttribute> and target attribute <targetAttribute> that appears as an attributeAssociation in the AttributeAssociationList, the attribute association <sourceEntityId>.<sourceAttribute> --> <targetEntityId>.<targetAttribute> is represented.

Note that a single instance of Association can represent multiple attribute associations, but they all refer to the same pair of source and target entity. For representing associations between multiple entity pairs one has to use multiple instances of the Association structure.

9.5.1.3 *Registering Associations using NGSI-9*

Associations can be registered with NGSI-9 in the same way as context providers are registered. For registering a set of associations, a registerContextRequest message that contains the corresponding ContextRegistration structures has to be sent.

9.5.1.4 *Retrieving Associations using NGSI-9*

Associations are retrieved using the standard operations DiscoverContextAvailability and SubscribeContextAvailability defined in NGSI-9. These operations are used to ask for sets of entity/attribute combinations. A special type of operation scope is defined for indicating the set of associations expected to be returned.

Scope Type	**Scope Value**
IncludeAssociations	One of the following keywords: SOURCES, TARGETS, ALL, or NONE

The four possible scope values have the following semantics:

- NONE means that no associations are expected to be returned. This is the default value when no scope of type IncludeAssociations is present.
- TARGETS is used when the requestor wishes to find out about the *target set* of a given set of entity/attribute combinations. The target set of an entity/attribute combination set is recursively defined as the smallest set such that (1) the original entity/attribute combinations are part of the target set and (2) if there is an association <sourceEntity>.<sourceAttribute> --> <targetEntity>.<targetAttribute> and <sourceEntity>.<sourceAttribute> is in the target set, then also <targetEntity>.<targetAttribute> is in the target set. The TARGETS keyword is used to request any association between entity/attribute combinations from the target set.
- SOURCES is used when the requestor wishes to find out where and how information about a given set of entity/attribute combinations can be retrieved. The *source set* of a set of entity/attribute combinations is recursively defined as the smallest set such that (1) the original entity/attribute combinations are part of the source set and (2) if there is an association <sourceEntity>.<sourceAttribute> --> <targetEntity>.<targetAttribute> and <targetEntity>.<targetAttribute> is in the source set, then also

<sourceEntity>.<sourceAttribute> is in the source set. The SOURCES keyword is used to request any association between entity/attribute combinations from the source set and, additionally, any context provider of any entity/attribute combination from the source set.
- ALL means that the union of the responses corresponding to the SOURCES and TARGETS keywords is expected.

9.5.2 Usage Examples

9.5.2.1 *Scenario Overview*

In this set of examples we consider a simple scenario to demonstrate the usage of associations. The main actor is an NGSI-9 server; in FI-WARE this role could be played by the ?Backend? Configuration Management GE from the IoT Service Enablement chapter.

The sequence of events is as follows:

- An application registers a provider of information on attribute *measurement* of entity *sensor5* using the registerContext operation of NGSI 9.
- Another application registers an association between *sensor5.measurement* --> *house3.temperature* using also the registerContext operation of NGSI 9.
- A discovery for *house3.temperature* is issued with scope SOURCES. In FI-WARE, the issuer could be an instance of the IoT Broker GE, who has received a request for information on the attribute *temperature* of entity *house3* and seeks for sources of this information. The NGSI 9 server returns both the association *sensor5.measurement* --> *house3.temperature* and the context provider of *sensor5.measurement*.
- A discovery for *sensor5.measurement* is issued with scope TARGETS. In FI-WARE, the issuer could also here be an instance of the IoT Broker GE, who has received an attribute value of *sensor5.measurement* and seeks to find out how to interpret this value. In this case the NGSI 9 server returns the association *sensor5.measurement* --> *house3.temperature*.

9.5.2.2 *Registration*

The first registration announces a context provider and thus associations play no role in it. The request message body would look as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<registerContextRequest>
  <contextRegistrationList>
     <contextRegistration>
        <entityIdList>
           <entityId type="TemperatureSensor" isPattern="false">
               <id>sensor5</id>
           </entityId>
        </entityIdList>
        <contextRegistrationAttributeList>
```

```
            <contextRegistrationAttribute>
               <name>measurement</name>
                  <isDomain>false</isDomain>
            </contextRegistrationAttribute>
         </contextRegistrationAttributeList>

<providingApplication>http://myHomeGateway22.org</providingAppli
cation>
      </contextRegistration>
   </contextRegistrationList>
   <duration>PT1M</duration>
   <registrationId></registrationId>
</registerContextRequest>
```

The second registration announces the association. As mentioned above, the association is represented in the metadata while the entity and attribute lists are absent.

```
<?xml version="1.0" encoding="UTF-8"?>
<registerContextRequest>
   <contextRegistrationList>
      <contextRegistration>
         <registrationMetadata>
            <contextMetadata>
               <name>association1</name>
               <type>Association</type>
               <value>
                  <entityAssociation>
                     <sourceEntityId    type="TemperatureSensor"
isPattern="false">
                        <id>Sensor5</id>
                     </sourceEntityId>
                     <targetEntityId                 type="House"
isPattern="false">
                        <id>house3</id>
                     </targetEntityId>
                  </entityAssociation>
                     <attributeAssociationList>
                        <attributeAssociation>

<sourceAttribute>measurement</sourceAttribute>
```

```
<targetAttribute>indoorTemperature</targetAttribute>
                          </attributeAssociation>
                       </attributeAssociationList>
                  </value>
              </contextMetadata>
          </registrationMetadata>
          <providingApplication>http://www.fi-
ware.eu/NGSI/association</providingApplication>
       </contextRegistration>
    </contextRegistrationList>
   <duration>PT1M</duration>
  <registrationId></registrationId>
</registerContextRequest>
```

9.5.2.3 *Discovery of Sources*

To discover the sources of *house3.temperature*, the following request is issued.

```
<?xml version="1.0" encoding="UTF-8"?>
<discoverContextAvailabilityRequest>
   <entityIdList>
      <entityId type="House" isPattern="false">
         <id>house3</id>
      </entityId>
   </entityIdList>
   <attributeList>
      <attribute>indoorTemperature</attribute>
   </attributeList>
   <restriction>
      <scope>
         <operationScope>
            <scopeType>IncludeAssociations</scopeType>
            <scopeValue>SOURCES</scopeValue>
         </operationScope>
      </scope>
   </restriction>
</discoverContextAvailabilityRequest>
```

The response from the NGSI 9 server looks as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<discoverContextAvailabilityResponse>
   <contextRegistrationResponseList>
      <contextRegistrationResponse>
         <contextRegistration>
            <entityIdList>
               <entityId                type="TemperatureSensor"
isPattern="false">
                  <id>sensor5</id>
               </entityId>
            </entityIdList>
            <contextRegistrationAttributeList>
               <contextRegistrationAttribute>
                  <name>measurement</name>
                  <isDomain>false</isDomain>
               </contextRegistrationAttribute>
            </contextRegistrationAttributeList>

<providingApplication>http://myHomeGateway22.org</providingAppli
cation>
         </contextRegistration>
      </contextRegistrationResponse>
      <contextRegistrationResponse>
         <contextRegistration>
            <registrationMetaData>
               <contextMetadata>
                  <name>association1</name>
                  <type>Association</type>
                  <value>
                     <sourceEntityId    type="TemperatureSensor"
isPattern=”false”>
                        <id>sensor5</id>
                     </sourceEntityId>
                     <targetEntityId                type="House"
isPattern=”false”>
                        <id>house3</id>
                     </targetEntityId>
                     <AttributeAssociationList>
                        <AttributeAssociation>

<sourceAttribute>measurement</sourceAttribute>
```

```
<targetAttribute>indoorTemperature</targetAttribute>
                        </AttributeAssociation>
                     </AttributeAssociationList>
                  </value>
               </contextMetadata>
            </registrationMetaData>
            <providingApplication>http://www.fi-
ware.eu/NGSI/association</providingApplication>
         </contextRegistration>
      </contextRegistrationResponse>
   </contextRegistrationResponseList>
</discoverContextAvailabilityResponse>
```

9.5.2.4 *Discovery of Targets*

The request for target discovery deviates from the source discovery by the scope value.

```
<?xml version="1.0" encoding="UTF-8"?>
<discoverContextAvailabilityRequest>
   <entityIdList>
      <entityId type="TemperatureSensor" isPattern="false">
         <id>sensor5</id>
      </entityId>
   </entityIdList>
   <attributeList>
      <attribute>measurement</attribute>
   </attributeList>
   <restriction>
      <scope>
         <operationScope>
            <scopeType>IncludeAssociations</scopeType>
            <scopeValue>TARGETS</scopeValue>
         </operationScope>
      </scope>
   </restriction>
</discoverContextAvailabilityRequest>
```

The response here looks as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<discoverContextAvailabilityResponse>
   <contextRegistrationResponseList>
      <contextRegistrationResponse>
         <contextRegistration>
            <registrationMetaData>
               <contextMetadata>
                  <name>association1</name>
                  <type>Association</type>
                  <value>
                     <sourceEntityId    type="TemperatureSensor"
isPattern=”false”>
                        <id>sensor5</id>
                     </sourceEntityId>
                     <targetEntityId                 type="House"
isPattern=”false”>
                        <id>house3</id>
                     </targetEntityId>
                     <AttributeAssociationList>
                        <AttributeAssociation>

<sourceAttribute>measurement</sourceAttribute>

<targetAttribute>indoorTemperature</targetAttribute>
                        </AttributeAssociation>
                     </AttributeAssociationList>
                  </value>
               </contextMetadata>
            </registrationMetaData>
            <providingApplication>http://www.fi-
ware.eu/NGSI/association</providingApplication>
         </contextRegistration>
      </contextRegistrationResponse>
   </contextRegistrationResponseList>
</discoverContextAvailabilityResponse>
```

9.6 FI-WARE NGSI: publicly available documents

Version 1.0 of the FI-WARE RESTful binding of OMA NGSI 9/10 can be retrieved here.

The zip file contains the following components:

- RESTful binding specification document for NGSI-9
- RESTful binding specification document for NGSI-10
- A conceptual description of the NGSI association concept of FI-WARE
- XML schema files defining the XML syntax of the message bodies
- XML files serving as examples for the XML message bodies

10 IETF CoRE Open RESTful API Specification

10.1.1.1 *Introduction to IETF CoRE*

IETF CoRE (Constrained RESTful Environments) is an IETF working group that provides a framework for applications intended to be run on constrained devices. The CoRE framework ~~views~~ regards sensor and actuator resources as web resources. As part of the framework for building these applications, CoRE has defined a Constrained Application Protocol (CoAP) for the manipulation of Resources on a Device. CoAP is an asynchronous and efficient client-server messaging protocol over UDP. It contains the same methods as in HTTP (GET, PUT, POST, DELETE) with an additional OBSERVE capability for initiating subscriptions. Currently there is no work on an application profile for CoAP in IETF. However, there exists work on this within other organizations such as the IPSO Alliance, although this has not been made publically available yet. The current set of CoRE specifications also describe a number of supporting functionality for constrained applications such as resource management, handling sleeping devices, security etc.

10.1.1.2 *IETF CoRE API core*

The IETF CoRE API used in Fiware is a RESTful, resource-oriented API accessed via HTTP or CoAP that can use a wide veriety of representations for information interchange such as plain text, xml or json.

The Ericsson Gateway, which is one implementation of the Fiware IoT gateway, makes use of IETF CoRE specified RESTful interfaces for the northbound interface towards the Backend Device Management GE and southbound towards CoAP-compliant devices. IETF CoRE can be mapped to both HTTP and CoAP. The Ericsson Gateway provides both HTTP and CoAP APIs northbound but only a CoAP API southbound to constrained devices.

For the northbound interface, the main interaction types are:

- Create, Read, Update, Delete operation on an IoT resource hosted on the gateway or on a device
- Lookup of resource description from the resource directory
- Initiate subscription to sensor resource

For the southbound interface, the main interaction types are:

- Create, Read, Update, Delete operation on an IoT resource hosted on a device
- Initiate subscription to sensor resource

10.1.1.3 *Intended Audience*

This guide is intended for developers of GE implementations and providers of CoAP devices intended for Fiware. This document specifies the API that has to be implemented in order to ensure interoperability with mainly the Backend Device Management GE from the IoT Chapter, as well as CoAP devices.

Prerequisites: Throughout this specification it is assumed that the reader is familiar with

- ReSTful web services
- HTTP/1.1

- IETF CoRE specifications available at: http://tools.ietf.org/wg/core/

10.1.1.4 *Change history*

This version of the IETF CoRE Guide replaces and obsoletes all previous versions. The most recent changes are described in the table below:

Revision Date	Changes Summary
July 12, 2012	• 1st stable version

10.1.2 General IETF CoRE API information

10.1.2.1 *Representation Format*

N/A. A number of different serialization formats can be used.

10.1.2.2 *Resource Identification*

For HTTP transport, this is made using the mechanisms described by HTTP protocol specification as defined by IETF RFC-2616. For CoAP check CoAP.

10.1.2.3 *Links and References*

The CoRE Link Format specification describes how resources link to each other. It is possible to discover resources on a device by sending a GET request to /.well-known/core.

10.1.3 API Operations

10.1.3.1 *Northbound API*

The northbound interface is based on the HTTP mapping of CoAP as described in: http://tools.ietf.org/id/draft-castellani-core-http-mapping-05.txt. The device CoAP URL is embedded in the HTTP URL together with the gateway address. According to this mode of operation an HTTP address: "http://authority/coap/SA_ADDR:{port}/resource" is translated to "coap://SA_ADDR:{port}/resource".

10.1.3.1.1 Accessing resources

Create, Read, Delete, Update on an resource:

Verb	URI	Description
GET	/coap/sensor_address/resourceName	Read resource
POST	/coap/sensor_address/resourceName	Update resource (e.g. actuate)
PUT	/coap/sensor_address/resourceName	Create resource

DELETE	/coap/sensor_address/resourceName	Delete resource

Examples:

```
Sensor data request
Req:                              [HTTP                              GET]
http://m2m.ericsson.com/coap/2011:DB8::11/gpio/btn       (Accept:
text/plain)


Res: 200 OK, Body: 2 (text/plain)


Actuation
Request:                           [HTTP                            POST]
http://m2m.ericsson.com/coap/2011:DB8::11/lt/ledr/on     (Accept:
text/plain)
          Body:1


Response: 200 OK (text/plain)
```

10.1.3.1.2 Accessing resource directory

Lookup and publish resource descriptions :

Verb	URI	Description
GET	/rd	Retrieve resource descriptions
POST	/rd	Publish resource description
PUT	/rd/{resourceID}	Update resource description
DELETE	/rd/{resourceID}	Delete resource description

10.1.3.2 ***Southbound API***

The southbound "native" interface towards devices is based on the CoAP protocol as described in http://tools.ietf.org/html/draft-ietf-core-coap-10.

10.1.3.2.1 Accessing resources

Create, Read, Update Delete, Observe resource:

Verb	URI	Description
GET	/resourceName	Read resource

POST	/resourceName	Update resource
PUT	/resourceName	Create resource
DELETE	/resourceName	Delete resource
OBSERVE	/resourceName	Subscribe to resource

Examples:

```
Sensor data request
Request: GET coap:/2011:DB8::11/gpio/btn (Accept: text/plain)

Response: 2.05 Content (text/plain)
Body: 2

Actuation
Req: PUT or POST /lt/ledr/on (Accept: text/plain)
Body:1

Res: 2.04 Changed (text/plain)

Observation (subscription)
Req: GET /gpio/btn (Accept: text/plain) (Observe:0)

Res: 2.05 Content (text/plain) (Observe:0)
Body: 2

Res: 2.05 Content (text/plain) (Observe:1)
Body: 3

Res: 2.05 Content (text/plain) (Observe:2)
Body: 4
```

11 ETSI M2M mId Open RESTful API Specification

11.1.1.1 *Introduction to the ETSI M2M mId API*

11.1.1.2 *ETSI M2M mId API Core*

The ETSI M2M mId API is a RESTful, resource-oriented API accessed via HTTP that uses XML-based or JSON-based representations for information interchange. The mId is the ETSI M2M interface between the Backend and Gateways/IoT-compliant Devices. This reference point provides the following functionality:

- Registration of Devices/Gateways to the Backend,
- Request to Read/Write subject to proper authorization information in the Backend,
- Subscription and notification to specific resource-level events,
- Device management actions (e.g. software upgrade, configuration management).

11.1.1.3 *Intended Audience*

This guide is intended for both developers of GE implementations and IoT Application programmers. For the former, this document specifies the API that has to be implemented in order to ensure interoperability with other GEs from the IoT Chapter of FI-WARE. For the latter, this document describes how to assemble instances of the FI-WARE IoT Platform.

Prerequisites: Throughout this specification it is assumed that the reader is familiar with

- ReSTful web services
- HTTP/1.1
- XML data serialization formats

We also refer the reader to the ETSI TS 102 921: M2M; mla,dla and mld interfaces and binding documents for details on the resource structure and message formats.

11.1.1.4 *API Change History*

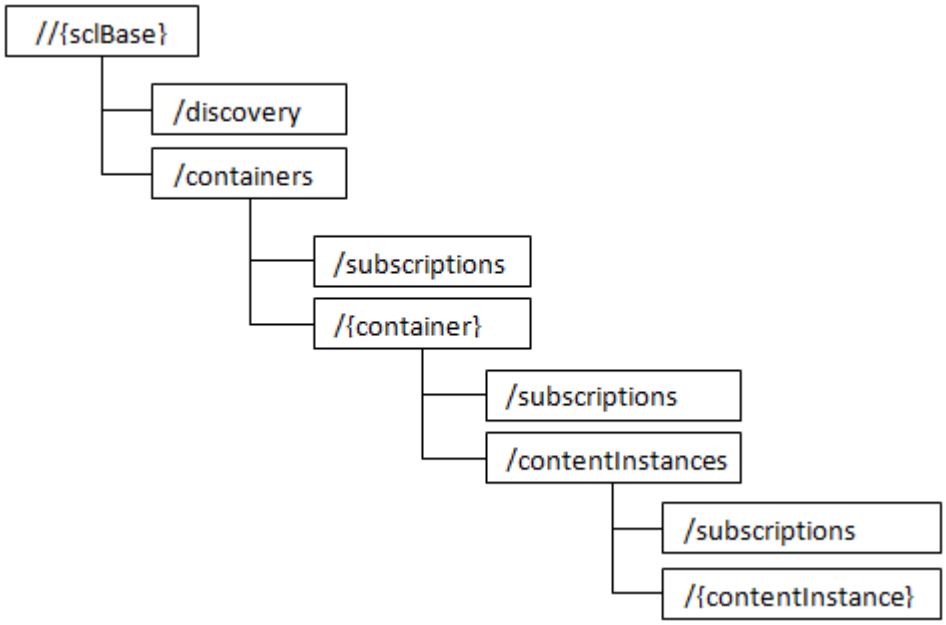
This version of the Gateway Device Management GE Northbound API Guide replaces and obsoletes all previous versions. The most recent changes are described in the table below

Revision Date	Changes Summary
October 14, 2013	• Third version

11.1.1.5 *Additional Resources*

This document is to be considered to be a guide to the planned ETSI M2M mId API elements. The complete specification can be found at ETSI TS 102 921: M2M; mla,dla and mld interfaces.

11.1.1.6 *General ETSI M2M mId API Information*



11.1.1.6.1 Representation Format

The ETSI M2M mId API supports currently XML as data serialization format.

11.1.1.6.2 Representation Transport

Resource representation is transmitted between client and server by using HTTP 1.1 protocol, as defined by IETF RFC-2616. Each time an HTTP request contains payload, a Content-Type header shall be used to specify the MIME type of wrapped representation. In addition, both client and server may use as many HTTP headers as they consider necessary.

11.1.2 API Operations

In this section we list the operations that are delivered. For an in-depth specification of the listed resources and operations, please refer to the official specifications.

11.1.2.1 *Data Retrieval*

11.1.2.2 *Container*

Container resource is a generic resource that shall be used to exchange data between applications and/or SCLs by using the container as a mediator that takes care of buffering the data. Exchange of data between applications (e.g. on device and network side) is abstracted from the need to set up direct connections and allows for scenarios where both parties in the exchange are not online at the same time.

Verb	URI	Description
GET	/{sclBase}/containers	Retrieve the content of a container root resource.
PUT	/{sclBase}/containers	Update the content of a container root resource
GET	/{sclBase}/containers/{container}	Read the attributes of a container instance and references to its direct child resources.
PUT	/{sclBase}/containers/{container}	Update all of the attributes in a container instance.
POST	/{sclBase}/containers/{container}	Create a new container instance resource in a containers collection.
DELETE	/{sclBase}/containers/{container}	Delete the container instance.

11.1.2.3 *Content Instance*

A Content Instance represents a data instance in the container. The content of the instance is opaque to the M2M platform and it might even be encrypted. However, there is meta-data associated with an instance which shall be accessible.

Verb	URI	Description
GET	/{sclBase}/containers/{container}/contentInstances	Read the content of the content instances root resource. This request is used to get a list of data of all instances in the addressed contentInstances collection.
GET	/{sclBase}/containers/{container}/contentInstances/{contentInstance}	Read a content instance resource. This operation is used to read the entire resource or an individual attribute.
POST	/{sclBase}/containers/{container}/contentInstances/{contentInstance}	Adds a content instance resource to the content instances root resource.
DELETE	/{sclBase}/containers/{container}/contentInstances/{contentInstance}	Delete a content instance resource.

11.1.2.4 *Subscription*

A subscriber (an Application or an SCL) may ask to be notified when resources are modified. A subscription is the relation between the subscriber and the Hosting-SCL of the Subscribed-to Resource. The Hosting SCL shall notify the subscriber of any changes in the Subscribed-to Resource under the conditions provided when the subscription was created or modified. A subscription shall be represented by a resource itself. This allows manipulation of the subscription in a resource oriented manner, e.g. the conditions of a subscription may be modified by modifying the <subscription> resource or parts thereof, or a subscriber may unsubscribe by deleting the subscription resource. The subscriptions resource structure is present in the following places:

- /{sclBase}/containers/subscriptions
- /{sclBase}/containers/{container}/subscriptions
- /{sclBase}/containers/{container}/contentInstances/subscriptions

11.1.2.5 *SubscriptionResource*

For keeping track of subscriptions to a subscribe-able resource, the subscriptions-resource is used as a child resource of the subscribe-able parent. The subscriptions-resource contains a collection of 0..n {subscription} resources that represent individual subscriptions to the subscribable resource (i.e. the parent of the subscriptions resource). A subscription instance resource shall represent an active (asynchronous) subscription to a resource. I.e. <resourceURI>/subscriptions/<subscription> denotes an active subscription to the resource identified by the resourceURI. Subscriptions shall only be possible on resources that have a subscriptions sub-resource.

Verb	URI	Description
GET	{resourceURI}/subscriptions	Retrieve a subscriptions root resource
GET	{resourceURI}/subscriptions/{subscription}	Reads a subscription instance resource
PUT	{resourceURI}/subscriptions/{subscription}	Updates the provided attributes of an existing subscription instance resource.
POST	{resourceURI}/subscriptions/{subscription}	Creates a new subscription instance resource. The subscription applies to the resource that is associated with the parent resource of the subscriptions collection resource.
DELETE	{resourceURI}/subscriptions/{subscription}	Deletes a subscription instance resource

12 FI-WARE Open Specifications Legal Notice (essential patent licence)

12.1 General information

"FI-WARE Project Partners" refers to Parties of the FI-WARE Project in accordance with the terms of the FI-WARE Consortium Agreement.

"Copyright Holders" of this FI-WARE Open Specification is/are the Party/Parties identified as the copyright owner/s on the wiki page of the FI-WARE Open Specification that contains the link to this Legal Notice.

12.2 Use Of Specification - Terms, Conditions & Notices

The material in this specification details, as of the date when Copyright Holders contributed it, a FI-WARE Generic Enabler Specification (hereinafter “Specification”) that is provided, in accordance with the terms, conditions and notices set forth below (“License”). This Specification does not represent a commitment to implement any portion of this Specification in any company's products. The information contained in this Specification is subject to change without notice.

12.3 Copyright License

Subject to all of the terms and conditions below, the Copyright Holders in this Specification hereby grant you, the individual or legal entity exercising permissions granted by this License, a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide royalty free license (without the right to sublicense) under its respective copyrights incorporated in the Specification, to copy and modify this Specification and to distribute copies of the modified version, and to use this Specification, to create and distribute special purpose specifications and software that is an implementation of this Specification.

12.4 Patent License

“Specification Essential Patents” shall mean patents and patent applications, which are necessarily infringed by an implementation compliant with the Specification and which are owned by any of the Copyright Holders of this Specification. “Necessarily infringed” shall mean that no commercially and/or technical reasonable alternative exists to avoid infringement.

Each of the Copyright Holders, hereby agrees to grant you, on fair, reasonable and non-discriminatory terms, a personal, nonexclusive, non-transferable, non-sub-licensable, royalty-free, paid up, worldwide license, under their respective Specification Essential Patents, to make, use, sell, offer to sell, import and/or distribute software implementations compliant with the Specification.

If you institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that this Specification constitutes direct or contributory patent infringement, then any patent licenses granted to you under this License for that Specification shall terminate as of the date such litigation is filed.

The FI-WARE Project Partners and/or Copyright Holders shall not be responsible for identifying patents for which a license may be required for any implementation of any FI-WARE Specification, or for conducting legal inquiries into the legal validity or scope

of those patents that are brought to its attention. FI-WARE specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

12.5 General Use Restrictions

Any unauthorized use of this Specification may violate copyright laws, trademark laws, and communications regulations and statutes. This Specification contains information which is protected by copyright. All Rights Reserved. This Specification shall not be used in any form or for any other purpose different from those herein authorized, without the permission of the respective Copyright Holders.

For avoidance of doubt, the rights granted are only those expressly stated in this Legal Notice herein. No other rights of any kind are granted by implication, estoppel, waiver or otherwise

12.6 Disclaimer Of Warranty

WHILE THIS SPECIFICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE FI-WARE PROJECT PARTNERS AND THE COPYRIGHT HOLDERS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS SPECIFICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, WARRANTY OF NON INFRINGEMENT OF THIRD PARTY RIGHTS, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE.

YOU ARE SOLELY RESPONSIBLE FOR DETERMINING THE APPROPRIATENESS OF USING OR REDISTRIBUTING THIS SPECIFICATION AND ASSUME ANY RISKS ASSOCIATED WITH YOUR EXERCISE OF PERMISSIONS UNDER THIS LICENSE

IN NO EVENT AND UNDER NO LEGAL THEORY SHALL THE FI-WARE PROJECT PARTNERS AND THE COPYRIGHT HOLDERS BE LIABLE FOR ERRORS CONTAINED IN THIS SPECIFICATION OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS SPECIFICATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF SOFTWARE DEVELOPED USING THIS SPECIFICATION IS BORNE BY YOU. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THE PATENT AND COPYRIGHT LICENSES GRANTED TO YOU TO USE THIS SPECIFICATION.

12.7 Trademarks

You shall not use any trademark, marks or trade names (collectively, "Marks") of the FI-WARE Project Partners or the Copyright Holders or the FI-WARE project without their prior written consent, except as required for reasonable and customary use in describing the origin of this Specification and reproducing the content of this Legal Notice.

12.8 Issue Reporting

This Specification is subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Procedure described on the web page http://www.fi-ware.eu.

However there is no obligation on the part of the Copyright Holder to provide any review, improvement, bug fixes or modifications this Specification to address any reported ambiguities, inconsistencies, or inaccuracies.

13 FI-WARE Open Specification Legal Notice (implicit patents license)

13.1 General Information

"FI-WARE Project Partners" refer to Parties of the FI-WARE Project in accordance with the terms of the FI-WARE Consortium Agreement.

Copyright Holders of the FI-WARE Open Specification is/are the Party/Parties identified as the copyright owner/s on the wiki page of the FI-WARE Open Specification that contains the link to this Legal Notice.

13.2 Use of Specification - Terms, Conditions & Notices

The material in this specification details a FI-WARE Generic Enabler Specification (hereinafter "Specification") and is provided in accordance with the terms, conditions and notices set forth below ("License"). This Specification does not represent a commitment to implement any portion of this Specification in any company's products. The information contained in this Specification is subject to change without notice.

13.3 Licenses

Subject to all of the terms and conditions below, the copyright holders in this Subject to all of the terms and conditions below, the Copyright Holders in this Specification hereby grant you, the individual or legal entity exercising permissions granted by this License, a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide, royalty free license (without the right to sublicense) under its respective copyrights incorporated in the Specification, to copy and modify this Specification and to distribute copies of the modified version, and to use this Specification, to create and distribute special purpose specifications and software that is an implementation of this Specification.

13.4 Patent Information

The FI-WARE Project Partners and/or Copyright Holders shall not be responsible for identifying patents for which a license may be required for any implementation of any FI-WARE Specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. FI-WARE specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

13.5 General Use Restrictions

Any unauthorized use of this Specification may violate copyright laws, trademark laws, and communications regulations and statutes. This Specification contains information which is protected by copyright. All Rights Reserved. This Specification shall not be used in any form or for any other purpose different from those herein authorized, without the permission of the respective Copyright Holders.

For avoidance of doubt, the rights granted are only those expressly stated in this Legal Notice herein. No other rights of any kind are granted by implication, estoppel, waiver or otherwise

13.6 Disclaimer Of Warranty

WHILE THIS SPECIFICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE FI-WARE PROJECT PARTNERS AND THE COPYRIGHT HOLDERS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS SPECIFICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, WARRANTY OF NON INFRINGEMENT OF THIRD PARTY RIGHTS, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE.

YOU ARE SOLELY RESPONSIBLE FOR DETERMINING THE APPROPRIATENESS OF USING OR REDISTRIBUTING THIS SPECIFICATION AND ASSUME ANY RISKS ASSOCIATED WITH YOUR EXERCISE OF PERMISSIONS UNDER THIS LICENSE

IN NO EVENT AND UNDER NO LEGAL THEORY SHALL THE FI-WARE PROJECT PARTNERS AND THE COPYRIGHT HOLDERS BE LIABLE FOR ERRORS CONTAINED IN THIS SPECIFICATION OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS SPECIFICATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF SOFTWARE DEVELOPED USING THIS SPECIFICATION IS BORNE BY YOU. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THE PATENT AND COPYRIGHT LICENSES GRANTED TO YOU TO USE THIS SPECIFICATION.

13.7 Trademarks

You shall not use any trademark, marks or trade names (collectively, "Marks") of the FI-WARE Project Partners or the Copyright Holders or the FI-WARE project without prior written consent, except as required for reasonable and customary use in describing the origin of this Specification and reproducing the content of this Legal Notice.

13.8 Issue Reporting

This Specification is subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Procedure described on the web page http://www.fi-ware.eu.

However there is no obligation on the part of the Copyright Holder to provide any review, improvement, bug fixes or modifications this Specification to address any reported ambiguities, inconsistencies, or inaccuracies.