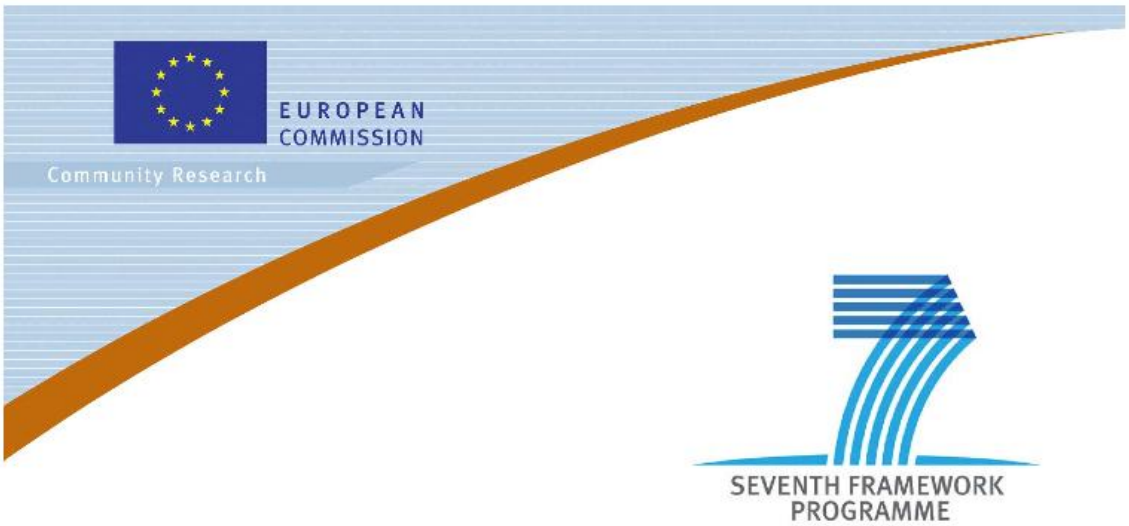


Private Public Partnership Project (PPP)

Large-scale Integrated Project (IP)

D.6.1.1b: FI-WARE GE Open Specifications (Data/Context Management Chapter)

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-WP6-D.6.1.1b

Project Document Date: 2012-10-11

Deliverable Type and Security: Public

Author: FI-WARE Consortium

Contributors: FI-WARE Consortium

Table of Content

1 Introduction

1.1 Executive Summary

This document comprises the Open Specifications of the Generic Enablers in the FI-WARE Data/Context Management chapter. These Generic Enablers provide advanced platform functionalities dealing with gathering, processing, interchange and exploitation of data at large scale, thus easing the development of intelligent, customized, personalized, context-aware and enriched application and services beyond those available on the current Internet.

The functionality of the chapter 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 enable smarter, more customized/personalized and context-aware applications and services by the means of a set of assets able to gather, exchange, process and analyze massive data in a fast and efficient way. Nowadays, several well-known free Internet services are based on business models that exploit massive data provided by end users.

This data is exploited in advertising or offered to 3rd parties so that they can build innovative applications. Twitter, Facebook, Amazon, Google and many others are examples of this.

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

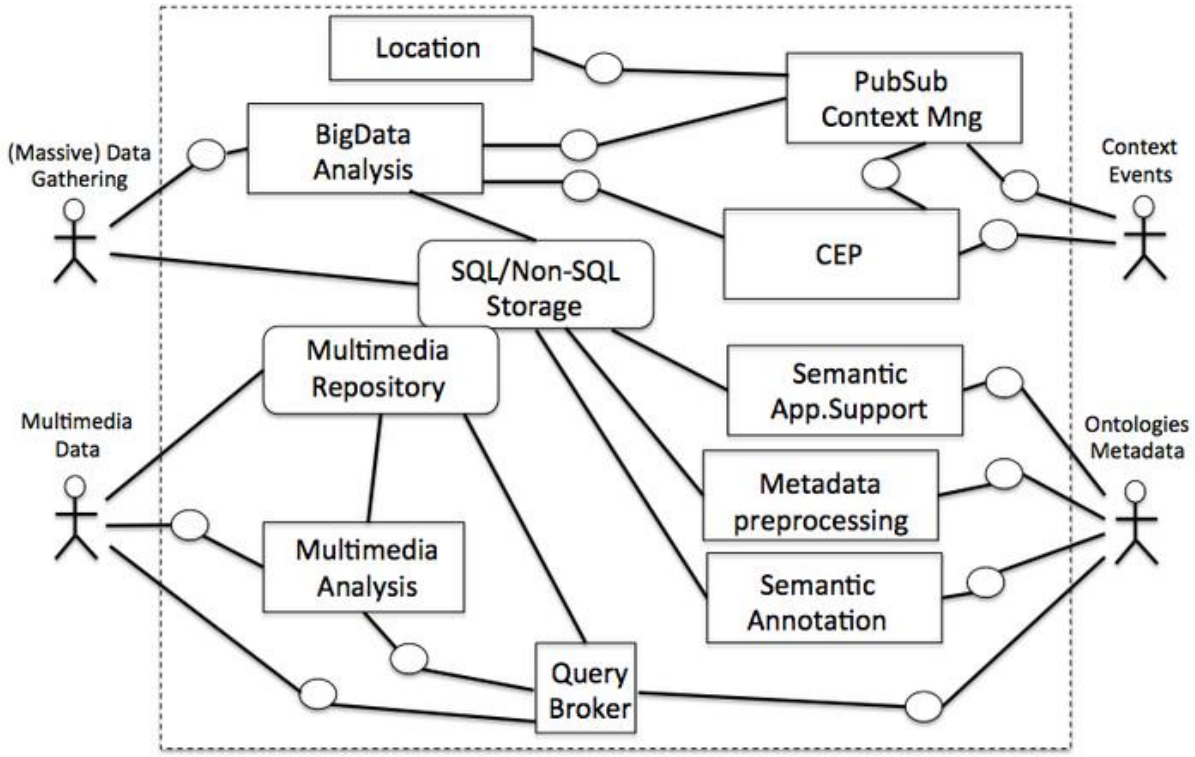
FI-WARE Data/Context Management GEs enables to:

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

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

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

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



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

http://wiki.fi-ware.eu

Data/Context Management

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/

FIWARE.OpenSpecification.Data.BigData

BigData Analysis Open RESTful API Specification (PRELIMINARY)

FIWARE.OpenSpecification.Data.CEP

Complex Event Processing Open RESTful API Specification (PRELIMINARY)

FIWARE.OpenSpecification.Data.CompressedDomainVideoAnalysis

Compressed Domain Video Analysis Open RESTful API Specification (PRELIMINARY)

FIWARE.OpenSpecification.Data.Location

Location Server Open RESTful API Specification (PRELIMINARY)

FIWARE.OpenSpecification.Data.MetadataPreprocessing

Metadata Preprocessing Open RESTful API Specification (PRELIMINARY)

FIWARE.OpenSpecification.Data.PubSub

FI-WARE NGSI Open RESTful API Specification (PRELIMINARY)

FI-WARE NGSI-9 Open RESTful API Specification (PRELIMINARY)

FI-WARE NGSI-10 Open RESTful API Specification (PRELIMINARY)

ContextML/CQL over HTTP Open RESTlike API Specification (PRELIMINARY)

ContextML API

CQL API

FIWARE.OpenSpecification.Data.QueryBroker

Query Broker Open RESTful API Specification (PRELIMINARY)

FIWARE.OpenSpecification.Data.SemanticAnnotation

FIWARE.OpenSpecification.Data.SemanticSupport

Semantic Support Open RESTful API Specification (PRELIMINARY)

FIWARE.ArchitectureDescription.Data.SemanticSupport.OMV Open Specification (DRAFT)

FI-WARE_Open_Specifications_Legal_Notice

Open Specifications Interim Legal Notice

1.6 Typographical Conventions

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

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

1.6.1 Links within this document

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

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

1.6.2 Figures

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

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

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

1.6.3 Sample software code

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

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

1.7 Acknowledgements

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

1.8 Keyword list

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

1.9 Changes History

Release	Major changes description	Date	Editor
v0	First review of deliverable submission	2012-08-31	TID
v1	Second review of deliverable submission	2012-10-31	TID
V2	Version for submission	2012-11-08	TID

2 FIWARE OpenSpecification Data BigData

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

Name	FIWARE.OpenSpecification.Data.BigData
Chapter	Data/Context Management,
Catalogue-Link to Implementation	<BigData Analysis>
Owner	FI-WARE Telefonica I+D, Andreu Urruela/Grant Croker

2.1 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.

2.1.1 Copyright

2.1.2 Legal Notice

2.2 Overview

2.2.1 Introduction to the BigData Analysis GE

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

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

2.2.1.1 *Data Stream Analysis*

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

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

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

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

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

2.2.1.2 *High-performance distributed file-system*

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

2.2.1.3 *NoSQL document-orientated storage*

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

2.2.2 Target Usage

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

On the other hand, Big Data Stream Processing could be defined as the technology to process continuous unbounded and large streams of data extracting relevant insights on the go. This technology could be applied to scenarios where it is not necessary to store all

incoming data or it has to be processed “on the go”, immediately after it becomes available. Additionally, this technology would be more suitable to big-data problems where low latency in generation of insights is expected. In this particular case, insights would be continuously generated, parallel to incoming data, allowing continuous estimations and predictions.

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

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

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

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

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

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

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

2.2.3 Example Scenario

Imagine you are receiving a high volume stream of data that contains, amongst other things, a customer reference number (IMSI), a terminal ID (IMEI) and the ID of the cell tower they

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

2.3 Basic Concepts

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

2.3.1 MapReduce

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

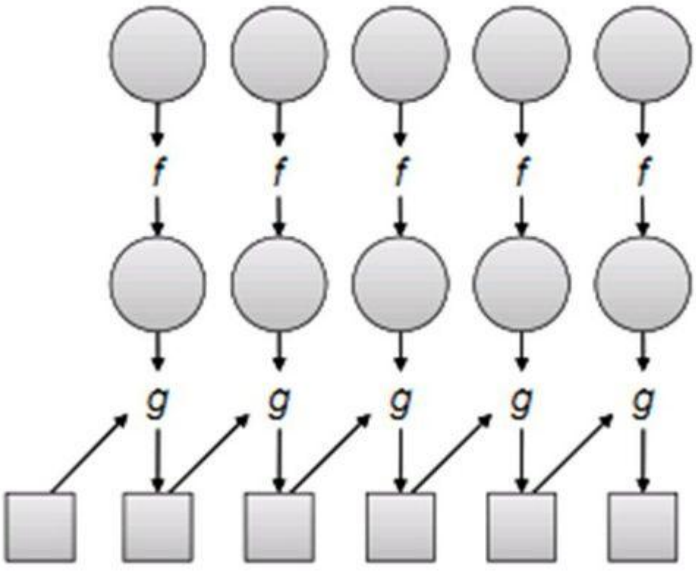


Figure BDA-1 – Functional programming diagram, with map (f) and fold (g) functions

From the field of functional programming, it is proved that any problem whose solution can be expressed in terms of commutative and associative functions, can be expressed in two types of functions: map (named also map in the MR paradigm) and fold (named reduce in the

MR paradigm). Any job must be expressed as a sequence of these functions. These functions have a restriction: they operate on some input data, and produce a result without side effects, i.e. without modifying neither the input data nor any global state. This restriction is the key point to allow an easy parallelization.

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

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

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

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

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

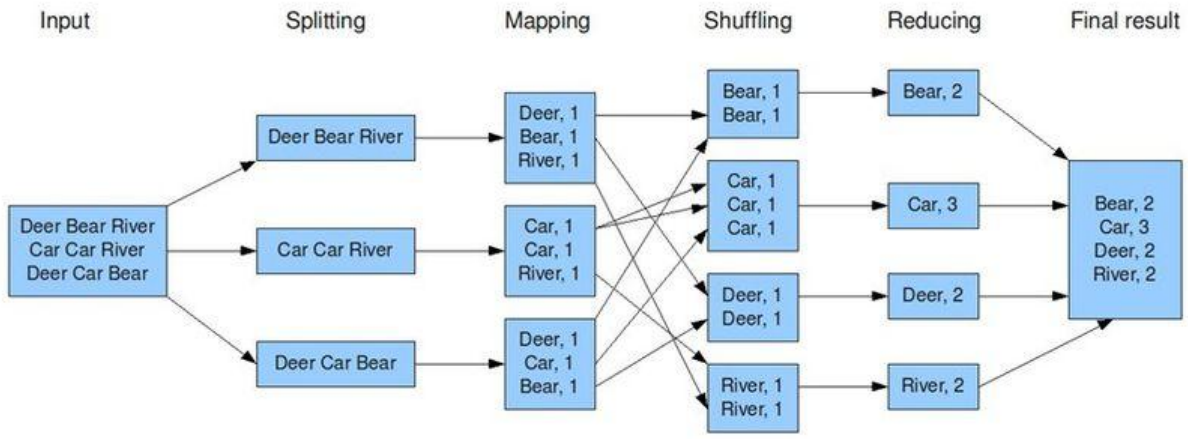


Figure BDA-2 – Word count algorithm implementation in MapReduce

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

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

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

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

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

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

2.3.2 NoSQL

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

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

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

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

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

The different types of NoSQL database are

- Column Store - Data storage is orientated to the column rather than the row as it is with traditional DBMS engines, favouring aggregate operations on columns typically used in data warehousing. Example implementations: Hadoop HBase and Google's BigTable.
- Key Value Store - A schema-less storage system, data is stored in key-value pairs. Data is accessed via a hash table using the unique key. Example implementation:[BDA9]
- Document Store - Similar to Key Value storage, document storage works with semi-structured data that contain a collection of key-value pairs. Unlike key-value storage these documents can contain child elements that store relevant knowledge to that particular document. Unlike in traditional DBMS engines, document orientated storage does not require that every document contain all the fields if no information is there for that particular document. Example implementations: [BDA13][BDA15]

- Graph Database - Using a graph structure, data about an entity is stored within a node, relationships between the nodes are defined in the edges that interconnect the nodes. This allows for lookups which utilize associative datasets as the information that relates to any given node is already present, eliminating the need to perform any joins. Example implementations: [BDA15]

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

2.4 Big Data Analysis Generic Architecture

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

- Data Stream
- Module Programming
- Analytical scripting

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

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

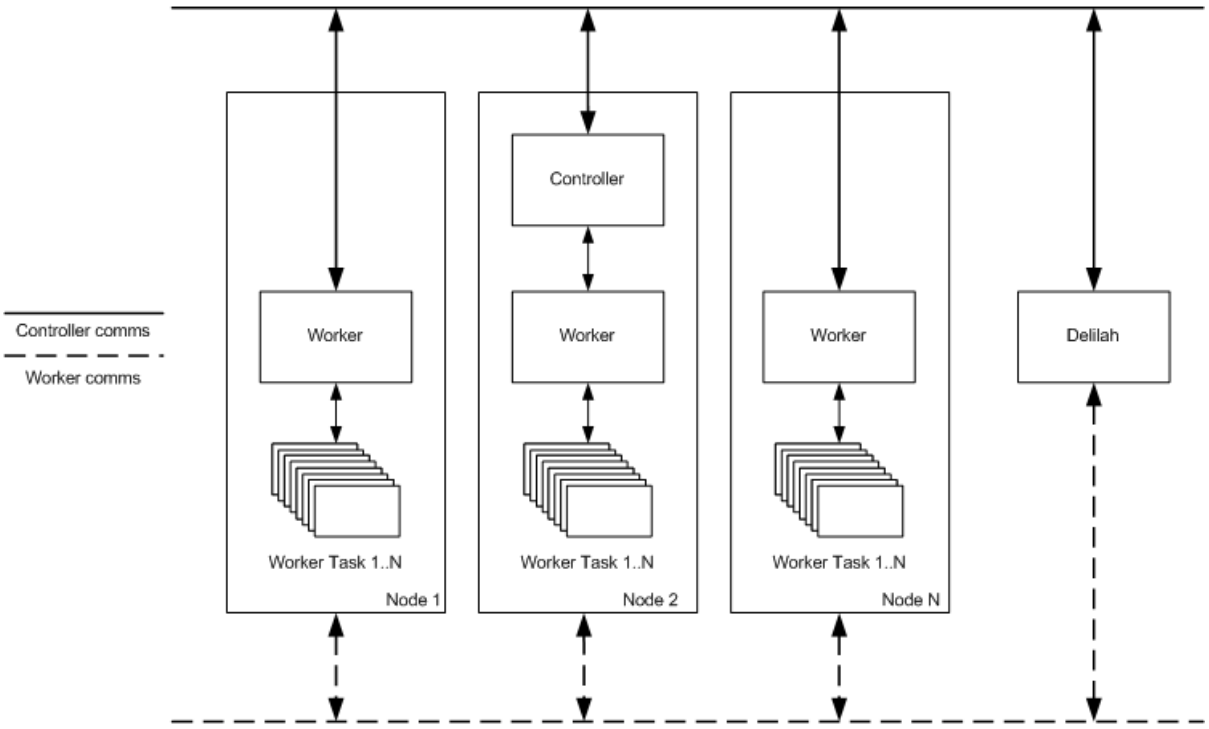


Figure BDA-3 - Platform overview

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

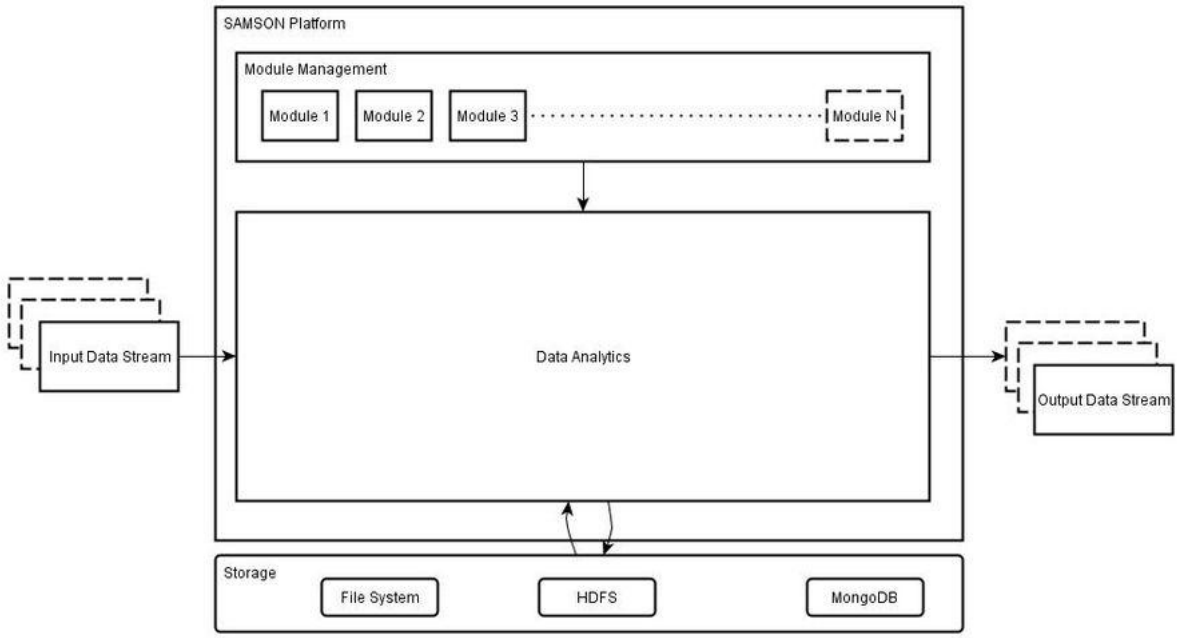


Figure BDA-4 - Overall architecture view

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

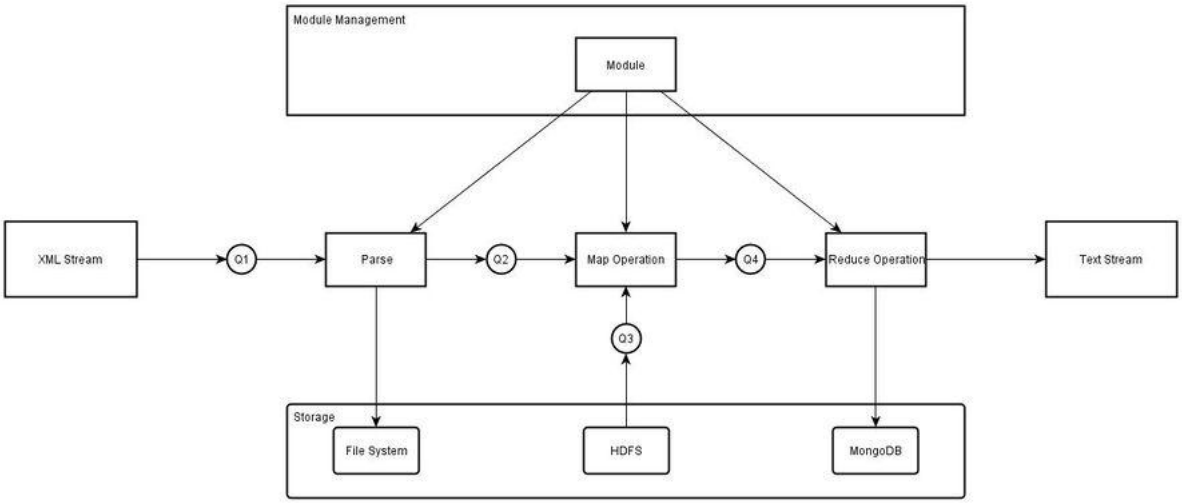


Figure BDA-5 - Example analytical flow

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

2.5 Main Interactions

2.5.1 Modules and Interfaces

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

2.5.2 Data Stream Interfaces

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

2.5.2.1 *Inject data into the platform*

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

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

2.5.2.2 *Extract data from the platform*

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

2.5.3 Analysis Module Programming

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

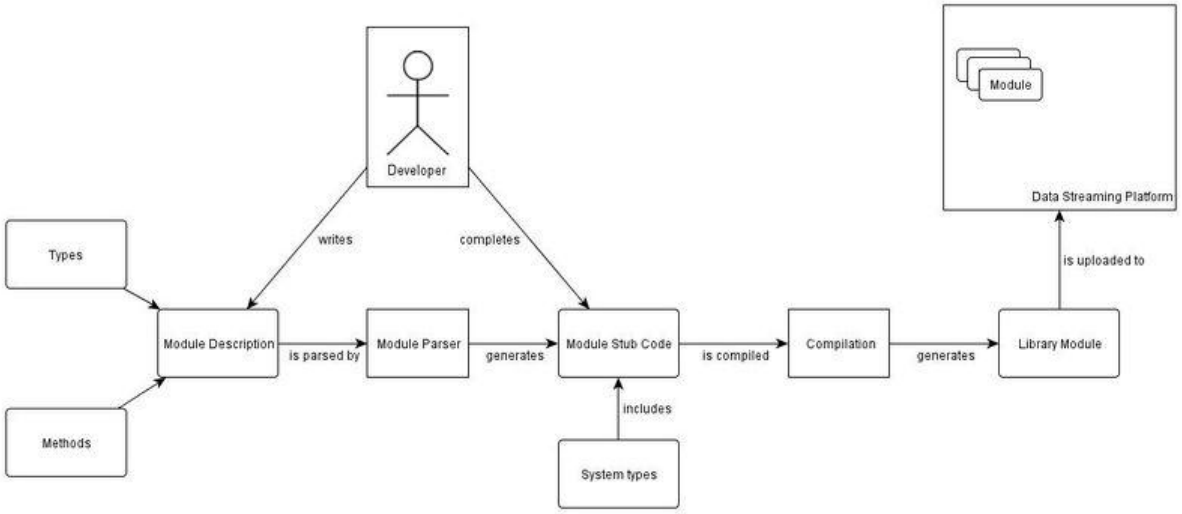


Figure BDA 6 - Developing a module

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

2.5.3.1 *System Module*

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

- Integer
- String
- Floating Point
- Date/Time

2.5.3.2 *Module Development*

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

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

Configuration File

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

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

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

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

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

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

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

```
parser parse_web_logs
{
    out web.webLog
}
```

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

Once the ***module*** definition file is complete it needs to be processed using the ***module parser*** to generate the necessary code stubs to implement the logic to parse the data. A

header file is generated for each operation defined in the ***module*** file. So with our example *parse_web_logs* a *parse_web_logs.h* is created with the necessary code need start adding the logic to ***parse*** the web logs.

Operation code

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

2.5.4 Analytical scripting

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

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

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

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

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

        # Operations;
        alias [parse_web]   01.parse_web;

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

    helpLine "Init stream operations"
}
```

2.5.4.1 *Command Line interface*

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

2.5.4.2 *General Commands*

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

2.5.4.3 *Stream Processing Commands*

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

2.5.4.4 *RESTful API*

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

Obtain the current status of the platform

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

Obtain a list of the current active operations

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

Obtain a list of the available modules

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

Obtain a list of the current queues

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

Obtain the output state of a given operation

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

Define a sequence of operations

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

2.6 Basic Design Principles

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

2.7 References

BDA1	MapReduce: Simplified Data Processing on Large Clusters
BDA2	System and method for efficient large-scale data processing
BDA3	http://hadoop.apache.org/
BDA4	http://sector.sourceforge.net/
BDA5	Sector and Sphere: the design and implementation of a high-performance data cloud
BDA6	http://research.microsoft.com/en-us/projects/Dryad/
BDA7	The Google File System
BDA8	HDFS Architecture Guide
BDA9	Dynamo: Amazon's Highly Available Key-value Store
BDA10	Cap Theorem
BDA11	Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services
BDA12	Eventually Consistent - Dr Werner Vogels
BDA13	http://couchdb.apache.org/
BDA14	http://www.mongodb.org/
BDA15	http://neo4j.org/

2.8 Detailed Specifications

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

2.8.1 Open API Specifications

- BigData Analysis Open RESTful API Specification (PRELIMINARY)

2.9 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.

- **Data** refers to information that is produced, generated, collected or observed that may be relevant for processing, carrying out further analysis and knowledge extraction. Data in FI-WARE has associated a **data type** and a**value**. FI-WARE will support a set of built-in **basic data types** similar to those existing in most programming languages. Values linked to basic data types supported in FI-WARE are referred as **basic data values**. As an example, basic data values like '2', '7' or '365' belong to the integer basic data type.
- A **data element** refers to data whose value is defined as consisting of a sequence of one or more <name, type, value> triplets referred as **data element attributes**, where the type and value of each attribute is either mapped to a basic data type and a basic data value or mapped to the data type and value of another data element.
- **Context** in FI-WARE is represented through **context elements**. A context element extends the concept of **data element** by associating an EntityId and EntityType to it, uniquely identifying the entity (which in turn may map to a group of entities) in the FI-WARE system to which the context element information refers. In addition, there may be some attributes as well as meta-data associated to attributes that we may define as mandatory for context elements as compared to data elements. Context elements are typically created containing the value of attributes characterizing a given entity at a given moment. As an example, a context element may contain values of some of the attributes "last measured temperature", "square meters" and "wall color" associated to a room in a building. Note that there might be many different context elements referring to the same entity in a system, each containing the value of a different set of attributes. This allows that different applications handle different context elements for the same entity, each containing only those attributes of that entity relevant to the corresponding application. It will also allow representing updates on set of attributes linked to a given entity: each of these updates can actually take the form of a context element and contain only the value of those attributes that have changed.
- An **event** is an occurrence within a particular system or domain; it is something that has happened, or is contemplated as having happened in that domain. Events typically lead to creation of some data or context element describing or representing the events, thus allowing them to processed. As an example, a sensor device may be measuring the temperature and pressure of a given boiler, sending a context element every five minutes associated to that entity (the boiler) that includes the value of these to attributes (temperature and pressure). The creation and sending of the context element is an event, i.e., what has occurred. Since the data/context elements that are generated linked to an event are the way events get visible in a computing system, it is common to refer to these data/context elements simply as "events".
- A **data event** refers to an event leading to creation of a data element.
- A **context event** refers to an event leading to creation of a context element.
- An **event object** is used to mean a programming entity that represents an event in a computing system [EPIA] like event-aware GEs. Event objects allow to perform operations on event, also known as **event processing**. Event objects are defined as

a data element (or a context element) representing an event to which a number of standard event object properties (similar to a header) are associated internally. These standard event object properties support certain event processing functions.

3 BigData Analysis Open RESTful API Specification (PRELIMINARY)

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

3.1 Introduction to the *BigData GE RESTful* API

Please check the FI-WARE Open Specifications Legal Notice to understand the rights to use FI-WARE Open Specifications.

3.1.1 BigData GE RESTful API Core

The BigData GE RESTful API is a RESTful, resource-oriented API access via HTTP that uses either XML or JSON based representations for information interchange. The API is designed to provide access to the following aspects of the BigData GE:

- Platform status
  - Current state of queues and operations
  - Logging
  - Loaded modules
- Managing data queues and operations
  - Programming new operations
  - Fetching results
- Uploading and downloading data

Documentation on interacting with the Apache HDFS and Mongodb RESTful interfaces can be found at:

- Apache HDFS - http://hadoop.apache.org/common/docs/r1.0.0/webhdfs.html
- MongoDB via *sleepy.mongoose* - https://github.com/kchodorow/sleepy.mongoose/wiki

3.1.2 Intended Audience

This specification is intended for use by data explorers, software developers and systems administrators. To use this information, the reader should firstly have a general understanding of the Generic Enabler service. You should also be familiar with:

- ReSTful web services
- HTTP/1.1
- JSON and/or XML data serialization formats.

3.1.3 API Change History

This version of the BigData GE RESTful API Guide replaces and obsoletes all previous versions. The most recent changes are described in the table below:

Revision Date	Changes Summary
Apr 30, 2012	• Initial Version

3.1.4 How to Read This Document

It is assumed that the reader is familiar with RESTful architectural style. The document uses the following notation:

- A bold, mono-spaced font is used to represent code or logical entities, e.g., HTTP method (GET, PUT, POST, DELETE).
- An italic font is used to represent document titles or some other kind of special text, e.g., URI.
- The variables are represented between brackets, e.g. {id} and in italic font. When the reader find it, can change it by any value.

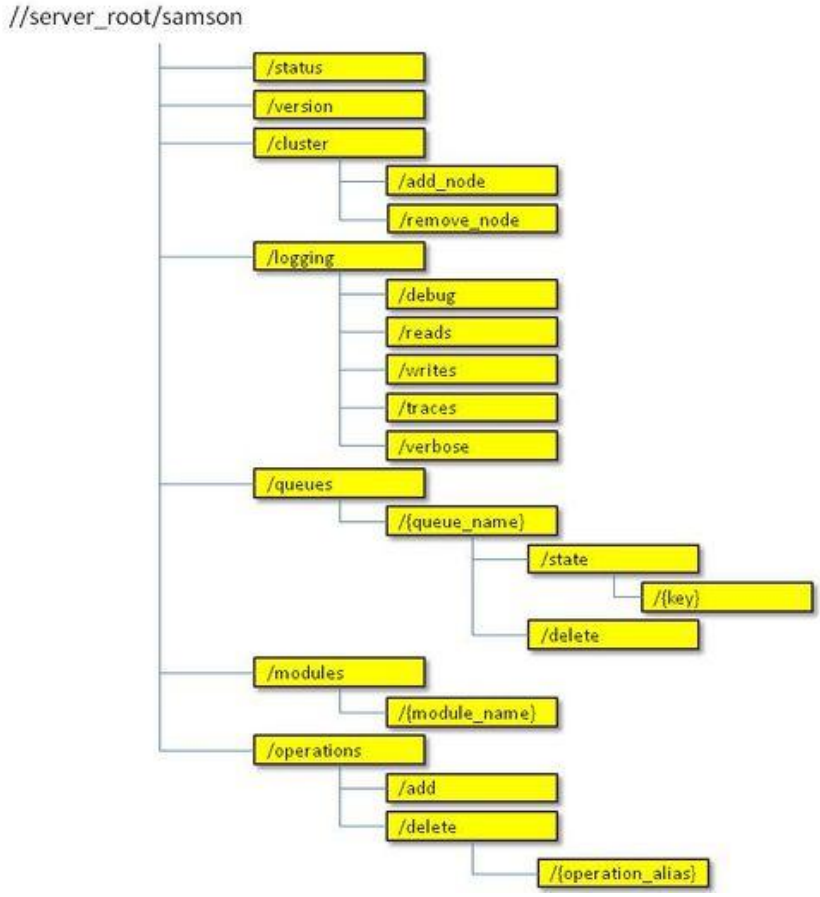
For a description of some terms used along this document, see the BigData Analysis Architecture

3.1.5 Additional Resources

You can download the most current version of this document from the FIWARE API specification website at <link to the url>. For more details about the BigDataAnalysis GE that this API is based upon, please refer to high level description. Related documents, including an Architectural Description, are available at the same site.

3.2 General *BigData GE RESTful* API Information

3.2.1 Resources Summary



3.2.2 Authentication

Each HTTP request against the *BigData GE RESTful API* requires the inclusion of specific authentication credentials. The specific implementation of this API currently supports Basic Authentication only.

3.2.3 Representation Format

The *BigData GE RESTFul API* supports both JSON and XML data formats. The request format is specified using the Content-Type header and is required for operations that have a request body. The response format can be specified in requests using either the Accept header or adding an .xml or .json extension to the request URI. Unless otherwise specified the document returned from any request will be in XML.

3.2.4 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.

3.2.5 Resource Identification

This section must explain which would be the resource identification used by the API in order to identify unambiguously the resource. For HTTP transport, this is made using the mechanisms described by HTTP protocol specification as defined by IETF RFC-2616.

3.2.6 Links and References

With the exception of the */samson/queues/{queue_name}/state/{key}* URI all requests to this GE will return a direct response without any link or redirection. Requests for */samson/queues/{queue_name}/state/{key}* are made against the node and not the cluster and as such it is possible that the value for the key is not present on the node where the request is being made. If this is the case the API will return the *Location* of the {key} via the following HTTP header:

```
HTTP/1.1 302 Found
Location:
http://{alternate_server_node}/samson/queues/{queue_name}/state/{key
}
```

It is then up to the RESTful client to make the additional request to fetch the required key value with the supplied URL.

3.2.7 Limits

There is no planned support for limits within the first release. Support for this feature is expected to be added in the next major release.

3.2.8 Versions

There are no plans to provide version information other than at the platform level (see /samson/version) in Release 1. It is expected that support for versioning will occur in a future release.

3.2.9 Faults

3.2.9.1 ***Synchronous Faults***

In this section, we provide the complete list of possible fault elements and error code, and if it is expected in all requests or not.

Fault Element	Associated Error Codes	Expected in All Requests?

GET	`400 Bad Request`, `404 Not Found`	[YES]
POST	`400 Bad Request`, `404 Not Found`	[YES]
DELETE	`400 Bad Request`, `404 Not Found`	[YES]

The response format of the fault will be provided in the format expected by the original request, i.e. an XML request will provide

3.3 API Operations

The following section provides the detail for each RESTful operation giving the expected input and output for each URI. For simplicity sake the API operations have been grouped into *Platform*, *Queues*, *Modules* and *Operations*.

3.3.1 Platform

3.3.1.1 ***General***

Verb	URI	Description
GET	/samson/status	Provide the current status of the cluster and its member nodes, equivalent to `/samson/cluster`
GET	/samson/version	Provide the current installed version

/samson/status

Return the current status of the platform, indicating which nodes make up the cluster.

As XML

```
GET /samson/status.xml HTTP/1.1
```

Sample result:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- SAMSON Rest interface -->
<samson>
  <cluster_information>
    <id><![CDATA[49204720432]]></id>
    <version><![CDATA[0]]></version>
    <cluster_node>
      <id><![CDATA[0]]></id>
      <host><![CDATA[localhost]]></host>
      <port><![CDATA[1234]]></port>
    </cluster_node>
  </cluster_information>
</samson>
```

As JSON:

```
GET /samson/status.json HTTP/1.1
```

Sample result:

```
{
  "cluster_information":
  {
    "id"      : 49204720432,
    "version" : 0,
    "cluster_node":
    {
      "id"   : "0",
      "host" : "localhost",
      "port" : "1234"
    }
  }
}
```

/samson/version

As XML:

```
GET /samson/version.xml HTTP/1.1
```

Sample result:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- SAMSON Rest interface -->
<samson>
<version><![CDATA[SAMSON v 0.6.1]]></version>
</samson>
```

As JSON:

```
GET /samson/version.json HTTP/1.1
```

Sample result:

```
{
  "version" : "SAMSON v 0.6.1"
}
```

3.3.1.2 *Cluster*

Verb	URI	Description
GET	/samson/cluster	Provide the current status of the cluster and its member nodes, equivalent to `/samson/status`
POST	/samson/cluster/add_node	Add {node} accessible via {port} to the cluster
DELETE	/samson/cluster/remove_node	Remove the {node} accessible via {port} from the cluster

/samson/cluster

See /samson/status.

/samson/cluster/add_node

Add a new node, identified by its *{name}* (hostname or IP address) and *{port}*, to the cluster.

As XML:

```
POST /samson/cluster/add_node.xml HTTP/1.1
<samson>
  <node>
    <name>{name}</name>
    <port>{port}</port>
  </node>
</samson>
```

Sample result:

```
HTTP/1.1 200 OK
```

As JSON:

```
POST /samson/cluster/add_node.json HTTP/1.1
{
  "node" :
    {
      "name" : "{name}",
      "port" : "{port}"
    }
}
```

Sample result:

```
HTTP/1.1 200 OK
```

/samson/cluster/remove_node

Remove new node, identified by its *{name}* (hostname or IP address) and *{port}*, from the cluster.

As XML:

```
DELETE /samson/cluster/remove_node.xml HTTP/1.1
<samson>
  <node>
    <name>{name}</name>
    <port>{port}</port>
  </node>
</samson>
```

Sample result:

```
HTTP/1.1 200 OK
```

As JSON:

```
DELETE /samson/cluster/remove_node.json HTTP/1.1
{
  "node" :
    {
      "name" : "{name}",
      "port" : "{port}"
    }
}
```

Sample result:

```
HTTP/1.1 200 OK
```

3.3.1.3 *Logging*

Verb	URI	Description
POST	/samson/logging/debug/on	Enable debug logging
POST	/samson/logging/debug/off	Enable debug logging
POST	/samson/logging/reads/on	Enable the logging of read operations
POST	/samson/logging/reads/off	Disable the logging of read operations
POST	/samson/logging/writes/on	Enable the logging of write operations
POST	/samson/logging/writes/off	Disable the logging of write operations
GET	/samson/logging/traces	Get a list of enabled traces
POST	/samson/logging/traces/off	Disable trace logging
POST	/samson/logging/traces/set	Enable tracing using the supplied {levels}
POST	/samson/logging/traces/add	Append tracing using the supplied {levels}
DELETE	/samson/logging/traces/remove	Delete/remove the specified tracing {levels}
GET	/samson/logging/verbose	Get the current verbose tracing level
POST	/samson/logging/verbose/off	Disable verbose tracing
POST	/samson/logging/verbose/set/{level}	Enable vebose tracing at levels 1-5

/samson/logging/debug/on

Enable debug logging.

As XML:

```
POST /samson/logging/debug/on.xml HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

As JSON:

```
POST /samson/logging/debug/on.json HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

/samson/logging/debug/off

Disable debug logging.

As XML:

```
POST /samson/logging/debug/off.xml HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

As JSON:

```
POST /samson/logging/debug/off.json HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

/samson/logging/reads/on

Enable logging of read operations.

As XML:

```
POST /samson/logging/reads/on.xml HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

As JSON:

```
POST /samson/logging/reads/on.json HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

/samson/logging/reads/off

Disable logging of read operations.

As XML:

```
POST /samson/logging/reads/off.xml HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

As JSON:

```
POST /samson/logging/reads/off.json HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

/samson/logging/writes/on

Enable logging of write operations.

As XML:

```
POST /samson/logging/writes/on.xml HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

As JSON:

```
POST /samson/logging/writes/on.json HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

/samson/logging/writes/off

Disable logging of write operations.

As XML:

```
POST /samson/logging/writes/off.xml HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

As JSON:

```
POST /samson/logging/writes/off.json HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

/samson/logging/traces

Fetch the current trace level.

As XML:

```
GET /samson/logging/traces.xml HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
<?xml version="1.0" encoding="UTF-8"?>
<!-- SAMSON Rest interface -->
<samson>
<message><![CDATA[Tracelevels: '1-5']]></message>
</samson>
```

As JSON:

```
POST /samson/logging/traces.json HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
{
  "traceLevels" : "1-5"
}
```

/samson/logging/traces/off

Disable all tracing.

As XML:

```
GET /samson/logging/traces/off.xml HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

As JSON:

```
POST /samson/logging/traces/off.json HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

/samson/logging/traces/set

Enable different trace levels using a comma separated list (e.g. 1,4,55) a range (e.g. 1-199) or combining both (e.g. 1,4,55,100-150)

As XML:

```
POST /samson/logging/traces/set.xml HTTP/1.1
<samson>
  <traces>
    <level>{levels}</level>
  </traces>
</samson>
```

Sample result:

```
HTTP/1.1 200 OK
```

As JSON:

```
POST /samson/logging/traces/set.json HTTP/1.1
{
  "traces" : "{levels}"
}
```

Sample result:

```

HTTP/1.1 200 OK
```

/samson/logging/traces/remove

Disable active trace levels using a comma separated list (e.g. 1,4,55) a range (e.g. 1-199) or combining both (e.g. 1,4,55,100-150)

As XML:

```
DELETE /samson/logging/traces/remove.xml HTTP/1.1
<samson>
  <traces>
    <level>{levels}</level>
  </traces>
</samson>
```

Sample result:

```
HTTP/1.1 200 OK
```

As JSON:

```
DELETE /samson/logging/traces/remove.json HTTP/1.1
{
  "traces" : "{levels}"
}
```

Sample result:

```
HTTP/1.1 200 OK
```

/samson/logging/verbose

Fetch the current verbose trace level.

As XML:

```
GET /samson/logging/verbose.xml HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
<?xml version="1.0" encoding="UTF-8"?>
<!-- SAMSON Rest interface -->
<samson>
<verbose><![CDATA[verbosity level: 0]]></verbose>
</samson>
```

As JSON:

```
GET /samson/logging/verbose.json HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
{
  "verbose" : "verbosity level: 0"
}
```

/samson/logging/verbose/off

Disable verbose tracing.

As XML:

```
POST /samson/logging/verbose/off.xml HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

As JSON:

```
GET /samson/logging/verbose/off.json HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

/samson/logging/verbose/set

Fetch the current verbose trace level.

As XML:

```
POST /samson/logging/verbose/set.xml HTTP/1.1
<samson>
  <verbose>{level}</verbose></verbose>
</samson>
```

Sample result:

```
HTTP/1.1 200 OK
```

As JSON:

```
POST /samson/logging/verbose/set.json HTTP/1.1
{
  "verbose" : "{level}"
}
```

Sample result:

```
HTTP/1.1 200 OK
```

3.3.2 Queues

3.3.2.1 *Status*

Verb	URI	Description
GET	/samson/queues	Provide the list of queues
GET	/samson/queues/{queue_name}/state/{key}	Provide the current state for the given {key} value in {queue_name}. If the key is not present on this node a HTTP 302 Redirect will be given with a link to the node that contains the key and its value.
GET	/samson/queues/{queue_name}	Provide detailed information about {queue_name}

/samson/queues

Provide the list of queues

As XML:

```
GET /samson/queues.xml HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
<?xml version="1.0" encoding="UTF-8"?>
<!-- SAMSON Rest interface -->
<samson>
  <queues>
    <queue>
      <name><![CDATA[quijote]]></name>
    </queue>
    <queue>
      <name><![CDATA[lines]]></name>
    </queue>
    <queue>
      <name><![CDATA[words]]></name>
    </queue>
  </queues>
</samson>
```

As JSON:

```
GET /samson/queues.json HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
{
    "queues": [
        {
            "name": "quijote"
        },
        {
            "name": "lines"
        },
        {
            "name": "words"
        }
    ]
}
```

/samson/queues/{queue_name}/state/{key}

Provide the current state for the given {key} value in {queue_name}. If the key is not present on this node a *HTTP 302 Redirect* will be given with a link to the node that contains the key and its value. Requests for */samson/queues/{queue_name}/state/{key}* are made against the node and not the cluster and as such it is possible that the value for the key is not present on the node where the request is being made. If this is the case the API will return the *Location* of the {key} via the following HTTP header:

```
HTTP/1.1 302 Found
```

```
Location:
http://{alternate_server_node}/samson/queues/{queue_name}/state/{key
}
```

It is then up to the RESTful client to make the additional request to fetch the required key value with the supplied URL.

As XML:

```
GET /samson/queues/{queue_name}/state/{key}.xml HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
<?xml version="1.0" encoding="UTF-8"?>
<!-- SAMSON Rest interface -->
<samson>
<key><![CDATA[top_categories]]></key>
<value>
  <hits>
    <concept><![CDATA[top_categories
778218_Corporate_Presence]]></concept>
    <count>15970.3</count>
    <time>1334759683</time>
  </hits>
  <hits>
    <concept><![CDATA[top_categories 778220_Education]]></concept>
    <count>13439.2</count>
    <time>1334759683</time>
  </hits><hits><concept><![CDATA[top_categories
778227_Portals]]></concept>
    <count>13161.1</count>
    <time>1334759683</time>
  </hits>
  <hits>
    <concept><![CDATA[top_categories
778237_Business_to_Business]]></concept>
    <count>11745.4</count>
    <time>1334759683</time>
  </hits>
  <hits>
  <concept><![CDATA[top_categories 778235_Services]]></concept>
    <count>4937.45</count>
    <time>1334759683</time>
  </hits>
  <hits>
    <concept><![CDATA[top_categories
4622803_Social_Media]]></concept>
    <count>1445.49</count>
    <time>1334759683</time>
  </hits>
  <hits>
    <concept><![CDATA[top_categories
4623956_Social_Networking]]></concept>
    <count>1388.57</count>
```

```
    <time>1334759683</time>
  </hits><hits><concept><![CDATA[top_categories
778226_News/Information]]></concept>
    <count>822.991</count>
    <time>1334759683</time>
  </hits>
  <hits>
    <concept><![CDATA[top_categories 778233_Technology]]></concept>
    <count>735.56</count>
    <time>1334759683</time>
  </hits>
  <hits>
    <concept><![CDATA[top_categories
778320_Promotional_Servers]]></concept>
    <count>729.077</count>
    <time>1334759683</time>
  </hits>
  <hits>
    <concept><![CDATA[top_categories 778230_Retail]]></concept>
    <count>612.822</count>
    <time>1334759683</time>
  </hits>
  <hits>
    <concept><![CDATA[top_categories 4962873_Newspapers]]></concept>
    <count>592.289</count>
    <time>1334759683</time>
  </hits>
  <hits>
    <concept><![CDATA[top_categories 778300_Web_Hosting]]></concept>
    <count>477.965</count>
    <time>1334759683</time>
  </hits>
  <hits>
    <concept><![CDATA[top_categories
778219_Directories/Resources]]></concept>
    <count>337.209</count>
    <time>1334759683</time>
  </hits>
  <hits>
    <concept><![CDATA[top_categories 778217_Community]]></concept>
    <count>321.315</count>
    <time>1334759683</time>
  </hits>
  <hits>
    <concept><![CDATA[top_categories 778256_Reference]]></concept>
    <count>269.56</count>
    <time>1334759683</time>
  </hits>
</value>
</samson>
```

As JSON:

```
GET /samson/queues/{queue_name}/state/{key}.json HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

/samson/queues/{queue_name}

Provide detailed information (size, number of key values, keys (if present)) for a given {queue_name}

As XML:

```
GET /samson/queues/{queue_name}.xml HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
<?xml version="1.0" encoding="UTF-8"?>
<!-- SAMSON Rest interface -->
<samson>
  <queue>
    <name><![CDATA[quijote]]></name>
    <key_values>0</key_values>
    <bytes>2071203</bytes>
    <total_key_values>0</total_key_values>
    <total_bytes>2071203</total_bytes>
    <key_values_sec>0</key_values_sec>
    <bytes_sec>144000</bytes_sec>
    <key><![CDATA[txt]]></key>
    <value><![CDATA[txt]]></value>
  </queue>
</samson>
```

As JSON:

```
GET /samson/queues/{queue_name}.json HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
{
  "queue" :
  {
      "name" : "quijote",
      "key_values" : 0,
      "bytes" : 2071203,
      "total_key_values" : 0,
      "total_bytes" : 2071203,
      "key_values_sec" : 0,
      "bytes_sec" : 144000,
      "key" : "system.Value",
      "value" : "system.Value"
  }
}
```

3.3.2.2 *Deletion*

Verb	URI	Description
DELETE	/samson/queues/{queue_name}/delete	Delete the supplied {queue_name}

/samson/queue/{queue_name}/delete

Delete the named queue.

As XML:

```
DELETE /samson/queues/{queue_name}/delete.xml HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

As JSON:

```
DELETE /samson/queues/{queue_name}/delete.xml HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

3.3.3 Modules

3.3.3.1 *Status*

Verb	URI	Description
GET	/samson/modules	Provide the list of the installed analytical modules
GET	/samson/modules/{module_name}	Provide information (defined types, operations and version) about {module_name}

/samson/modules

Provide the list of the installed analytical modules

As XML:

```
GET /samson/modules.xml HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
<?xml version="1.0" encoding="UTF-8"?>
<!-- SAMSON Rest interface -->
<samson>
  <modules>
    <module>
```

```
      <name><![CDATA[system]]></name>
    </module>
    <module>
      <name><![CDATA[webp]]</name>
    </module>
  </modules
</samson>
```

As JSON:

```
GET /samson/modules.json HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
{
    "modules": [
        "system",
        "txt",
        "web",
        "webp"
    ]
}
```

/samson/modules/{module_name}

Provide information (defined types, operations and version) about {module_name}

As XML:

```
GET /samson/modules/{module_name}.xml HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
<?xml version="1.0" encoding="UTF-8"?>
<!-- SAMSON Rest interface -->
<samson>
  <module>
    <name><![CDATA[system]]></name>
    <version><![CDATA[0.2]]</version>
    <author><![CDATA[Andreu Urruela]]</author>
    <operations>
      <operation>
        <name><![CDATA[system.map]]</name>
        <type><![CDATA[map]]></type>
        <inputs>
          <input><![CDATA[system.Value-system.Value]]></input>
        </inputs>
        <outputs>
          <output><![CDATA[system.Value-system.Value]]></output>
        </output>
      </operation>
      <operation>
```

```
        <name><![CDATA[system.parse]]</name>
        <type><![CDATA[parser]]></type>
        <inputs>
          <input><![CDATA[system.Value-system.Value]]></input>
        </inputs>
        <outputs>
          <output><![CDATA[system.Value-system.Value]]></output>
        </output>
      </operation>
      <operation>
        <name><![CDATA[system.reduce]]</name>
        <type><![CDATA[reduce]]></type>
        <inputs>
          <input><![CDATA[system.Value-system.Value]]></input>
        </inputs>
        <outputs>
          <output><![CDATA[system.Value-system.Value]]></output>
        </output>
      </operation>
      .
      .
      .
    </operations>
    <datatypes>
      <datatype><![CDATA[system.Date]]></datatype>
      <datatype><![CDATA[system.Double]]></datatype>
      <datatype><![CDATA[system.Int32]]></datatype>
      <datatype><![CDATA[system.String]]></datatype>
    </datatypes>
  </module>
</samson>
```

As JSON:

```
GET /samson/modules/{module_name}.json HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
{
    "module": {
        "system": {
            "version": "0.2",
            "author": "Andreu Urrela",
            "operations": [
                {
                    "name": "system.map",
                    "type": "map",
                    "inputs": [
                        "system.Value-system.Value"
                    ],
                    "outputs": [
                        "system.Value-system.Value"
```

```
                    ]
                },
                {
                    "name": "system.parse",
                    "type": "parse",
                    "inputs": [
                        "system.Value-system.Value"
                    ],
                    "outputs": [
                        "system.Value-system.Value"
                    ]
                },
                {
                    "name": "system.reduce",
                    "type": "reduce",
                    "inputs": [
                        "system.Value-system.Value"
                    ],
                    "outputs": [
                        "system.Value-system.Value"
                    ]
                }
            ],
            "datatypes": [
                "system.Date",
                "system.Double",
                "system.Int32",
                "system.String"
            ]
        }
    }
}
```

3.3.4 Operations

3.3.4.1 *Status*

Verb	URI	Description
GET	/samson/operations	List the current active operations
GET	/samson/operations/{operation_name}	Provide information about {operation_name}

/samson/operations

List the current active operations

As XML:

```
GET /samson/operations.xml HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
<?xml version="1.0" encoding="UTF-8"?>
<!-- SAMSON Rest interface -->
<samson>
  <streaming_operations>
    <streaming_operation>
      <name><![CDATA[webp.emit_hits]]></name>
      <operation><![CDATA[webp.emit_hits]]></operation>
    </streaming_operation>
    <streaming_operation>
      <name><![CDATA[webp.hit_1hour.01.reduce_hits]]></name>
      <operation><![CDATA[hit.reduceHits]]></operation>
    </streaming_operation>
    <streaming_operation>
      <name><![CDATA[webp.hit_1hour.20.reduce_tops]]></name>
      <operation><![CDATA[hit.reduceHitCollections]]></operation>
    </streaming_operation>
    <streaming_operation>
      <name><![CDATA[webp.parse]]></name>
      <operation><![CDATA[webp.parse_web_logs]]></operation>
    </streaming_operation>
  </streaming_operations>
</samson>
```

As JSON:

```
GET /samson/operations.json HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
{
  "streaming_operations" :
```

```
  [
    {
      "name" : "webp.emit_hits",
      "operation" : "webp.emit_hits"
    },
    {
      "name" : "webp.hit_1hour.01.reduce_hits",
      "operation" : "hit.reduceHits"
    },
    {
      "name" : "webp.hit_1hour.20.reduce_tops",
      "operation" : "hit.reduceHitCollections"
    },
    {
      "name" : "webp.parse",
      "operation" : "webp.parse_web_logs"
    }
  ]
}
```

/samson/operations/{operation_name}

List the details of a currently active operation.

As XML:

```
GET /samson/operations/{operation_name}.xml HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
<?xml version="1.0" encoding="UTF-8"?>
<!-- SAMSON Rest interface -->
<samson>
  <streaming_operations>
    <streaming_operation>
      <name><![CDATA[webp.emit_hits]]></name>
      <operation><![CDATA[webp.emit_hits]]></operation>
      <operation_count>22600</operation_count>
      <in>
        <bytes>907234089</bytes>
        <key_values>6390862</key_values>
        <bytes_sec>22504</bytes_sec>
        <key_values_sec>388</key_values_sec>
      </in>
      <out>
        <bytes>109345</bytes>
        <key_values>6342010</key_values>
        <bytes_sec>22504</bytes_sec>
        <key_values_sec>388</key_values_sec>
      </out>
    </streaming_operation>
  </streaming_operations>
```

```
</samson>
```

As JSON:

```
GET /samson/operations/{operation_name}.json HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
{
  "streaming_operation" :
  {
    "name" : "webp.emit_hits",
    "operation" : "webp.emit_hits",
    "operation_count" : 22600,
    "in" :
    {
      "bytes" : 907234089,
      "key_values" : 6390862,
      "bytes_sec" : 22504,
      "key_values_sec" : 388
    },
    "out" :
    {
      "bytes" : 109345,
      "key_values" : 6342010,
      "bytes_sec" : 22504,
      "key_values_sec" : 388
    },
  },
}
```

3.3.4.2 *Management*

Verb	URI	Description
PUT	/samson/operations/add	Define a new operation
DELETE	/samson/operations/delete/{operation_name}	Remove a previously defined {operation_name}

/samson/operations/add

Add an operation to the platform assigning an {operation_alias} to the {operation_name} with the input and output queues and parameters needed execute the operation.

As XML:

```
PUT /samson/operations/add.xml HTTP/1.1
<samson>
  <operation>
    <alias>{operation_alias}</alias>
    <name>{operation_name}</name>
    <input_queues>
      <input>queue1</input>
      <input>queue2</input>
    </input_queues>
    <output_queues>
      <output>queue3</output>
    </output_queues>
    <parameters>
      <parameter>
        <parameter_name></parameter_name>
        <parameter_value></parameter_value>
      </parameter>
      <parameter>
        <parameter_name></parameter_name>
        <parameter_value></parameter_value>
      </parameter>
    </parameters>
  </operation>
<samson>
```

Sample result:

```
HTTP/1.1 200 OK
```

As JSON:

```
PUT /samson/operations/add.json HTTP/1.1
{
  "operation" :
  {
    "alias" : "{operation_alias}",
    "name" : "{operation_name}",
    "input_queues" : ["queue1", "queue2"],
    "output_queues" : ["queue3"],
    "parameters" : [ {"{parameter_name}", "{parameter_value}"},
{"{parameter_name}", "{parameter_value}"} ]
  }
```

Sample result:

```
HTTP/1.1 200 OK
```

/samson/operations/delete

Delete the supplied {operation_alias}.

As XML:

```
DELETE /samson/operations/delete/{operation_alias}.xml HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

As JSON:

```
DELETE /samson/operations/delete/{operation_alias}.json HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

4 FIWARE OpenSpecification Data CEP

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

Name	FIWARE.OpenSpecification.Data.CEP
Chapter	Data/Context Management,
Catalogue-Link to Implementation	<Complex Event Processing>
Owner	IBM Haifa Research Lab, Tali Yatzkar-Haham

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.eu and similar pages in order to understand the complete context of the FI-WARE project.

4.1.1 Copyright

- Copyright © 2012 by IBM

4.1.2 Legal Notice

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

4.2 Overview

4.2.1 Introduction to the CEP GE

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

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

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

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

They can provide events in two modes:

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

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

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

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

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

- Distributing the processing power

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

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

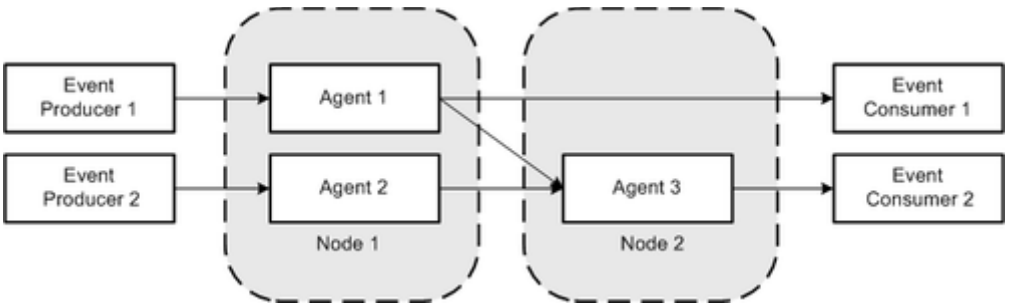


Illustration of an Event Processing Network made of producers, agents and consumers

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

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

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

4.2.1.1 *Pattern Detection*

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

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

- All, meaning that all the events specified should arrive for the pattern to match. E.g., wait to get status events from all the 4 locations, where each status event arrives with the quantity of reservations. Alert if the total reservations are higher than some threshold.

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

4.2.1.2 *Conditions*

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

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

4.2.1.3 *Actions*

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

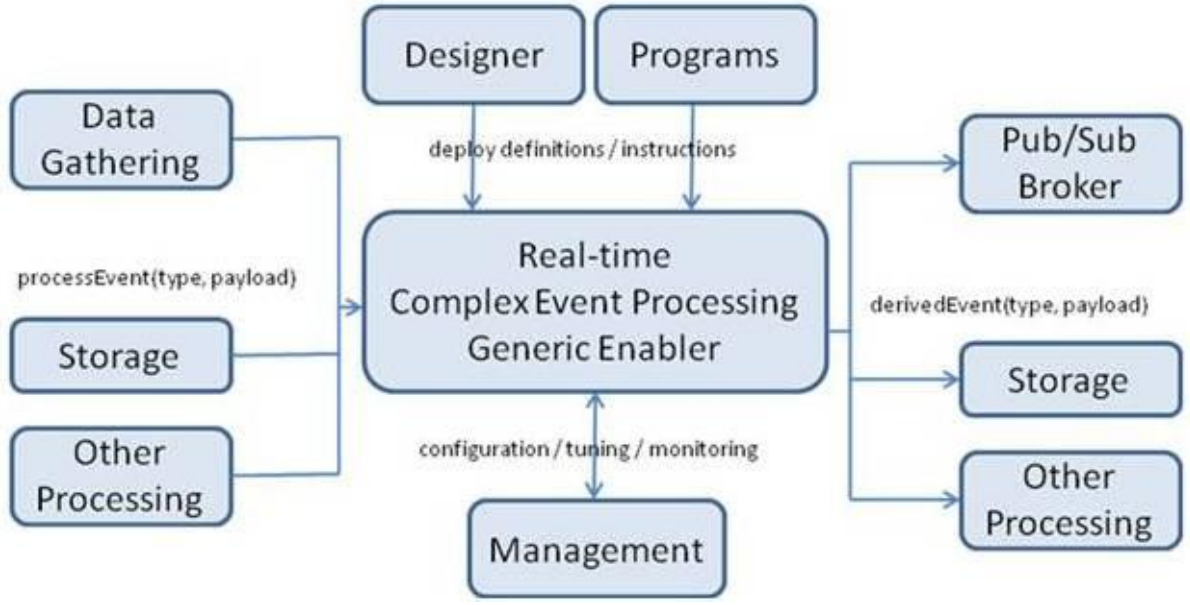
4.2.2 Target Usage

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

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

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

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



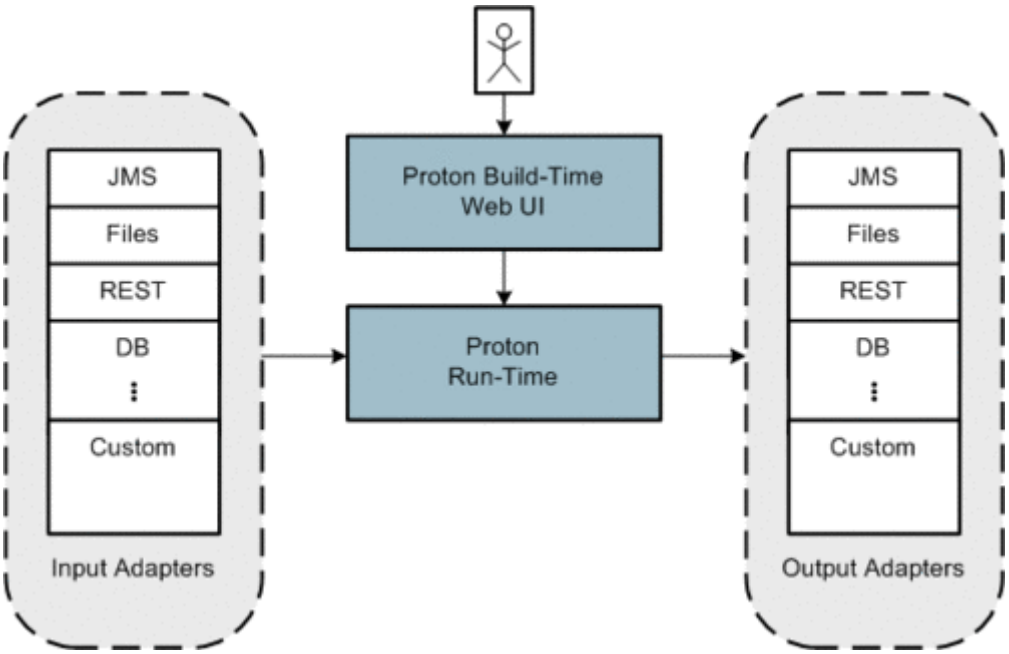
Interactions with and APIs of the Real-time CEP Generic Enabler

4.3 Basic Concepts

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

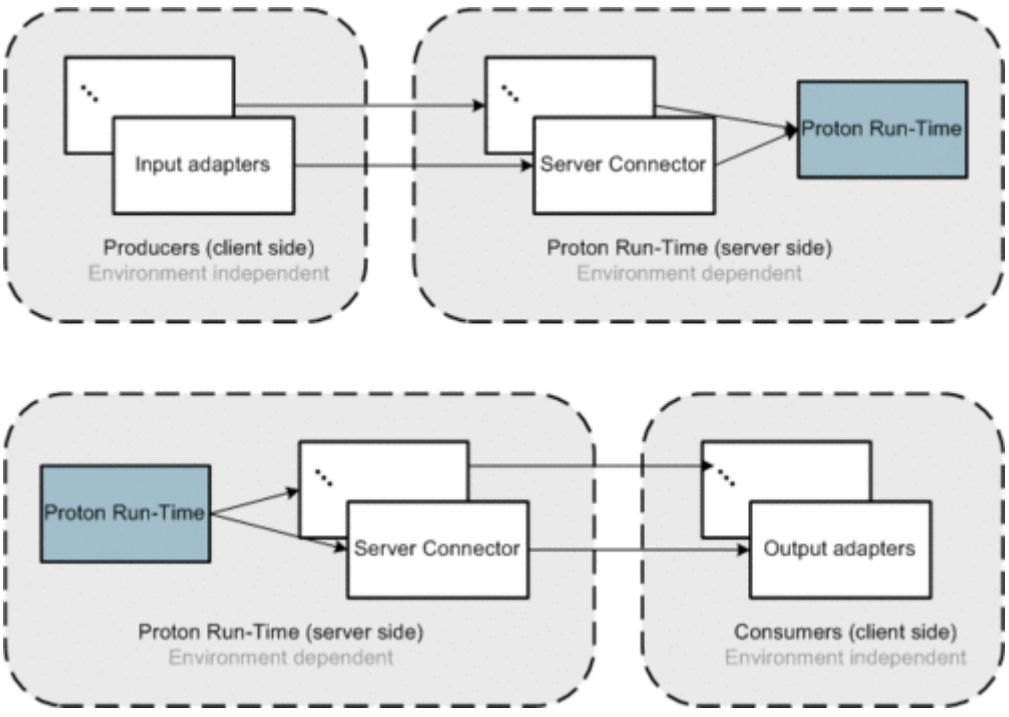
- Input Adapters for getting incoming events
- Output Adapters for sending derived events
- CEP Application definition

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



CEP High Level Architecture

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



Adapters layer representation

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

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

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

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

4.3.1 Adapters design principles

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

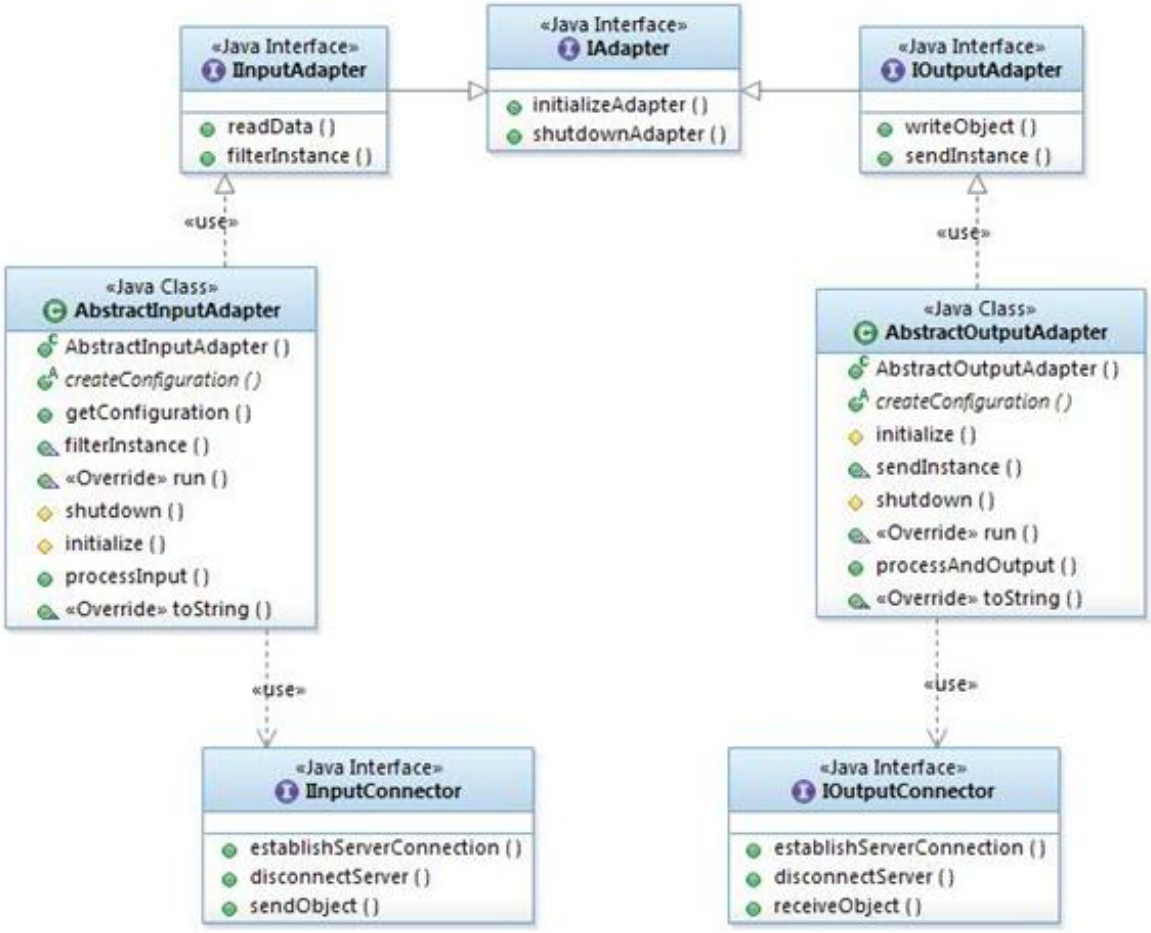
The design of adapter's layer satisfies the following principles:

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

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

4.3.2 Adapters design

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



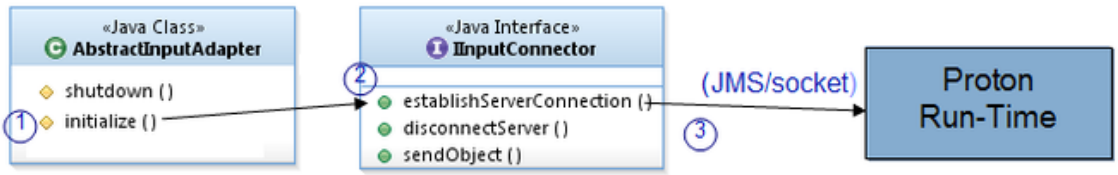
Adapters framework class diagram

4.4 Main Interactions

4.4.1 Definition of Input Adapters

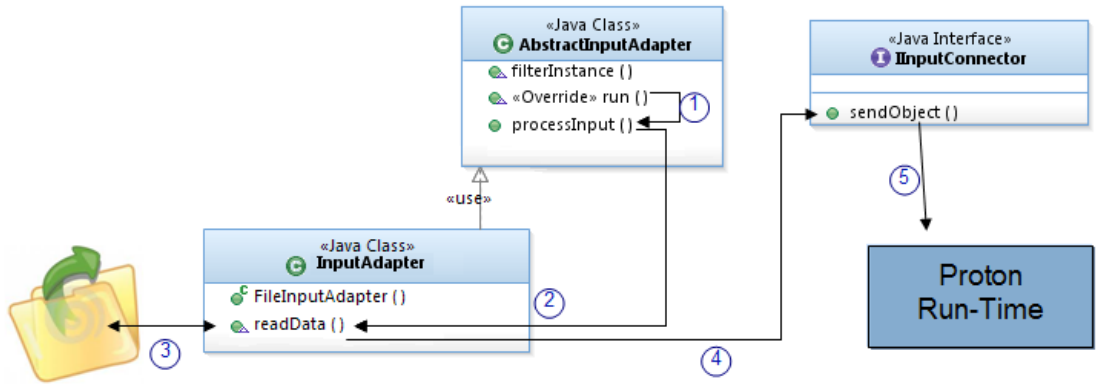
4.4.1.1 *Runtime*

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



Adapter initialization sequence-establishing connection to CEP server

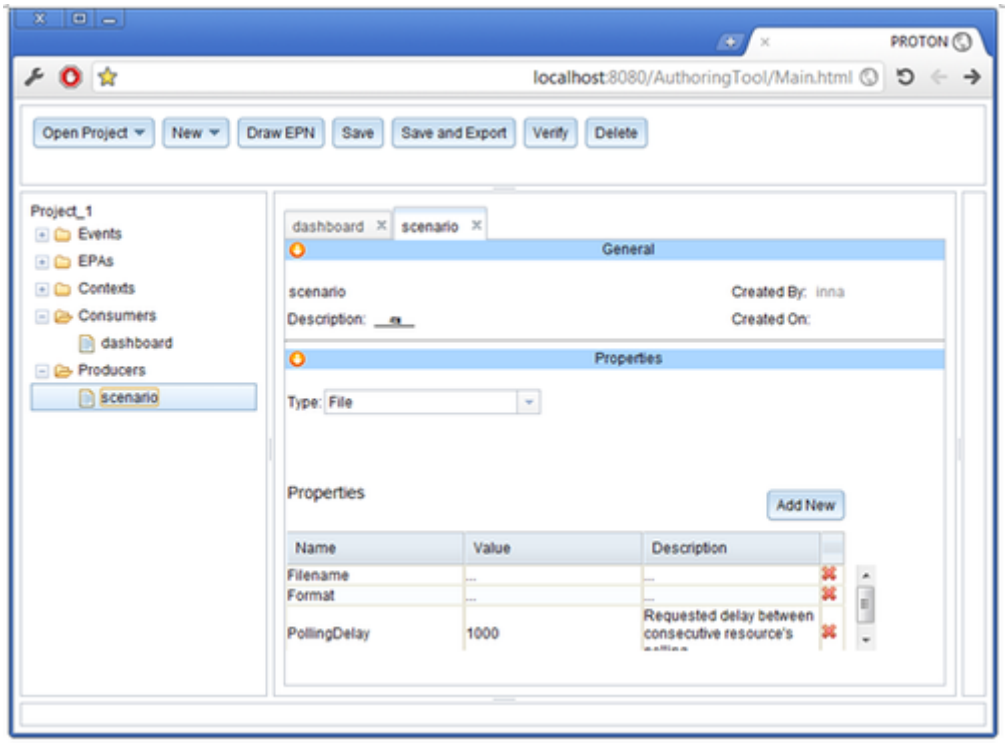
Once the adapter starts running (see below a diagram for processing sequence), it constantly polls the data source for changes (1,2), transforms an entry within the data source into CEP readable event object (3), and sends the information via server connector to the server (4), without being aware of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



Input adapter processing sequence

4.4.1.2 *Definition*

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



Buildtime representation of a producer definition

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

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

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

Producer of file type JSON definition

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

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

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

Producer of JMS type JSON definition

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

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

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


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

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


Producer of type JMS formatted text message JSON definition

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

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

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

Producer of REST type JSON definition

4.4.1.3 *Interfaces to implement for custom adapter*

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

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

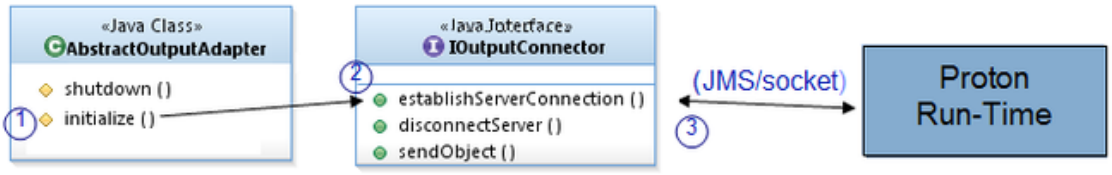
- ***IInputAdapterConfiguration createConfiguration(ProducerMetadata metadata)*** - creates the adapter-specific configuration object after extracting the relevant properties from the producer metadata object
- ***void initializeAdapter()*** - a method existing in the abstract interface but which should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

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

4.4.2 Definition of Ouput Adapters

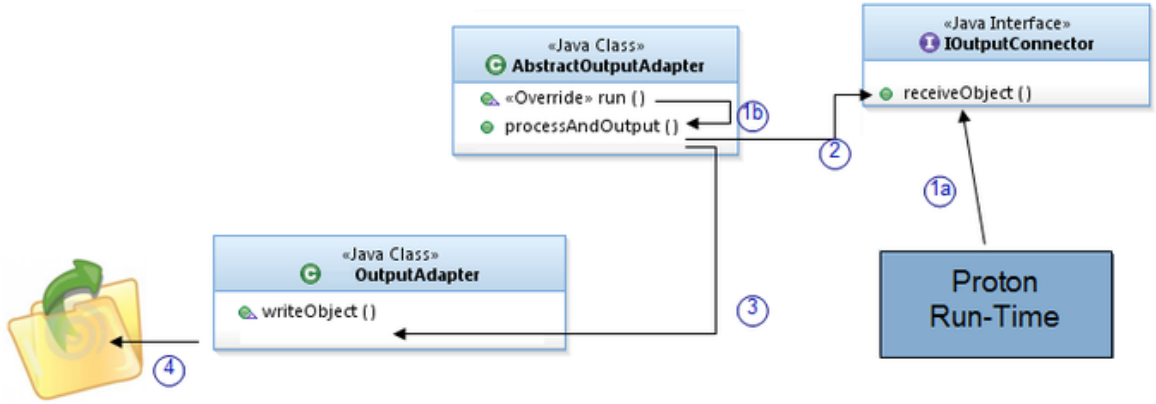
4.4.2.1 *Runtime*

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



Adapter initialization sequence-establishing connection to CEP server

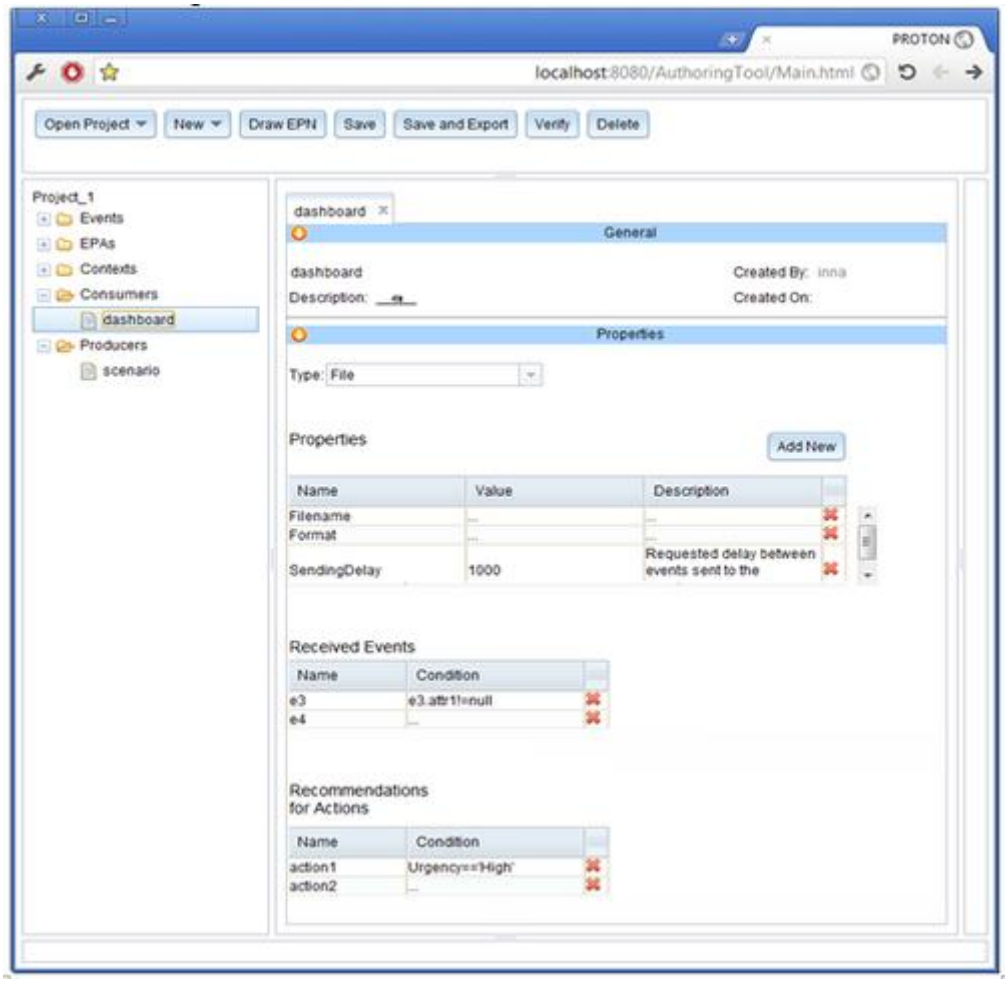
The CEP runtime pushes all published events for the specific consumer to the consumer's connector object, where it is stored in the queue. (see step 1a in the figure below). The output adapter accesses the queue and pulls the event objects from the queue. Once the adapter starts running (see the processing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



Output adapter processing sequence

4.4.2.2 *Definition*

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



Buildtime representation of a consumer definition

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

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

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

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

Consumer of file type JSON definition

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

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

Consumer of type JMS object message JSON definition

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

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

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

Consumer of type JMS formatted text message JSON definition

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

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

Consumer of REST type JSON definition

4.4.2.3 *Interfaces to implement for custom adapter*

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

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

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

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

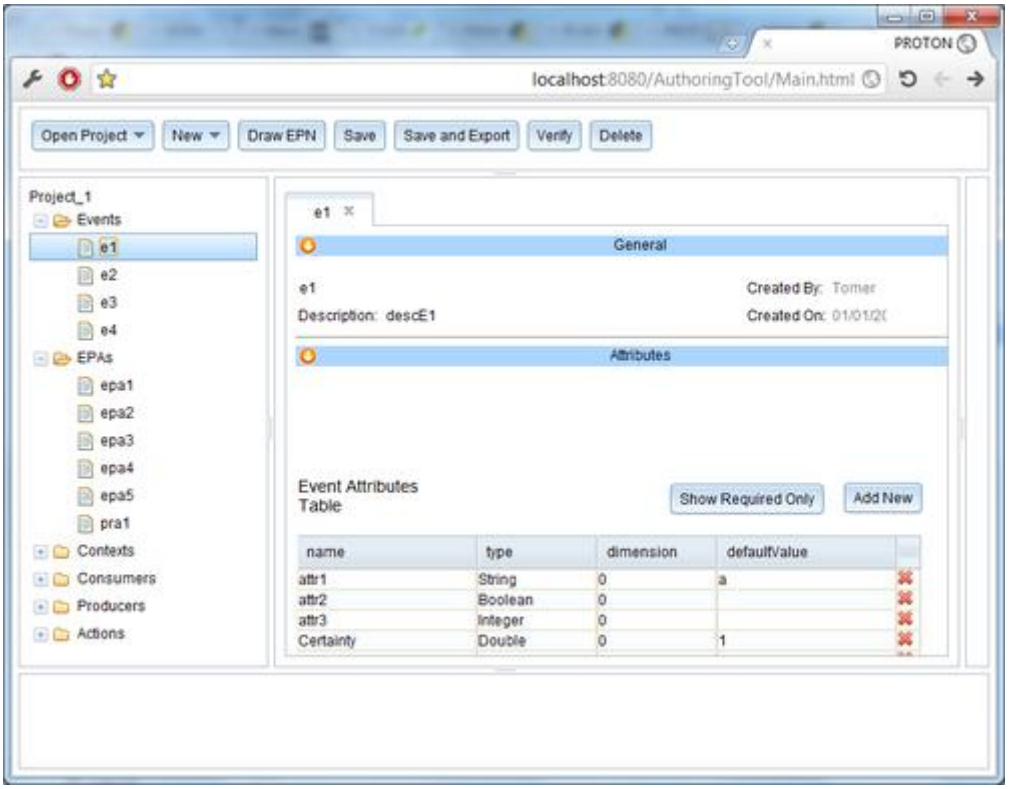
4.4.3 Definition of CEP Application

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

The building blocks of a CEP application are:

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

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



CEP Web based User interface for application definition

4.5 Basic Design Principles

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

4.6 References

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

4.7 Detailed Specifications

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

4.7.1 Open API Specifications

- Complex Event Processing Open RESTful API Specification (PRELIMINARY)

4.8 Re-utilised Technologies/Specifications

The CEP authoring tool is running on Apache Tomcat web server

4.9 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.

- **Data** refers to information that is produced, generated, collected or observed that may be relevant for processing, carrying out further analysis and knowledge extraction. Data in FI-WARE has associated a **data type** and a**value**. FI-WARE will support a set of built-in **basic data types** similar to those existing in most programming languages. Values linked to basic data types supported in FI-WARE are referred as **basic data values**. As an example, basic data values like ‘2’, ‘7’ or ‘365’ belong to the integer basic data type.
- A **data element** refers to data whose value is defined as consisting of a sequence of one or more <name, type, value> triplets referred as **data element attributes**, where the type and value of each attribute is either mapped to a basic data type and a basic data value or mapped to the data type and value of another data element.
- **Context** in FI-WARE is represented through **context elements**. A context element extends the concept of **data element** by associating an EntityId and EntityType to it,

uniquely identifying the entity (which in turn may map to a group of entities) in the FI-WARE system to which the context element information refers. In addition, there may be some attributes as well as meta-data associated to attributes that we may define as mandatory for context elements as compared to data elements. Context elements are typically created containing the value of attributes characterizing a given entity at a given moment. As an example, a context element may contain values of some of the attributes “last measured temperature”, “square meters” and “wall color” associated to a room in a building. Note that there might be many different context elements referring to the same entity in a system, each containing the value of a different set of attributes. This allows that different applications handle different context elements for the same entity, each containing only those attributes of that entity relevant to the corresponding application. It will also allow representing updates on set of attributes linked to a given entity: each of these updates can actually take the form of a context element and contain only the value of those attributes that have changed.

- An **event** is an occurrence within a particular system or domain; it is something that has happened, or is contemplated as having happened in that domain. Events typically lead to creation of some data or context element describing or representing the events, thus allowing them to processed. As an example, a sensor device may be measuring the temperature and pressure of a given boiler, sending a context element every five minutes associated to that entity (the boiler) that includes the value of these to attributes (temperature and pressure). The creation and sending of the context element is an event, i.e., what has occurred. Since the data/context elements that are generated linked to an event are the way events get visible in a computing system, it is common to refer to these data/context elements simply as "events".
- A **data event** refers to an event leading to creation of a data element.
- A **context event** refers to an event leading to creation of a context element.
- An **event object** is used to mean a programming entity that represents an event in a computing system [EPIA] like event-aware GEs. Event objects allow to perform operations on event, also known as **event processing**. Event objects are defined as a data element (or a context element) representing an event to which a number of standard event object properties (similar to a header) are associated internally. These standard event object properties support certain event processing functions.

5 Complex Event Processing Open RESTful API Specification (PRELIMINARY)

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

5.1 Introduction to the CEP GE REST API

As described in the previous chapter, CEP has three main interfaces: one for getting input events using input adapters, second for sending output events using output adapters and a third for getting an application specific definitions. In the first release of the CEP GE, it is planned to support the first two interfaces as a RESTful, resource oriented API services as a client. That is, the CEP GE can activate RESTful services of other GEs or of external applications for receiving input events and for sending output events using its input and output adapters. In the second release we plan to add the CEP GE RESTful service for getting input events in a push mode as well. In addition, we plan to add RESTful services for administrating the CEP engine (start, stop, update application definitions).

5.2 Introduction to the *CEP* API

5.2.1 CEP API

The CEP supports a RESTful, resource-oriented API accessed via HTTP that uses either JSON-based or tag-delimited format representations for getting input events and for sending events. The CEP GE uses a RESTful client input adapter which is capable to access a RESTful web service and pull input events using its GET method and a RESTful client output adapter which is capable to access a RESTful web service and push output events using its PUT method.

5.2.2 Intended Audience

This specification is intended for software developers that want to use the CEP GE allowing it to get incoming events and send output events. To use this information, the reader should have a general understanding of the Generic Enabler service FIWARE.ArchitectureDescription.Data.CEP. You should also be familiar with:

- ReSTful web services
- HTTP/1.1
- JSON or tag-delimited data serialization formats.

5.2.3 API Change History

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

Revision Date	Changes Summary
Apr 30, 2012	• Initial Version
...	• ...

5.2.4 How to Read This Document

In the whole document it is taken the assumption that reader is familiarized with REST architecture style. Along the document, some special notations are applied to differentiate some special words or concepts. The following list summarizes these special notations.

- A bold, mono-spaced font is used to represent code or logical entities, e.g., HTTP method (GET, PUT, POST, DELETE).
- An italic font is used to represent document titles or some other kind of special text, e.g., URI.
- The variables are represented between brackets, e.g. {id} and in italic font. When the reader find it, can change it by any value.

For a description of some terms used along this document, see FIWARE.ArchitectureDescription.Data.CEP.

5.2.5 Additional Resources

You can download the most current version of this document from the FIWARE API specification website at <link to the url>. For more details about the CEP GE Adapters that this API is based upon, please refer to FIWARE.ArchitectureDescription.Data.CEP. Related documents, including an Architectural Description, are available at the same site.

5.3 General CEP API Information

5.3.1 Resources Summary

The CEP GE uses a REST input adapter that activates a REST service as a client, allowing the CEP GE REST input adapter to access the REST web service declared by the event producer and pull events using the GET method. The CEP GE uses a REST output adapter that activates a REST service as a client, allowing the CEP GE REST output adapter to access the REST web service declared by the event consumer and push events using the PUT method.

5.3.2 Representation Format

The CEP REST adapters supports JSON-based or tag-delimited formats for the received input event. The format is specified in the CEP producer definition (for pulling input events from it) and in the CEP consumer definition (for pushing output events to it) and is passed as part of the requests using the Content-Type header.

5.3.3 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.

5.4 API Operations

In this section we go in depth for each operation. In order to provide good comprehensive of the API operations, we would suggest to group them into similar functionalities within subsections. e.g. operations related to VM management in the Cloud Chapter or Context Entity Management in IoT.

5.4.1 Getting Events API

The CEP activates REST API for getting incoming events in a pull mode. The CEP plays as a REST client in this API:

Verb	URI example	Description
GET	/application-name/producer	Retrieve all available incoming events

/application-name/producer

Retrieve all available incoming events from a producer.

In tag-delimited format:

```
GET /application-name/producer
Accept: text/plain
```

Sample result:

```
Name=ShipPosition;ShipID=RTX33;Long=46;Lat=55;Speed=4.0;Time=1333033
200;
Name=ShipPosition;ShipID=JF166;Long=47;Lat=55;Speed=2.0;Time=1333033
260;
```

In JSON format:

```
GET /application-name/producer
Accept: application/json
```

Sample result (for a different event):

```
{"Name":"TrafficReport", "volume":"1000"}
```

5.4.2 Sending Events API

The CEP activates REST API for sending output events (in a push mode). The CEP plays as a REST client in this API:

Verb	URI example	Description
PUT	/application-name/consumer	Send an output event to a consumer

/application-name/consumer

Send an output event to a consumer

In tag-delimited format:

```
PUT /application-name/consumer
Content-type:text/plain
Name=TrafficReport;Certainty=0.0;Cost=0.0;EventSource=;OccurrenceTim
e=null;Annotation=;Duration=0.0;volume=1000;
EventId=40f68052-3c7c-4245-ae5a-
6e20def2e618;ExpirationTime=null;Chronon=null;DetectionTime=13491818
99221;
```

In JSON format:

```
PUT /application-name/consumer
Content-type:application/json
{"Cost":"0.0","Certainty":"0.0","Name":"TrafficReport","EventSource"
:"","Duration":"0.0","Annotation":"",
 "volume":"1000","EventId":"e206b5e8-9f3a-4711-9f46-
d0e9431fe215","DetectionTime":"1350311378034"}
```

6 FIWARE OpenSpecification Data CompressedDomainVideoAnalysis

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

Name	FIWARE.OpenSpecification.Data.CompressedDomainVideoAnalysis
Chapter	Data/Context Management,
Catalogue-Link to Implementation	<Compressed Domain Video Analysis>
Owner	Siemens AG, Marcus Laumer

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.

6.1.1 Copyright

- Copyright © 2012 by SIEMENS

6.1.2 Legal Notice

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

6.2 Overview

6.2.1 Introduction to the Compressed Domain Video Analysis GE

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

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

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

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

6.2.2 Target Usage

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

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

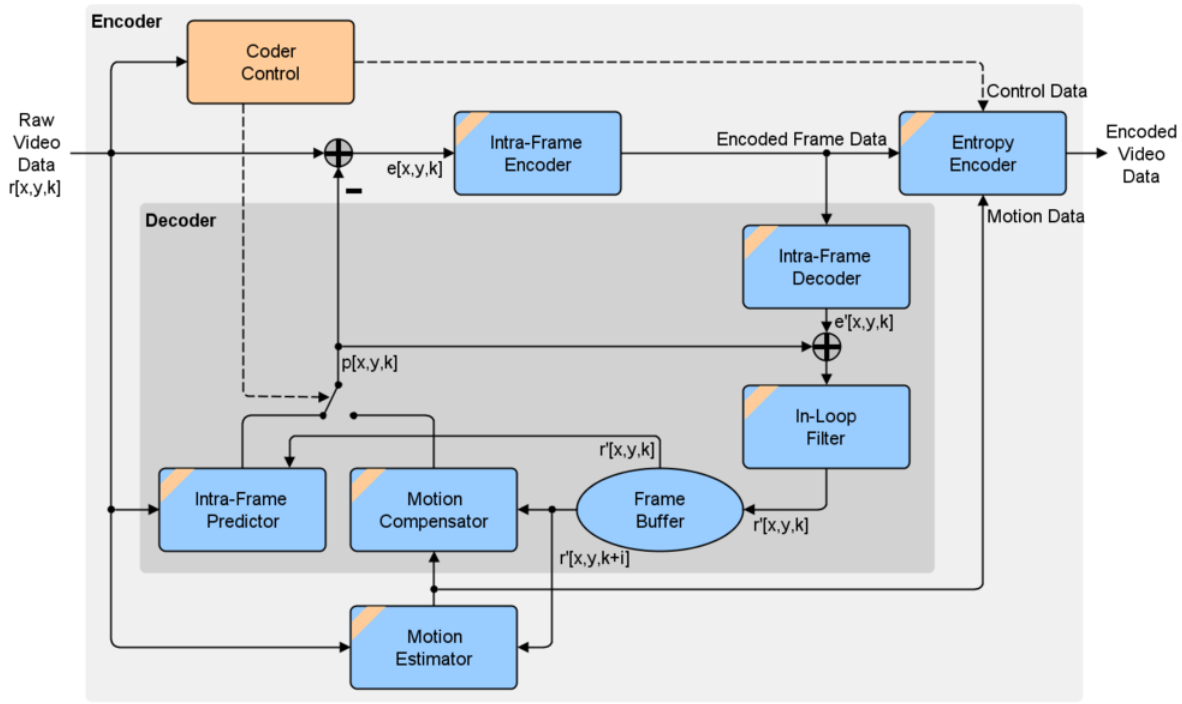
Example applications in different industries addressed by this Generic Enabler are:

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

6.3 Basic Concepts

6.3.1 Block-Based Hybrid Video Coding

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



Block diagram of a block-based hybrid video coder

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

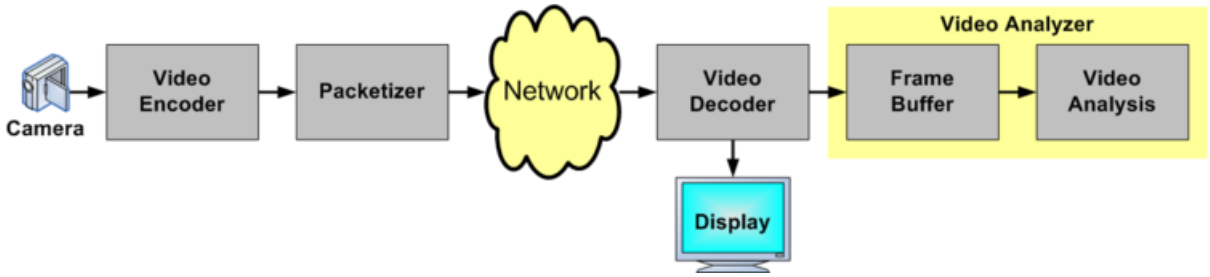
- **Coder Control**: Controls all other components to fulfill pre-defined stream properties, like a certain bit rate or quality. (Indicated by colored block corners)
- **Intra-Frame Encoder**: This component usually performs a transform to the frequency domain, followed by quantization and scaling of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

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

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

6.3.2 Compressed Domain Video Analysis

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



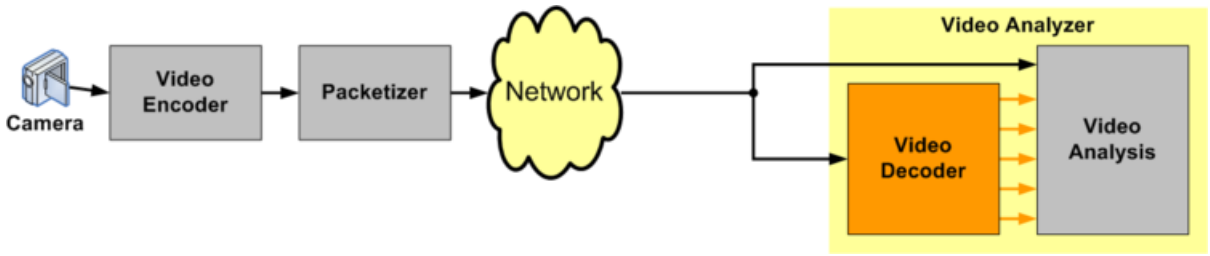
A simple pixel domain processing chain

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

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

Due to the limitations of pixel domain approaches, more and more attempts were made to transfer the video analysis procedures from pixel domain to compressed domain. Working

within compressed domain means to work directly on compressed data. The following figure gives an example for a compressed domain processing chain.

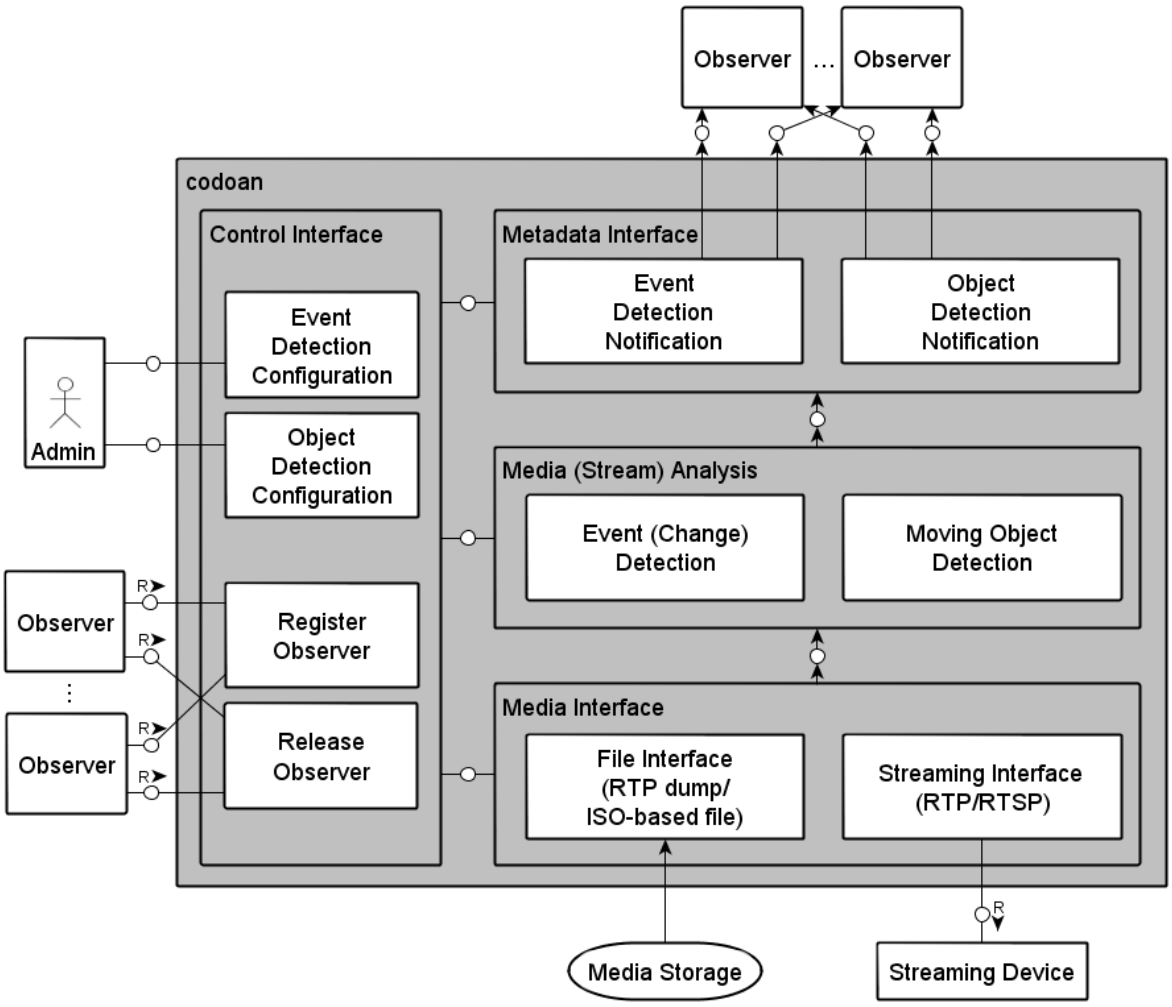


A simple compressed domain processing chain

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

6.4 Architecture

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



CDVA GE – Functional description

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

6.4.1 Media Interface

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

- **Media Storage**: A multimedia file has already been generated and is stored on a server in a file system or in a database. For analysis, the media file can be accessed independently of the original timing. This means that analysis can happen slower or

faster than real-time and random access on the timed media data can be performed. The **File Interface** is able to process the following file types:

- RTP dump file format used by the RTP Tools, as described in [rtpdump]
- An ISO-based file format (e.g., MP4), according to ISO/IEC 14496-12 [ISO08], is envisioned

- **Streaming Device**: A video stream is generated by a device (e.g., a video camera) and streamed over a network using dedicated transport protocols (e.g., RTP, DASH). For analysis, the media stream can be accessed only in its original timing, since the stream is generated in real time. The **Streaming Interface** is able to process the following stream types:
  - Real-time Transport Protocol (RTP) packet streams as standardized in RFC 3550 [RFC3550]. Payload formats to describe the contained compression format can be further specified (e.g., RFC 6184 [RFC6184] for the H.264/AVC payload).
  - Media sessions established using RTSP (RFC 2326 [RFC2326])
  - HTTP-based video streams (e.g., REST-like APIs). URLs/URIs could be used to identify the relevant media resources. (envisioned)

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

6.4.2 Media (Stream) Analysis

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

Currently two sub-components are integrated:

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

In principle, the analytics operations can be done in real time. In practical implementations, this depends on computational resources, the complexity of the algorithm and the quality of

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

6.4.3 Metadata Interface

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

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

6.4.4 Control Interface

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

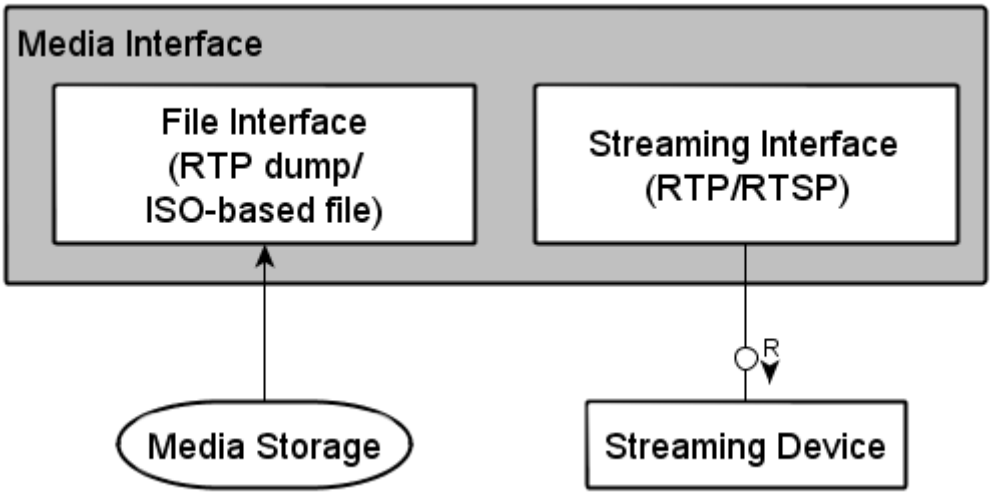
- **Register Observer** : Registered Observers will be notified in case of detections.
- **Release Observer** : Released Observers will not be notified anymore.

6.5 Main Interactions

6.5.1 Interfaces

6.5.1.1 *Media Interface*

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



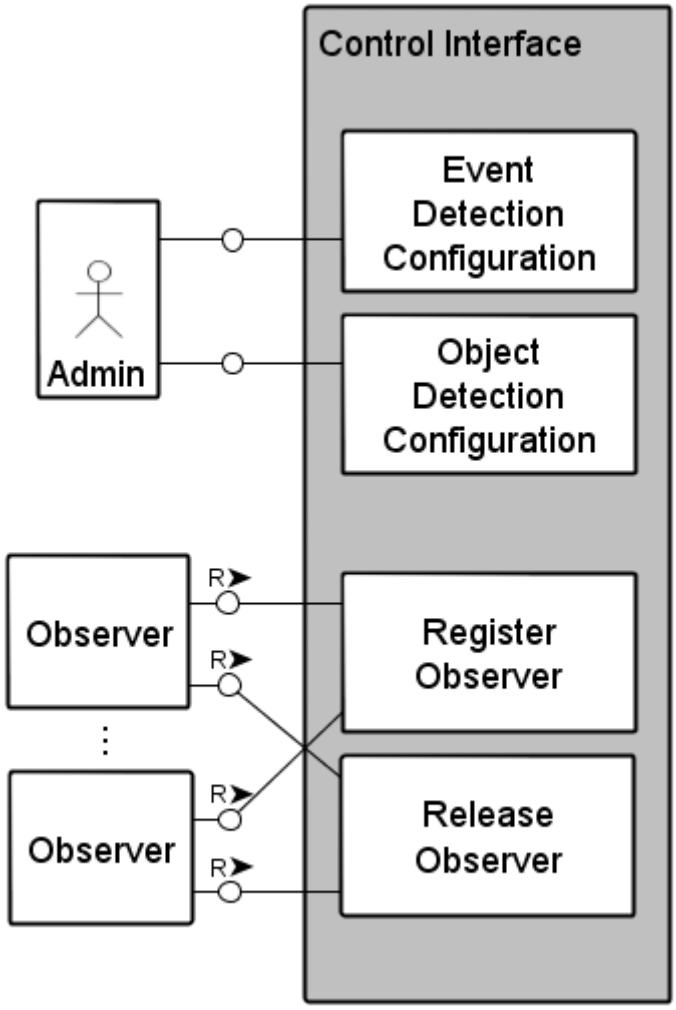
CDVA GE – Media Interface

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

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

6.5.1.2 *Control Interface*

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

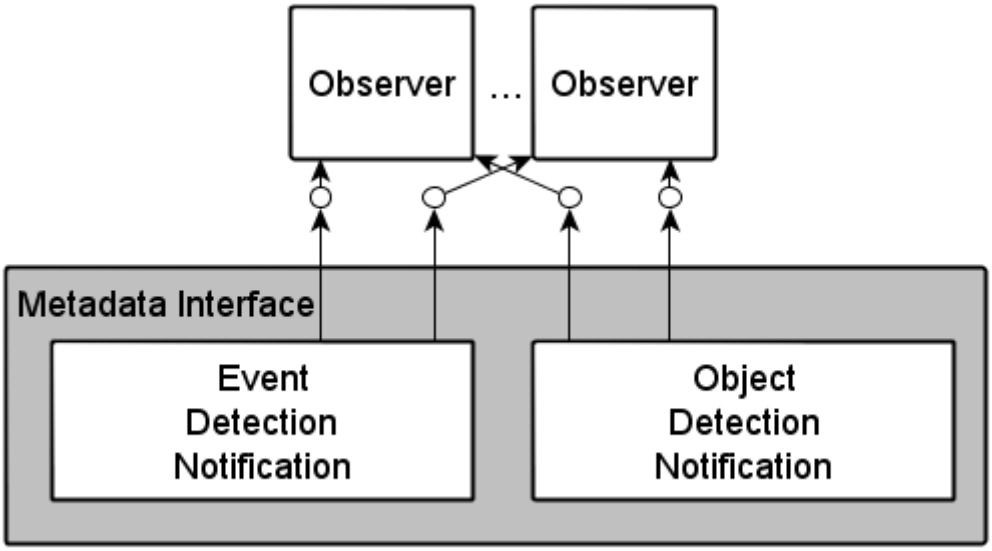


CDVA GE – Control Interface

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

6.5.1.3 *Metadata Interface*

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



CDVA GE – Metadata Interface

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

6.5.2 Operations

6.5.2.1 *Configuration*

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

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

6.5.2.2 *Notification*

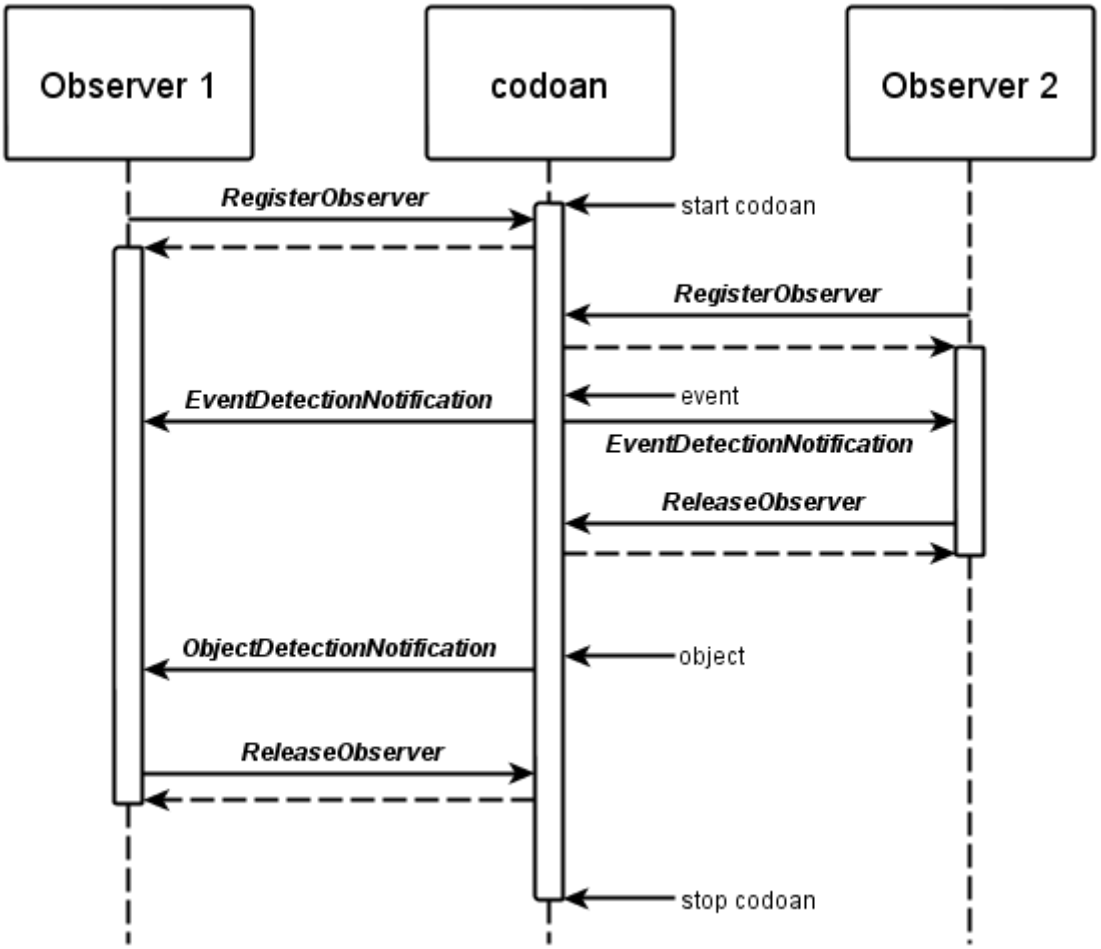
Observers can send requests for registration and release by calling the following methods, respectively:

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

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

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

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



CDVA GE – Example scenario

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

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

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

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

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

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

6.6 Basic Design Principles

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

- GE implementations support performing parallel analyses using different sub-components.

6.7 References

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

6.8 Detailed Specifications

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

6.8.1 Open API Specifications

- Compressed Domain Video Analysis Open RESTful API Specification (PRELIMINARY)

6.9 Re-utilised Technologies/Specifications

The following technologies/specifications are incorporated in this GE:

- ISO/IEC 14496-12:2008, Information technology – Coding of audio-visual objects – Part 12: ISO base media file format
- Real-Time Transport Protocol (RTP) / RTP Control Protocol (RTCP) as defined in RFC 3550
- Real-Time Streaming Protocol (RTSP) as defined in RFC 2326
- RTP Payload Format for H.264 Video as defined in RFC 6184
- ONVIF Specifications
- rtpdump format as defined in RTP Tools

6.10 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.

- **Data** refers to information that is produced, generated, collected or observed that may be relevant for processing, carrying out further analysis and knowledge extraction. Data in FI-WARE has associated a **data type** and a**value**. FI-WARE will support a set of built-in **basic data types** similar to those existing in most programming languages. Values linked to basic data types supported in FI-WARE are referred as **basic data values**. As an example, basic data values like '2', '7' or '365' belong to the integer basic data type.
- A **data element** refers to data whose value is defined as consisting of a sequence of one or more <name, type, value> triplets referred as **data element attributes**, where the type and value of each attribute is either mapped to a basic data type and a basic data value or mapped to the data type and value of another data element.
- **Context** in FI-WARE is represented through **context elements**. A context element extends the concept of **data element** by associating an EntityId and EntityType to it, uniquely identifying the entity (which in turn may map to a group of entities) in the FI-WARE system to which the context element information refers. In addition, there may be some attributes as well as meta-data associated to attributes that we may define as mandatory for context elements as compared to data elements. Context elements are typically created containing the value of attributes characterizing a given entity at a given moment. As an example, a context element may contain values of some of the attributes "last measured temperature", "square meters" and "wall color" associated to a room in a building. Note that there might be many different context elements referring to the same entity in a system, each containing the value of a different set of attributes. This allows that different applications handle different context elements for the same entity, each containing only those attributes of that entity relevant to the corresponding application. It will also allow representing updates on set of attributes linked to a given entity: each of these updates can actually take

the form of a context element and contain only the value of those attributes that have changed.

- An **event** is an occurrence within a particular system or domain; it is something that has happened, or is contemplated as having happened in that domain. Events typically lead to creation of some data or context element describing or representing the events, thus allowing them to processed. As an example, a sensor device may be measuring the temperature and pressure of a given boiler, sending a context element every five minutes associated to that entity (the boiler) that includes the value of these to attributes (temperature and pressure). The creation and sending of the context element is an event, i.e., what has occurred. Since the data/context elements that are generated linked to an event are the way events get visible in a computing system, it is common to refer to these data/context elements simply as "events".
- A **data event** refers to an event leading to creation of a data element.
- A **context event** refers to an event leading to creation of a context element.
- An **event object** is used to mean a programming entity that represents an event in a computing system [EPIA] like event-aware GEs. Event objects allow to perform operations on event, also known as **event processing**. Event objects are defined as a data element (or a context element) representing an event to which a number of standard event object properties (similar to a header) are associated internally. These standard event object properties support certain event processing functions.

7 Compressed Domain Video Analysis Open RESTful API Specification (PRELIMINARY)

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

7.1 Introduction to the Compressed Domain Video Analysis GE API

Please check the FI-WARE Open Specifications Legal Notice to understand the rights to use FI-WARE Open Specifications.

7.1.1 Compressed Domain Video Analysis GE API Core

The Compressed Domain Video Analysis GE API is a RESTful, resource-oriented API accessed via HTTP that uses XML-based representations for information interchange.

7.1.2 Intended Audience

This specification is intended for both software/application developers and application providers. This document provides a full specification of how to interoperate with platforms that implement Compressed Domain Video Analysis GE API.

In order to use this specifications, the reader should firstly have a general understanding of the appropriate Generic Enabler supporting the API (Compressed Domain Video Analysis GE Product Vision).

7.1.3 API Change History

This version of the Compressed Domain Video Analysis GE API Guide replaces and obsoletes all previous versions. The most recent changes are described in the table below:

Revision Date	Changes Summary
May 2, 2012	Initial version
May 21, 2012	• Adapted to new template • Added operations for multiple CDVA instances
August 21, 2012	Updated API operations
August 23, 2012	Changed GE name to Compressed Domain Video Analysis
November 8, 2012	Updated API operations according to new software version

7.1.4 How to Read This Document

All FI-WARE RESTful API specifications will follow the same list of conventions and will support certain common aspects. Please check Common aspects in FI-WARE Open Restful API Specifications.

For a description of some terms used along this document, see the Compressed Domain Video Analysis GE Architecture Description.

The ONVIF specifications and the OASIS Web Services Notification standard define XML structures and elements that are used within the notification module of the Compressed Domain Video Analysis GE (see Notification). The analyzed media is received by using RTP, as defined by IETF RFC-3550, and RTSP, as defined by IETF RFC-2326.

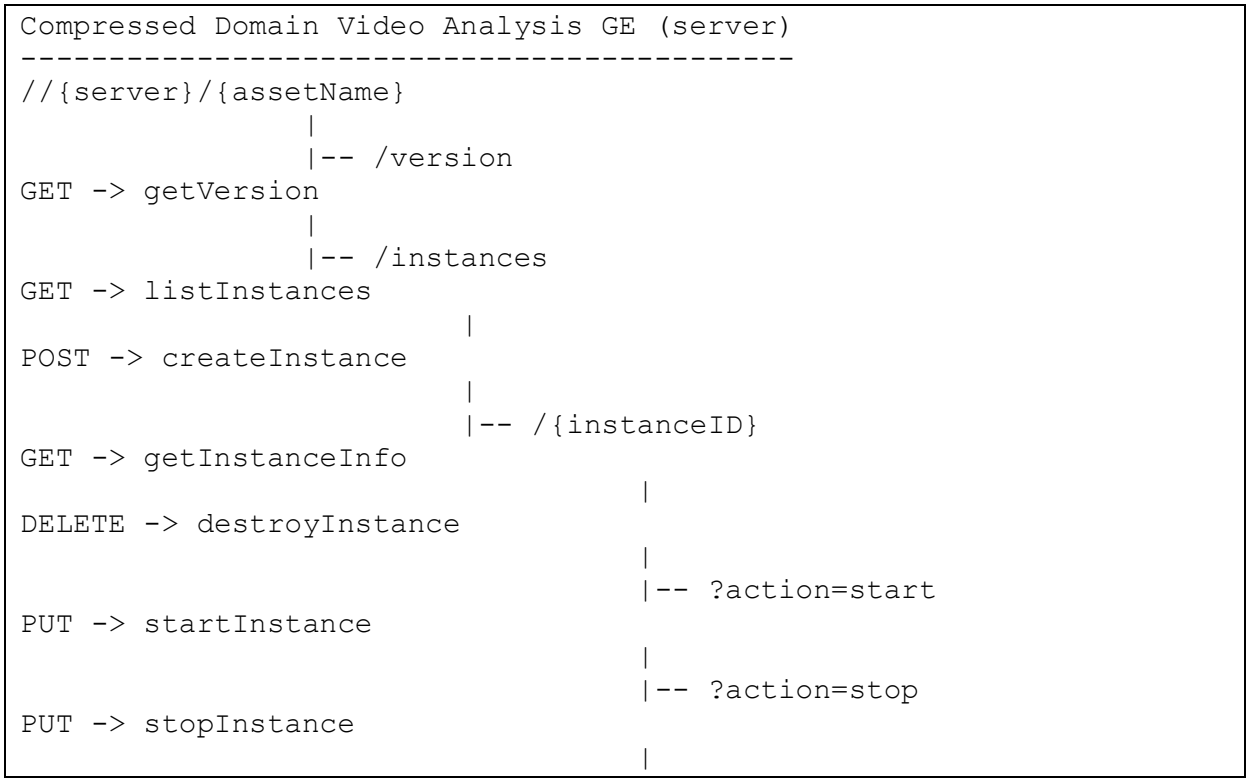
7.1.5 Additional Resources

You can download the most current version of this document from the FI-WARE API specification website: Compressed Domain Video Analysis Open RESTful API Specification. For more details about the Compressed Domain Video Analysis GE that this API is based upon, please refer to Compressed Domain Video Analysis GE Product Vision. Related documents, including an Architectural Description, are available at the same site.

7.2 General Compressed Domain Video Analysis GE API Information

7.2.1 Resources Summary

The resource summary is shown in the following graphical diagram.



```
                                        |-- /config
GET -> getInstanceConfig
                                        |
PUT -> configureInstance
                                        |
                                        |-- /sinks
GET -> listSinks
                                                |
POST -> addSink
                                                |
                                                |-- /{sinkID}
GET -> getSinkInfo

DELETE -> removeSink

Sink (client)
-------------
//{sinkNotificationURI}
```

7.2.2 Representation Format

The Compressed Domain Video Analysis GE API supports XML-based representation formats for both requests and responses. This is specified by setting the Content-Type header to *application/xml*, if the request/response has a body.

7.2.3 Resource Identification

The resource identification for HTTP transport is made using the mechanisms described by HTTP protocol specification as defined by IETF RFC-2616.

7.2.4 Links and References

Request forwarding is not supported in Version 1 of the Compressed Domain Video Analysis GE.

7.2.5 Limits

Limits are not yet specified for Version 1 of the Compressed Domain Video Analysis GE.

7.2.6 Versions

The current version of the used implementation of the Compressed Domain Video Analysis GE can be requested by the following HTTP request:

```
GET //{server}/{assetName}/version HTTP/1.1
```

7.2.7 Extensions

Querying extensions is not supported in Version 1 of the Compressed Domain Video Analysis GE.

7.2.8 Faults

Fault elements and their associated error codes are described in the following table.

Fault Element	Error Code	Reason Phrase	Description	Expected in All Requests?
GET, **POST**, **PUT**, **DELETE**	400	Bad Request	The client sent an invalid request the server is not able to process. The message body may contain a detailed description of this error.	[YES]
GET, **POST**, **PUT**, **DELETE**	404	Not Found	The requested resource does not exist. The message body may contain a detailed description of this error.	[YES]
GET, **POST**, **PUT**, **DELETE**	405	Method Not Allowed	The used HTTP method is not allowed for the requested resource. The message body may contain a detailed description of this error.	[YES]
GET, **POST**, **PUT**, **DELETE**	500	Internal Server Error	An unforeseen error occurred at the server. The message body may contain a detailed description of this error.	[YES]

7.3 API Operations

7.3.1 /version

Verb	URI	Description
GET	//{server}/{assetName}/version	Returns the current version of the Compressed Domain Video Analysis GE implementation

7.3.1.1 *getVersion*

Sample request:

```
GET //192.0.2.1/codoan/version HTTP/1.1
Accept: application/xml
```

Sample response:

```
HTTP/1.1 200 OK
Content-Length: 188
Content-Type: application/xml
Server: codoan RESTful web server (Mongoose web server)

<?xml version="1.0" encoding="UTF-8"?>
<Codoan>
  <Version>codoan v1.0</Version>
  <Copyright>(c) 2010-2012 Imaging and Computer Vision, Siemens
Corporate Technology</Copyright>
</Codoan>
```

On success, the response code to this request is as stated in the example above. In case an error occurred, one of the error codes described in section Faults is returned.

7.3.2 /instances

Verb	URI	Description
GET	//{server}/{assetName}/instances	Lists all active instances of the Compressed Domain Video Analysis GE
POST	//{server}/{assetName}/instances	Creates a new instance of the Compressed Domain Video Analysis GE

7.3.2.1 *listInstances*

Sample request:

```
GET //192.0.2.1/codoan/instances HTTP/1.1
Accept: application/xml
```

Sample response:

```
HTTP/1.1 200 OK
Content-Length: 71
Content-Type: application/xml
Server: codoan RESTful web server (Mongoose web server)

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

7.3.2.2 *createInstance*

Sample request:

```
POST //192.0.2.1/codoan/instances HTTP/1.1
Accept: application/xml
Content-Length: 185
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<Codoan>
  <Instances>
    <Instance        detectEvents="true"        detectObjects="true"
streamURI="rtsp://192.0.2.2/stream1"/>
  </Instances>
</Codoan>
```

Sample response:

```
HTTP/1.1 201 Created
Content-Length: 228
Content-Type: application/xml
Server: codoan RESTful web server (Mongoose web server)

<?xml version="1.0" encoding="UTF-8"?>
<Codoan>
  <Instances>
    <Instance          activeSinks="0"          detectEvents="true"
detectObjects="true"           id="101"           isRunning="false"
streamURI="rtsp://192.0.2.2/stream1"/>
  </Instances>
</Codoan>
```

On success, the response codes to these requests are as stated in the examples above. In case an error occurred, one of the error codes described in section Faults is returned.

7.3.3 /{instanceID}

Verb	URI	Description
GET	//{server}/{assetName}/instances/{instanceID}	Returns information about an existing instance of the Compressed Domain Video Analysis GE
DELETE	//{server}/{assetName}/instances/{instanceID}	Destroys an existing instance of the Compressed Domain Video Analysis GE
PUT	//{server}/{assetName}/instances/{instanceID}?action=start	Starts the analysis of an existing instance of the Compressed Domain Video Analysis GE
PUT	//{server}/{assetName}/instances/{instanceID}?action=stop	Stops the analysis of an existing instance of the Compressed Domain Video Analysis GE

7.3.3.1 *getInstanceInfo*

Sample request:

```
GET //192.0.2.1/codoan/instances/101 HTTP/1.1
Accept: application/xml
```

Sample response:

```
HTTP/1.1 200 OK
Content-Length: 228
Content-Type: application/xml
Server: codoan RESTful web server (Mongoose web server)

<?xml version="1.0" encoding="UTF-8"?>
<Codoan>
  <Instances>
    <Instance activeSinks="0" detectEvents="true"
detectObjects="true" id="101" isRunning="false"
streamURI="rtsp://192.0.2.2/stream1"/>
  </Instances>
</Codoan>
```

7.3.3.2 *destroyInstance*

Sample request:

```
DELETE //192.0.2.1/codoan/instances/101 HTTP/1.1
Accept: application/xml
```

Sample response:

```
HTTP/1.1 200 OK
Content-Length: 228
Content-Type: application/xml
Server: codoan RESTful web server (Mongoose web server)

<?xml version="1.0" encoding="UTF-8"?>
<Codoan>
  <Instances>
    <Instance activeSinks="0" detectEvents="true"
detectObjects="true" id="101" isRunning="false"
streamURI="rtsp://192.0.2.2/stream1"/>
  </Instances>
</Codoan>
```

7.3.3.3 *startInstance*

Sample request:

```
PUT //192.0.2.1/codoan/instances/101?action=start HTTP/1.1
Accept: application/xml
```

Sample response:

```
HTTP/1.1 200 OK
Content-Length: 227
Content-Type: application/xml
Server: codoan RESTful web server (Mongoose web server)

<?xml version="1.0" encoding="UTF-8"?>
<Codoan>
  <Instances>
    <Instance activeSinks="1" detectEvents="true"
detectObjects="true" id="101" isRunning="true"
streamURI="rtsp://192.0.2.2/stream1"/>
  </Instances>
</Codoan>
```

7.3.3.4 *stopInstance*

Sample request:

```
PUT //192.0.2.1/codoan/instances/101?action=stop HTTP/1.1
Accept: application/xml
```

Sample response:

```
HTTP/1.1 200 OK
Content-Length: 228
Content-Type: application/xml
Server: codoan RESTful web server (Mongoose web server)

<?xml version="1.0" encoding="UTF-8"?>
<Codoan>
  <Instances>
    <Instance activeSinks="0" detectEvents="true"
detectObjects="true" id="101" isRunning="false"
streamURI="rtsp://192.0.2.2/stream1"/>
  </Instances>
</Codoan>
```

On success, the response codes to these requests are as stated in the examples above. In case an error occurred, one of the error codes described in section Faults is returned.

7.3.4 /config

Verb	URI	Description
GET	//{server}/{assetName}/instances/{instanceID}/config	Returns the configuration of an existing instance of the Compressed Domain Video Analysis GE
PUT	//{server}/{assetName}/instances/{instanceID}/config	Configures an existing instance of the Compressed Domain Video Analysis GE

7.3.4.1 *getInstanceConfig*

Sample request:

```
GET //192.0.2.1/codoan/instances/101/config HTTP/1.1
Accept: application/xml
```

Sample response:

```
HTTP/1.1 200 OK
Content-Length: 740
Content-Type: application/xml
Server: codoan RESTful web server (Mongoose web server)

<?xml version="1.0" encoding="UTF-8"?>
<Codoan>
  <Instances>
    <Instance activeSinks="0" detectEvents="true"
detectObjects="true" id="101" isRunning="false"
streamURI="rtsp://192.0.2.2/stream1">
```

```
      <Configuration>
        <Event>
          <NumberOfTrainingFrames>40</NumberOfTrainingFrames>
          <SlidingWindowSize>10</SlidingWindowSize>
          <ThresholdANORPFactor>1.2</ThresholdANORPFactor>
          <ThresholdARPSFactor>1.75</ThresholdARPSFactor>
          <ThresholdIFrame>5</ThresholdIFrame>
        </Event>
        <Object>
          <BoxFilterSize>3</BoxFilterSize>
          <ThresholdH264MOC>6</ThresholdH264MOC>
        </Object>
      </Configuration>
    </Instance>
  </Instances>
</Codoan>
```

7.3.4.2 *configureInstance*

Sample request:

```
PUT //192.0.2.1/codoan/instances/101/config HTTP/1.1
Accept: application/xml
Content-Length: 380
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<Codoan>
  <Instances>
    <Instance detectEvents="false" detectObjects="true"
streamURI="rtsp://192.0.2.2/stream1">
      <Configuration>
        <Object>
          <BoxFilterSize>3</BoxFilterSize>
          <ThresholdH264MOC>6</ThresholdH264MOC>
        </Object>
      </Configuration>
    </Instance>
  </Instances>
</Codoan>
```

Sample response:

```
HTTP/1.1 200 OK
Content-Length: 741
Content-Type: application/xml
Server: codoan RESTful web server (Mongoose web server)

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

```
    <Instance activeSinks="0" detectEvents="false"
detectObjects="true" id="101" isRunning="false"
streamURI="rtsp://192.0.2.2/stream1">
      <Configuration>
        <Event>
          <NumberOfTrainingFrames>40</NumberOfTrainingFrames>
          <SlidingWindowSize>10</SlidingWindowSize>
          <ThresholdANORPFactor>1.2</ThresholdANORPFactor>
          <ThresholdARPSFactor>1.75</ThresholdARPSFactor>
          <ThresholdIFrame>5</ThresholdIFrame>
        </Event>
        <Object>
          <BoxFilterSize>3</BoxFilterSize>
          <ThresholdH264MOC>6</ThresholdH264MOC>
        </Object>
      </Configuration>
    </Instance>
  </Instances>
</Codoan>
```

On success, the response codes to these requests are as stated in the examples above. In case an error occurred, one of the error codes described in section Faults is returned.

7.3.5 /sinks

Verb	URI	Description
GET	//{server}/{assetName}/instances/{instanceID}/sinks	Lists all active sinks of an existing instance of the Compressed Domain Video Analysis GE
POST	//{server}/{assetName}/instances/{instanceID}/sinks	Adds a new sink to an existing instance of the Compressed Domain Video Analysis GE

7.3.5.1 *listSinks*

Sample request:

```
GET //192.0.2.1/codoan/instances/101/sinks HTTP/1.1
Accept: application/xml
```

Sample response:

```
HTTP/1.1 200 OK
Content-Length: 260
Content-Type: application/xml
Server: codoan RESTful web server (Mongoose web server)

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

```
  <Instances>
    <Instance activeSinks="0" detectEvents="true"
detectObjects="true" id="101" isRunning="false"
streamURI="rtsp://192.0.2.2/stream1">
      <Sinks/>
    </Instance>
  </Instances>
</Codoan>
```

7.3.5.2 *addSink*

Sample request:

```
POST //192.0.2.1/codoan/instances/101/sinks HTTP/1.1
Accept: application/xml
Content-Length: 293
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<Codoan>
  <Instances>
    <Instance detectEvents="true" detectObjects="true"
streamURI="rtsp://192.0.2.2/stream1">
      <Sinks
sinkNotificationURI="http://192.0.2.3/notification/stream1">
    </Instance>
  </Instances>
</Codoan>
```

Sample response:

```
HTTP/1.1 201 Created
Content-Length: 328
Content-Type: application/xml
Server: codoan RESTful web server (Mongoose web server)

<?xml version="1.0" encoding="UTF-8"?>
<Codoan>
  <Instances>
    <Instance activeSinks="1" detectEvents="true"
detectObjects="true" id="101" isRunning="false"
streamURI="rtsp://192.0.2.2/stream1">
      <Sinks id="201"
sinkNotificationURI="http://192.0.2.3/notification/stream1">
    </Instance>
  </Instances>
</Codoan>
```

On success, the response codes to these requests are as stated in the examples above. In case an error occurred, one of the error codes described in section Faults is returned.

7.3.6 /{sinkID}

Verb	URI	Description
GET	//{server}/{assetName}/instances/{instanceID}/sinks/{sinkID}	Returns information about of an existing sink of the Compressed Domain Video Analysis GE
DELETE	//{server}/{assetName}/instances/{instanceID}/sinks/{sinkID}	Removes an existing

		sink of the Compressed Domain Video Analysis GE

7.3.6.1 *getSinkInfo*

Sample request:

```
GET //192.0.2.1/codoan/instances/101/sinks/201 HTTP/1.1
Accept: application/xml
```

Sample response:

```
HTTP/1.1 200 OK
Content-Length: 328
Content-Type: application/xml
Server: codoan RESTful web server (Mongoose web server)

<?xml version="1.0" encoding="UTF-8"?>
<Codoan>
  <Instances>
    <Instance activeSinks="1" detectEvents="true"
detectObjects="true" id="101" isRunning="false"
streamURI="rtsp://192.0.2.2/stream1">
      <Sinks id="201"
sinkNotificationURI="http://192.0.2.3/notification/stream1">
    </Instance>
  </Instances>
</Codoan>
```

7.3.6.2 *removeSink*

Sample request:

```
DELETE //192.0.2.1/codoan/instances/101/sinks/201 HTTP/1.1
Accept: application/xml
```

Sample response:

```
HTTP/1.1 200 OK
Content-Length: 328
Content-Type: application/xml
Server: codoan RESTful web server (Mongoose web server)

<?xml version="1.0" encoding="UTF-8"?>
<Codoan>
  <Instances>
    <Instance activeSinks="1" detectEvents="true"
detectObjects="true" id="101" isRunning="false"
streamURI="rtsp://192.0.2.2/stream1">
      <Sinks id="201"
sinkNotificationURI="http://192.0.2.3/notification/stream1">
    </Instance>
```

```
  </Instances>
</Codoan>
```

On success, the response codes to these requests are as stated in the examples above. In case an error occurred, one of the error codes described in section Faults is returned.

7.3.7 //{sinkNotificationURI}

Verb	URI	Description
POST	//{sinkNotificationURI}	Notifies the sink in case of events or detected objects

7.3.7.1 *notifySink*

Sample request:

```
POST //192.0.2.3/notification/stream1 HTTP/1.1
Content-Length: 1763
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<MetadataStream xmlns="http://www.onvif.org/ver10/schema"
                xmlns:wsn="http://docs.oasis-open.org/wsn/b-2"
                xmlns:codoan="http://www.fi-
ware.eu/data/mma/codoan/schema"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
                xsi:schemaLocation="http://www.fi-
ware.eu/data/mma/codoan/schema codoan.xsd">

  <Event>
    <wsn:NotificationMessage>
      <wsn:Message>
        <codoan:EventDescription>
          <codoan:EventType>GlobalChange</codoan:EventType>
          <codoan:FrameNumber>100</codoan:FrameNumber>
        </codoan:EventDescription>
      </wsn:Message>
    </wsn:NotificationMessage>
  </Event>

  <VideoAnalytics>
    <Frame UtcTime="2012-05-10T18:12:05.432Z"
codoan:FrameNumber="100">
      <Object ObjectId="0">
        <Appearance>
          <Shape>
            <BoundingBox bottom="15.0" top="5.0" right="25.0"
left="15.0"/>
            <CenterOfGravity x="20.0" y="10.0"/>
          </Shape>
```

```
        </Appearance>
      </Object>
      <Object ObjectId="1">
        <Appearance>
          <Shape>
            <BoundingBox bottom="25.0" top="15.0" right="35.0"
left="25.0"/>
            <CenterOfGravity x="30.0" y="20.0"/>
          </Shape>
        </Appearance>
      </Object>
    </Frame>
  </VideoAnalytics>

  <Extension>
    <codoan:StreamProperties>
      <codoan:StreamUri>rtsp://camera/stream1</codoan:StreamUri>
      <codoan:GopSize>10</codoan:GopSize>
      <codoan:FrameRate>25.0</codoan:FrameRate>
      <codoan:FrameWidth>352</codoan:FrameWidth>
      <codoan:FrameHeight>288</codoan:FrameHeight>
    </codoan:StreamProperties>
  </Extension>
</MetadataStream>
```

Assumed response:

```
HTTP/1.1 200 OK
```

8 FIWARE OpenSpecification Data Location

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

Name	FIWARE.OpenSpecification.Data.Location
Chapter	Data/Context Management,
Catalogue-Link to Implementation	<Location Platform>
Owner	Thales Alenia Space, Tanguy Bourgault

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.

8.1.1 Copyright

- Copyright © 2012 by Thales

8.1.2 Legal Notice

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

8.2 Overview

8.2.1 Introduction to the Data Location GE

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

- Third-party location clients

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

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

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

- Mobile end-users

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

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

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

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

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

8.2.2 Target usage

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

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

8.3 Basic Concepts

8.3.1 Third-party location services

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

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

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

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

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

8.3.2 Access control and privacy management

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

8.3.3 Mobile end-user services

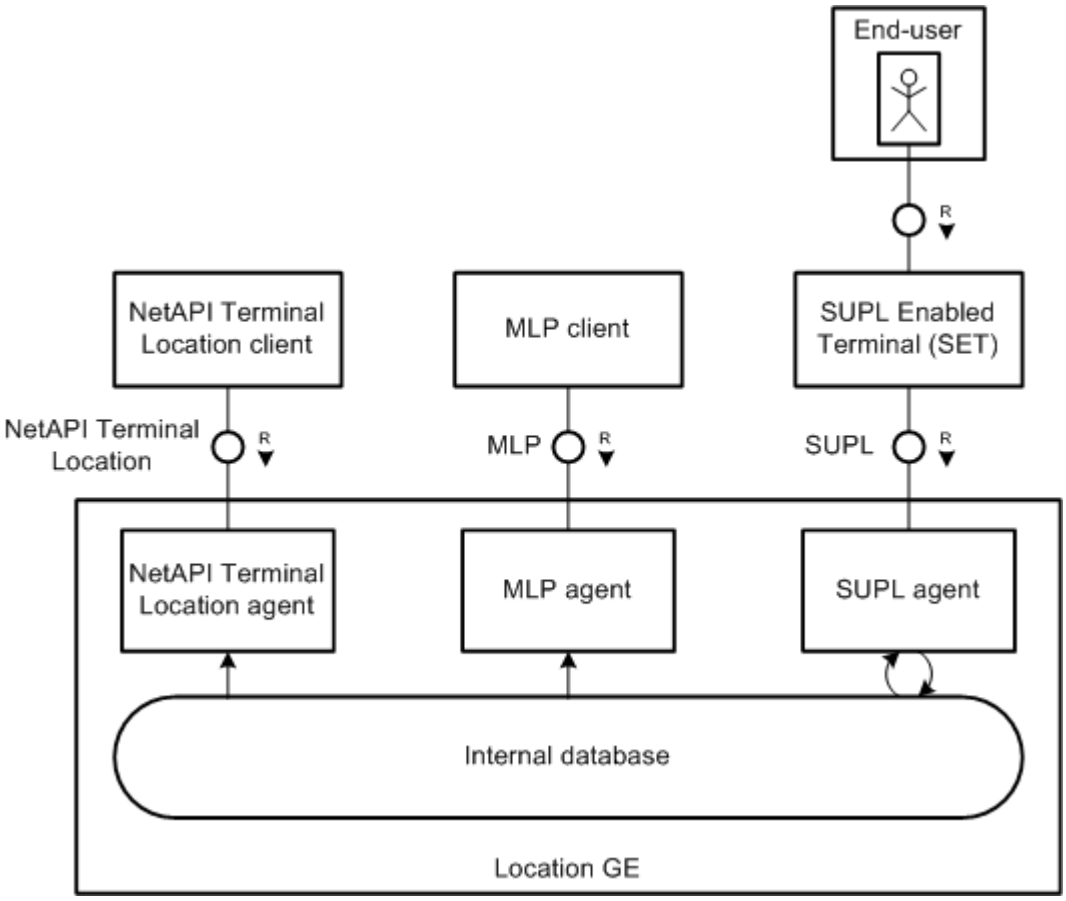
Services provided for mobile end-users are standardized under the name "set-initiated" procedures, since the location request is established by the SUPL Enabled Terminal (SET) on behalf of the end-user launching the application requiring location information.

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

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

8.3.4 Interfaces and data model

The following diagram illustrates the interfaces previously presented:



The following agents are the grounds of the Location GE:

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

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

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

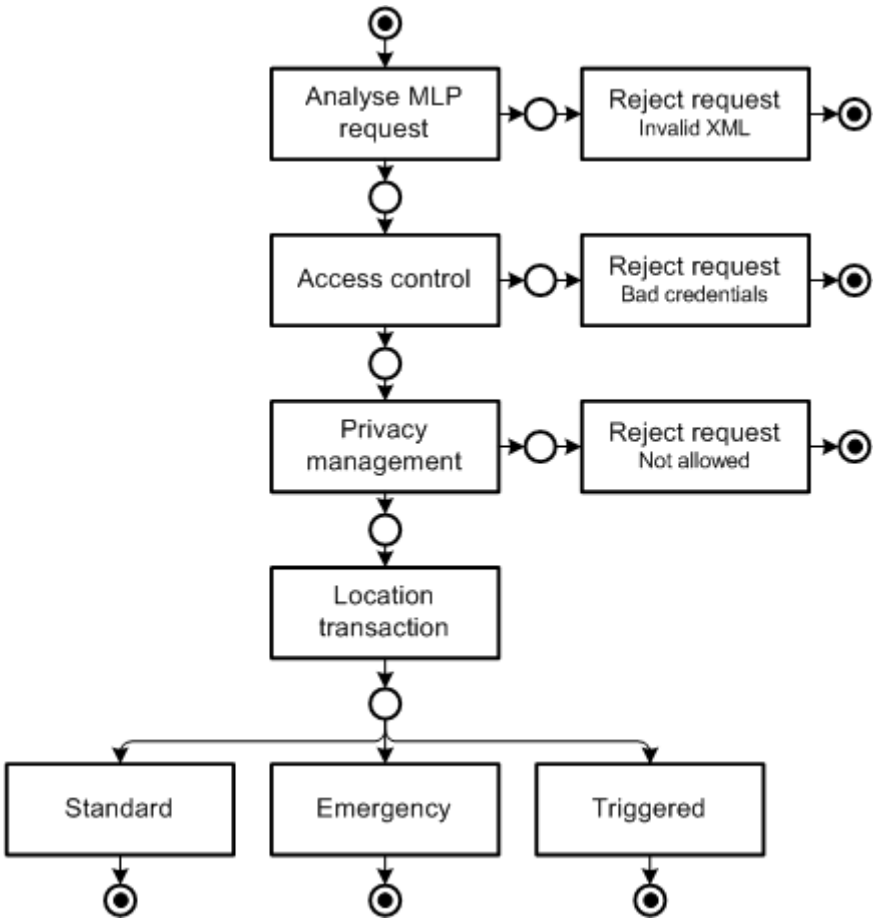
- Third-party information: third-party account credentials and settings.
- Third-party location services information: contains many parameters, including level of authority (lawful/standard), authorized level of accuracy (low/high), type of location authorized (standard/emergency/tracking), flow control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

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

8.4 Main Interactions

8.4.1 MLP services

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



Access control and privacy management

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

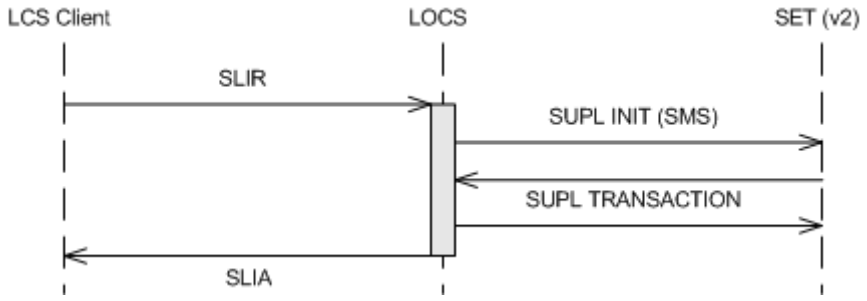
```
<?xml version="1.0" ?>
<svc_init ver="3.2.0">
  <hdr ver="3.2.0">
    <client>
      <id>login</id>
      <pwd>password</pwd>
      <serviceid>servicename</serviceid>
    </client>
    <requestor type="MSISDN">
      <id>33612345680</id>
    </requestor>
  </hdr>
  <!-- Location request -->
</svc_init>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

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

and target end-user MSISDN are utilized to check the end-user privacy policy previously presented.

Standard Location Immediate Service

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



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

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

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

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

```
      </shape>
      <alt>0</alt>
      <alt_unc>707</alt_unc>
    </pd>
  </pos>
  <pos>
    <msid>33612345679</msid>
    <pd>
      <time>20020623134454</time>
      <shape>
        <EllipticalArea>
             <coord>
              <X>50.445668</X>
              <Y>2.803677</Y>
             </coord>
             <angle>0.0</angle>
             <semiMajor>707</semiMajor>
             <semiMinor>707</semiMinor>
             <angularUnit>Radians</angularUnit>
        </EllipticalArea>
      </shape>
      <alt>0</alt>
      <alt_unc>707</alt_unc>
    </pd>
  </pos>
</slia>
```

Emergency Location Immediate Service

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

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

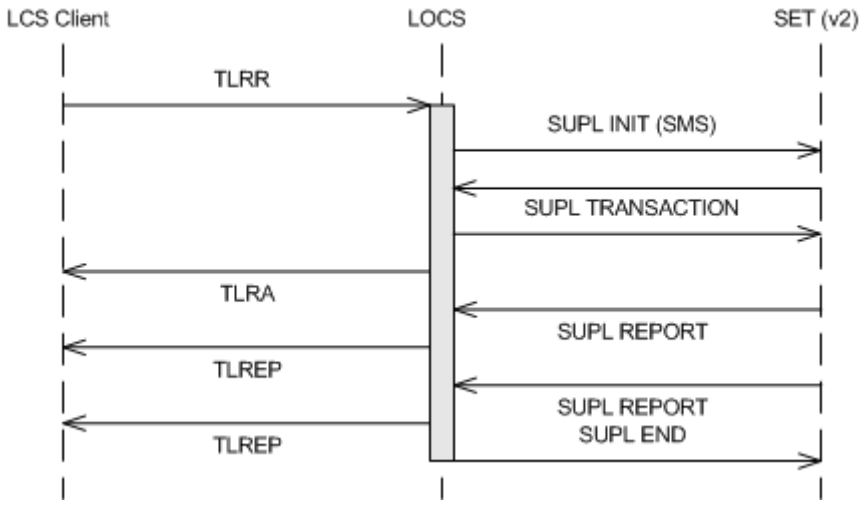
This request triggers an emergency network-initiated SUPL transaction towards the handset. Once the handset location is retrieved, the Location Platform responds with an EME_LIA response instead of a SLIA, containing the position of the target end-user:

```
<eme_lia ver="3.2.0">
  <eme_pos>
    <msid type="MSISDN">33612345678</msid>
    <pd>
      <time>20020623134454</time>
      <shape>
```

```
        <EllipticalArea>
             <coord>
              <X>50.445668</X>
              <Y>2.803677</Y>
             </coord>
             <angle>0.0</angle>
             <semiMajor>707</semiMajor>
             <semiMinor>707</semiMinor>
             <angularUnit>Radians</angularUnit>
        </EllipticalArea>
      </shape>
      <alt>0</alt>
      <alt_unc>707</alt_unc>
    </pd>
  </eme_pos>
</eme_lia>
```

8.4.1.1 *Triggered Location Reporting Service*

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



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

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

```
  </pushaddr>
  <loc_type type="CURRENT"/>
</tlrr>
```

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

```
<tlra ver="3.2.0">
  <req_id>25293</req_id>
</tlra>
```

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

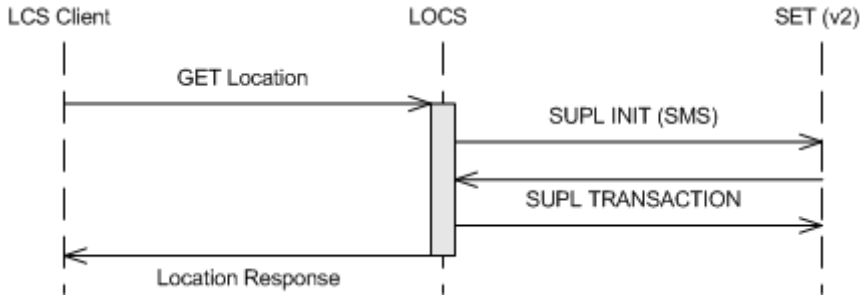
```
<tlrep ver="3.2.0">
  <req_id>25293</req_id>
  <trl_pos trl_trigger="PERIODIC">
    <msid type="MSISDN">33612345679</msid>
    <pd>
      <time>20020623134453</time>
      <shape>
        <EllipticalArea>
             <coord>
              <X>50.445668</X>
              <Y>2.803677</Y>
             </coord>
             <angle>0.0</angle>
             <semiMajor>707</semiMajor>
             <semiMinor>707</semiMinor>
             <angularUnit>Radians</angularUnit>
        </EllipticalArea>
      </shape>
      <alt>0</alt>
      <alt_unc>707</alt_unc>
    </pd>
  </trl_pos>
</tlrep>
```

8.4.2 NetAPI Terminal Location services

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

8.4.2.1 *Location Query*

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



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

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

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

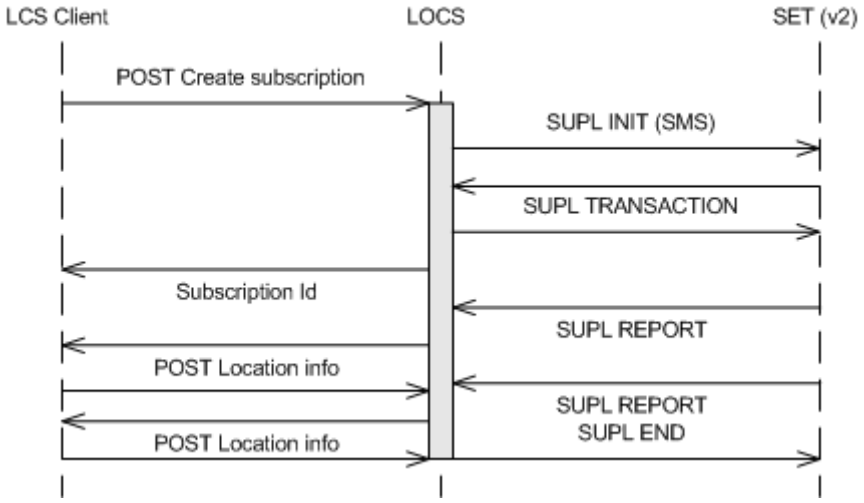
```
 HTTP/1.1 200 OK
 Content-Type: application/xml
 Content-Length: nnnn
 Date: Thu, 02 Jun 2011 02:51:59 GMT

 <?xml version="1.0" encoding="UTF-8"?>
 <tl:terminalLocationList
xmlns:common="urn:oma:xml:rest:netapi:common:1"
xmlns:tl="urn:oma:xml:rest:netapi:terminallocation:1">
  <tl:terminalLocation>
   <tl:address>33611223344</tl:address>
   <tl:locationRetrievalStatus>Retrieved
   </tl:locationRetrievalStatus>
   <tl:currentLocation>
    <tl:latitude>49.999737</tl:latitude>
    <tl:longitude>-60.00014</tl:longitude>
    <tl:altitude>30.0</tl:altitude>
    <tl:accuracy>55</tl:accuracy>
```

```
    <tl:timestamp>2012-04-17T09:21:32.893+02:00</tl:timestamp>
   </tl:currentLocation>
   <tl:errorInformation>
    <common:messageId>QOP_NOT_ATTAINABLE</common:messageId>
    <common:text>The requested QoP cannot be provided.</common:text>
   </tl:errorInformation>
  </tl:terminalLocation>
 </tl:terminalLocationList>
```

8.4.2.2 *Location Subscriptions*

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



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

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

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

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

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

```
 HTTP/1.1 201 Created
 Content-Type: application/xml
 Location:
http://example.com/location/v1/subscriptions/periodic/sub003
 Content-Length: nnnn
 Date: Thu, 02 Jun 2011 02:51:59 GMT

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

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

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

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

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

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

8.4.3 Positioning

8.4.3.1 *A-GNSS location technology*

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

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

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

8.4.3.2 *C-ID location technology*

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

8.4.3.3 *Location Technology selection*

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

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

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

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

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

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

8.5 Basic Design Principles

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

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

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

8.6 References

MLP	Mobile Location Protocol (MLP), Open Mobile Alliance, specification OMA-TS-MLP-V3_2-20110719-A
SUPL	Secure User Plane Location Protocol (SUPL), Open Mobile Alliance, specification

	OMA-TS-ULP-V2_0-20111222-D
RRLP	Radio Resource LCS Protocol (RRLP), 3GPP, specification 3GPP TS 44.031 V9.2.0 (2010-03)

8.7 Detailed Specifications

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

8.7.1 Open API Specifications

- Location Server Open RESTful API Specification (PRELIMINARY)

8.8 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.

- **Data** refers to information that is produced, generated, collected or observed that may be relevant for processing, carrying out further analysis and knowledge extraction. Data in FI-WARE has associated a **data type** and a**value**. FI-WARE will support a set of built-in **basic data types** similar to those existing in most programming languages. Values linked to basic data types supported in FI-WARE are referred as **basic data values**. As an example, basic data values like ‘2’, ‘7’ or ‘365’ belong to the integer basic data type.
- A **data element** refers to data whose value is defined as consisting of a sequence of one or more <name, type, value> triplets referred as **data element attributes**, where the type and value of each attribute is either mapped to a basic data type and a basic data value or mapped to the data type and value of another data element.
- **Context** in FI-WARE is represented through **context elements**. A context element extends the concept of **data element** by associating an EntityId and EntityType to it, uniquely identifying the entity (which in turn may map to a group of entities) in the FI-WARE system to which the context element information refers. In addition, there may be some attributes as well as meta-data associated to attributes that we may define as mandatory for context elements as compared to data elements. Context elements

are typically created containing the value of attributes characterizing a given entity at a given moment. As an example, a context element may contain values of some of the attributes “last measured temperature”, “square meters” and “wall color” associated to a room in a building. Note that there might be many different context elements referring to the same entity in a system, each containing the value of a different set of attributes. This allows that different applications handle different context elements for the same entity, each containing only those attributes of that entity relevant to the corresponding application. It will also allow representing updates on set of attributes linked to a given entity: each of these updates can actually take the form of a context element and contain only the value of those attributes that have changed.

- An **event** is an occurrence within a particular system or domain; it is something that has happened, or is contemplated as having happened in that domain. Events typically lead to creation of some data or context element describing or representing the events, thus allowing them to processed. As an example, a sensor device may be measuring the temperature and pressure of a given boiler, sending a context element every five minutes associated to that entity (the boiler) that includes the value of these to attributes (temperature and pressure). The creation and sending of the context element is an event, i.e., what has occurred. Since the data/context elements that are generated linked to an event are the way events get visible in a computing system, it is common to refer to these data/context elements simply as "events".
- A **data event** refers to an event leading to creation of a data element.
- A **context event** refers to an event leading to creation of a context element.
- An **event object** is used to mean a programming entity that represents an event in a computing system [EPIA] like event-aware GEs. Event objects allow to perform operations on event, also known as **event processing**. Event objects are defined as a data element (or a context element) representing an event to which a number of standard event object properties (similar to a header) are associated internally. These standard event object properties support certain event processing functions.

9 Location Server Open RESTful API Specification (PRELIMINARY)

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

9.1 Dedicated API Introduction

Please check the FI-WARE Open Specifications Legal Notice to understand the rights to use FI-WARE Open Specifications.

9.2 Introduction to the Restful Network API for Terminal Location

9.2.1 Network API for Terminal Location

The Network API for Terminal Location is a RESTful, resource-oriented API accessed via HTTP that uses XML-based representations for information interchange.

In the scope of FIWARE, a subset of the normalized OMA-TS-REST_NetAPI _TerminalLocation **REST_NetAPI_TerminalLocation** specification is applied. Exact implemented subset is described in following chapters (please refer to below figure).

To summarize, the following operations are supported :

- Obtain the current terminal location
- Manage client-specific subscriptions to periodic notifications
- Manage client-specific subscriptions to area (circle) notifications

9.2.2 Intended Audience

This specification is intended for both software developers and Cloud Providers. For the former, this document provides a full specification of how to interoperate with Location GE platform that implements Terminal Location API. For the latter, this specification indicates the interface to be provided in order to clients to interoperate with Location GE to provide the described functionalities. To use this information, the reader should firstly have a general understanding of the Location Generic Enabler and be familiar with :

- ReSTful web services
- HTTP/1.1
- JSON and/or XML data serialization formats."

9.2.3 API Change History

This version of the Network API for Terminal Location Guide replaces and obsoletes all previous versions.

The following APIs defines the baseline reference API :

[REST_NetAPI_Common] Common definitions for RESTful Network APIs, Open Mobile Alliance™, OMA-TSREST_NetAPI_Common-V1_0, URL: **REST_NetAPI_Common**

[REST_NetAPI_TerminalLocation] RESTful Network API for Terminal Location, Open Mobile Alliance™, OMA-TSREST_NetAPI_TerminalLocation-V1_0, URL: **REST_NetAPI_TerminalLocation**

History of specific changes that overrides this baseline is described in the following table.

Date	Comment
Apr 18, 2012	• Initial Version
Sept 26, 2012	• Complete Support for url-form-encoded and json API format • Add Periodic Subscription service API

9.2.4 How to Read This Document

In the whole document it is taken the assumption that reader is familiarized with REST architecture style. Along the document, some special notations are applied to differentiate some special words or concepts. The following list summarizes these special notations.

- A bold, mono-spaced font is used to represent code or logical entities, e.g., HTTP method (GET, PUT, POST, DELETE).
- An italic font is used to represent document titles or some other kind of special text, e.g., URI.
- The variables are represented between brackets, e.g. {id} and in italic font. When the reader find it, can change it by any value.

9.2.5 Additional Resources

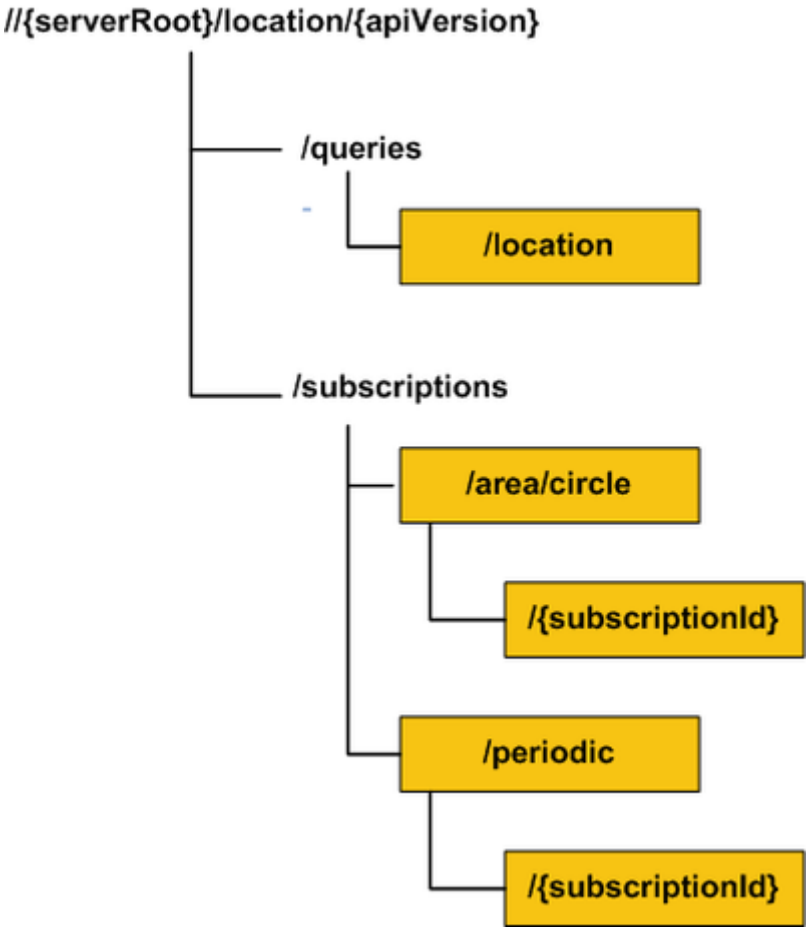
You can download the most current version of this document from the FIWARE API specification website at the Summary of FI-WARE Open Specifications. For more details about the Location GE that this API is based upon, please refer to "Architecture Description of Location GE". Related documents, including an Architectural Description, are available at the same site."

9.3 General Location Server REST API Information

The specification provides resource definitions, the HTTP verbs applicable for each of these resources, and the element data structures, as well as support material including flow diagrams and examples using the various supported message body formats (i.e. XML).

9.3.1 Resources Summary

The *{apiVersion}* URL variable SHALL have the value “v1” to indicate that the API corresponds to this version of the specification. See **REST_NetAPI_Common** which specifies the semantics of this variable.



serverRoot = server base url (hostname+port)

9.3.2 Authentication

No specific authentication scheme is put in place at HTTP level (no SSL over HTTP). Applicative authentication is performed thanks to request parameters.

9.3.3 Representation Format

Important notice :

The request support different data serialization format :

- XML : the request format specified in the Content-Type header is supposed to be **application/xml** MIME type.
- form-urlencoded : the request format specified in the Content-Type header is supposed to be a**pplication/x-www-form-urlencoded MIME** type.
- JSON : the request format specified in the Content-Type header is supposed to be **application/json** MIME type.

Note: only the request body is encoded as application/x-www-form-urlencoded, the response is still encoded as XML or JSON depending on the preference of the client and the capabilities of the server. Names and values MUST follow the application/x-www-form-urlencoded character escaping rules from W3C_URLENC .

Different format examples are provided for each kind of services, when applicable.

9.3.4 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.

9.3.5 Resource Identification

The resource identification used by the API in order to identify unambiguously the resource will be provided over time. For HTTP transport, this is made using the mechanisms described by HTTP protocol specification as defined by IETF RFC-2616.

9.3.6 Links and References

None

9.3.7 Limits

A maximum of 15 location query requests per second is allowed.

9.3.8 Versions

Querying the version is NOT supported (already included in the resources tree).

9.3.9 Faults

Please find below a list of possible fault elements and error codes

Associated Error Codes	**Description**	**Expected in All Requests?**
400 (“Bad Request”)	The document in the entity-body, if any, contains an error message. Hopefully the client can understand the error message and use it to fix the problem.	[YES]
404 (“Not Found”)	The requested URI doesn’t map to any resource. The server has no clue what the client is asking for.	[YES]
500 (“Internal Server Error”)	There’s a problem on the server side. The document in the entity-body, if any, is an error message. The error message probably won’t do much good, since the client can’t fix the server problem.	[YES]

9.4 Data Types

9.4.1 XML NameSpaces

The XML namespace for the Terminal Location data types is:

urn:oma:xml:rest:netapi:terminallocation:1

The 'common' namespace prefix is used in the present document refers to the XML namespace of the data types defined in **REST_NetAPI_Common** :

urn:oma:xml:rest:netapi:common:1

9.4.2 Requester

This section details the requester string format accepted by the API.

The format has the following string pattern : ***<service> : <password>.***

To be authorized, the following condition shall be met :

- Service <service> must exists in the LOCS database
- Service must be associated to a ServiceProvider with access password equal to <password>

9.4.3 Structures

This subsection describes the XML structure used in the Terminal Location API.

9.4.3.1 *Type:TerminalLocation*

A type containing device address, retrieval status and location information. As this can be related to a query of a group of terminal devices, the locationRetrievalStatus element is used to indicate whether the information for the device was retrieved or not, or if an error occurred.

Element	Type	Optional	Description
address	xsd:anyURI	No	Address of the terminal to which the location information (tel URI)
locationRetrievalStatus	common:RetrievalStatus	No	Status of retrieval for this terminal address
currentLocation	LocationInfo	Yes	Location of terminal. It is only provided if location Retrieval Status = Retrieved.
errorInformation	common:ServiceError	Yes	Must be included when location Retrieval Status = Error. This is the reason for the error.

9.4.3.2 *Type:TerminalLocationList*

A type containing a list of terminal locations.

Element	Type	Optional	Description
terminalLocation	TerminalLocation[1..unbounded]	No	Collection of the terminal locations.

9.4.3.3 *Type:LocationInfo*

A type containing location information with latitude, longitude and altitude, in addition the accuracy and a timestamp of the information are provided.

Element	Type	Optional	Description
latitude	xsd:float	No	Location latitude.
longitude	xsd:float	No	Location longitude.
altitude	xsd:float	Yes	Location altitude.
accuracy	xsd:int	No	Accuracy of location provided in meters.
timestamp	xsd:datetime	No	Date and time that location was collected.

9.4.3.4 *Type:PeriodicNotificationSubscription*

A type containing data for periodic notification.

Element	Type	Optional	Description
clientCorrelator	xsd:string	Yes	A correlator that the client MAY use to tag this particular resource representation during a request to create a resource on the server. In case the element is present, the server SHALL not alter its value, and SHALL provide it as part of the representation of this resource. In case the element is not present, the server SHALL NOT generate it.
resourceURL	xsd:anyURI	Yes	Self referring URL. The resourceURL SHALL NOT be included in POST requests by the client, but MUST be included in POST requests representing notifications by the server to the client, when a complete representation of the resource is embedded in the notification. The resourceURL MUST also be included in responses to any HTTP method that

			returns an entity body, and in PUT requests.
link	common:Link [0..unbounded]	Yes	Link to other resources that are in relationship with the resource.
callbackReference	common:CallbackReference	No	Notification callback definition. See **REST_NetAPI_Common** for details
requester	xsd:datetime	Yes	Mandatory for POST request for subscription creation. It identifies the entity that is requesting the Information. See **Requester** detailed format
address	xsd:anyURI	No	Addresses of terminal to monitor (e.g tel URI).
requestedAccuracy	xsd:float	No	Accuracy of the provided location in meters.
frequency	xsd:int	No	Maximum frequency (in seconds) of notifications per subscription (can also be considered minimum time between notifications).
duration	xsd:int	No	Period of time (in seconds) notifications are provided for.

9.4.3.5 *Type:CircleNotificationSubscription*

A type containing data for notification, when the area is defined as a circle.

Element	Type	Optional	Description
clientCorrelator	xsd:string	Yes	A correlator that the client MAY use to tag this particular resource representation during a request to create a resource on the server. In case the element is present, the server SHALL not alter its value, and SHALL provide it as part of the representation of this resource. In case the element is not present, the server SHALL NOT generate it.
resourceURL	xsd:anyURI	Yes	Self referring URL. The resourceURL SHALL NOT be included in POST requests by the client, but MUST be included in POST requests representing notifications by the server to the client, when a complete representation of the resource is embedded in the notification.

			The resourceURL MUST also be included in responses to any HTTP method that returns an entity body, and in PUT requests.
link	common:Link[0..unbounded]	Yes	Link to other resources that are in relationship with the resource.
callbackReference	common:CallbackReference	No	Notification callback definition.See **REST_NetAPI_Common** for details
requester	xsd:datetime	Yes	Mandatory for POST request for subscription creation. It identifies the entity that is requesting the Information.See **Requester** detailed format
address	xsd:anyURI	No	Addresses of terminal to monitor (e.g tel URI).
latitude	xsd:float	No	Latitude of center point.
longitude	xsd:float	No	Lontitude of center point.
radius	xsd:int	No	Radius of circle around center point in meters.
trackingAccuracy	xsd:float	No	Number of meters of acceptable error in tracking location.
enteringLeavingCriteria	EnteringLeavingCriteria	No	Indicates whether the notification should occur when the terminal enters or leaves the target area.
frequency	xsd:int	No	Maximum frequency (in seconds) of notifications per subscription (can also be considered minimum time between notifications).
duration	xsd:int	No	Period of time (in seconds) notifications are provided for.
count	xsd:int	No	Maximum number of notifications.

9.4.3.6 *Type:SubscriptionNotification*

A type containing the notification subscription.

Element	Type	Optional	Description
callbackData	xsd:string	Yes	CallbackData if passed by the application in the receipt Request element during the associated subscription operation.See **REST_NetAPI_Common** for details
terminalLocation	TerminalLocation[1..unbounded]	No	Collection of the terminal locations.
enteringLeavingCriteria	EnteringLeavingCriteria	Yes	Indicates whether the notification was caused by the terminal entering or leaving the target area.
link	common:Link[0..unbounded]	Yes	Link to other resources that are in relationship with the resource.

9.4.3.7 *Type:SubscriptionCancellationNotification*

A type containing the subscription cancellation notification.

Element	Type	Optional	Description
callbackData	xsd:string	Yes	CallbackData if passed by the application in the receipt Request element during the associated subscription operation.See **REST_NetAPI_Common** for details
address	xsd:anyURI	Yes	Address of terminal if the error applies to.
reason	common:ServiceError	No	Reason notification is being discontinued.
link	common:Link[0..unbounded]	Yes	Link to other resources that are in relationship with the resource.

9.4.3.8 *Type:RequestError*

A type containing request error response description.

Element	Type	Optional	Description
serviceException	common:serviceException	Yes	Used when request execution fails (format error, position method failure, etc..)
policyException	common:policyException	Yes	Used when request execution is not authorized.

9.5 API Operations

The following chapter give a detailed overview of the resources defined in this specification, the data type of their representation, the allowed HTTP methods, and some examples.

9.5.1 Location Query

Purpose: poll terminal location

Resource	HTTP Verb	Base URI	Data Structures	Description
Terminal location	GET	http://{serverRoot}/location/{apiVersion}/queries/location	TerminalLocationList	return current location of the terminal or multiple terminals

This figure below shows a scenario to return location for a single terminal or a group of terminals.

The resource:

- To get the location information for a single terminal or a group of terminals, read the resource below with the URL parameters containing terminal address or addresses

http://{serverRoot}/location/{apiVersion}/queries/location

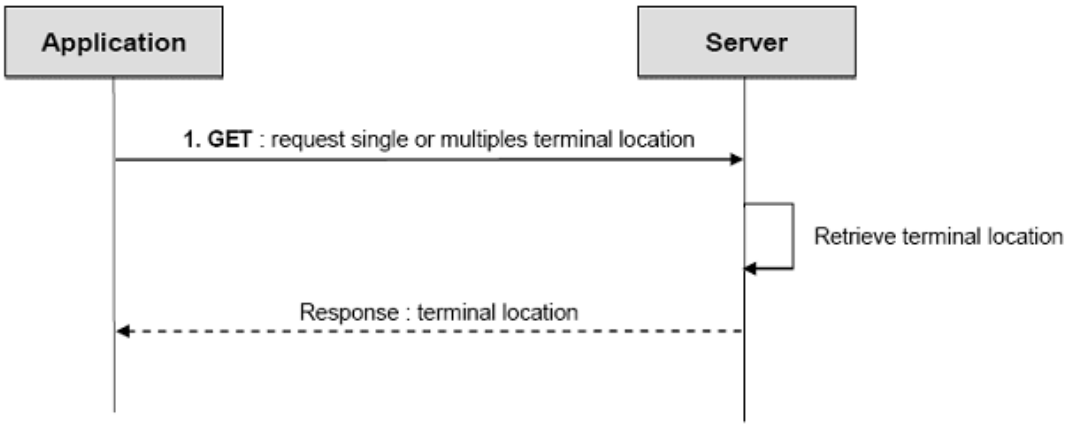


Figure 2 Location query

Outline of flow:

1. An application requests the distance between two terminals by using GET with the resource URL and providing two different terminal addresses as Request URL parameters.
2. It receives the terminal distance information.

9.5.1.1 *Detailed resources description*

GET Request :

If the format of the request is not correct, a **ServiceException** will be returned If the requester parameter is present and the requester is not authorized, a **PolicyException** will be returned.

Name	Type	Optional	Description
requester	xsd:anyURI	No	It identifies the entity that is requesting the information (See Requester specific format). If the requester is not authorized to retrieve location info, a policy exception will be returned.
address	xsd:anyURI [1..unbounded]	No	Address(es) of the terminal device(s) for which the location information is requested. Examples: (e.g tel URI tel:+19585550100,..)
requestedAccuracy	xsd:int	No	Accuracy of location information requested.
acceptableAccuracy	xsd:int	No	Accuracy that is acceptable for a response.
maximumAge	xsd:int	Yes	Maximum acceptable age (in seconds) of the location information that is returned.
responseTime	xsd:int	Yes	Indicates the maximum time (in seconds)

			that the application can accept to wait for a response.
tolerance	DelayTolerance	No	Indicates the priority of response time versus accuracy.

9.5.1.2 *Response codes*

Code	Description
200	Request is OK
400	Request is KO

9.5.1.3 *Examples*

Application/xml format

Example 1: (one terminal address, qop (quality of positioning accuracy) acceptable but does not match requested one)

Request:

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

Response:

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

 <?xml version="1.0" encoding="UTF-8"?>
 <tl:terminalLocationList
xmlns:common="urn:oma:xml:rest:netapi:common:1"
xmlns:tl="urn:oma:xml:rest:netapi:terminallocation:1">
  <tl:terminalLocation>
   <tl:address>33611223344</tl:address>
   <tl:locationRetrievalStatus>Retrieved
   </tl:locationRetrievalStatus>
   <tl:currentLocation>
    <tl:latitude>49.999737</tl:latitude>
    <tl:longitude>-60.00014</tl:longitude>
    <tl:altitude>30.0</tl:altitude>
    <tl:accuracy>55</tl:accuracy>
```

```
    <tl:timestamp>2012-04-17T09:21:32.893+02:00</tl:timestamp>
   </tl:currentLocation>
   <tl:errorInformation>
    <common:messageId>QOP_NOT_ATTAINABLE</common:messageId>
    <common:text>The requested QoP cannot be provided.</common:text>
   </tl:errorInformation>
  </tl:terminalLocation>
 </tl:terminalLocationList>
```

Example 2: (format error, missing address)

Request:

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

Response:

```
 HTTP/1.1 400 BadRequest
 Content-Type: application/xml
 Content-Length: nnnn
 Date: Thu, 02 Jun 2011 02:51:59 GMT


 <?xml version="1.0" encoding="UTF-8"?>
 <common:RequestError
xmlns:common="urn:oma:xml:rest:netapi:common:1"
xmlns:tl="urn:oma:xml:rest:netapi:terminallocation:1">
  <common:serviceException>
   <common:messageId>FORMAT_ERROR</common:messageId>
   <common:text> A protocol element in the request has invalid
format.</common:text>
  </common:serviceException>
 </common:RequestError>
```

Example 3: (unauthorized requester, bad password)

Request:

```
GET
/location/v1/queries/location?requester=test:badpassword&address=336
11223344&requestedAccuracy=50&acceptableAccuracy=60
 &maximumAge=100&tolerance=DelayTolerant HTTP/1.1
 Accept: application/xml
 Host: example.com
```

Response:

```
 HTTP/1.1 400 BadRequest
 Content-Type: application/xml
 Content-Length: nnnn
 Date: Thu, 02 Jun 2011 02:51:59 GMT

 <?xml version="1.0" encoding="UTF-8"?>
 <common:RequestError
xmlns:common="urn:oma:xml:rest:netapi:common:1"
xmlns:tl="urn:oma:xml:rest:netapi:terminallocation:1">
  <common:policyException>
   <common:messageId>UNAUTHORIZED_APPLICATION</common:messageId>
   <common:text>The requested location-based application is not
allowed to access the location server or a wrong password has been
supplied.</common:text>
  </common:policyException>
 </common:RequestError>
```

Application/json format

Request:

```
 GET
location/v1/queries/location?requester=test:test&address=33611223344
&tolerance=LowDelay&requestedAccuracy=1000
&acceptableAccuracy=1000 HTTP/1.1
 Content-Type: application/json
 Accept: application/json
 Host: example.com
```

Response:

```
 HTTP/1.1 200 OK
 Content-Type: application/json
```

```
Content-Length: nnnn

{"terminalLocationList": {"terminalLocation": {
"address": "tel:+19585550100",
"currentLocation": {
"accuracy": "100",
"altitude": "1001.0",
"latitude": "-80.86302",
"longitude": "41.277306",
"timestamp": "2011-06-04T00:27:23.000Z"
},
"locationRetrievalStatus": "Retrieved"
}}}
```

9.5.2 Periodic Notification Subscription

Purpose: Periodic location subscription

This resource is used to control subscriptions for periodic location notification for a particular client.

Resource	HTTP Verb	Base URI	Data Structures	Description
Periodic notification subscriptions	POST	http://{serverRoot}/location/{apiVersion}/subscriptions/periodic	PeriodicNotificationSubscription	create new subscription.
Periodic individual notification subscription	DELETE	http://{serverRoot}/location/{apiVersion}/subscriptions/periodic/{subscriptionid}	None	delete one subscription.
Client notifications on periodic terminal location retrieved	POST	Notification URL provided by client in notification subscription	SubscriptionNotification or SubscriptionCancellation Notification	signal notification

This figure below shows a scenario to control subscriptions for periodic notifications about terminal location for a particular client.

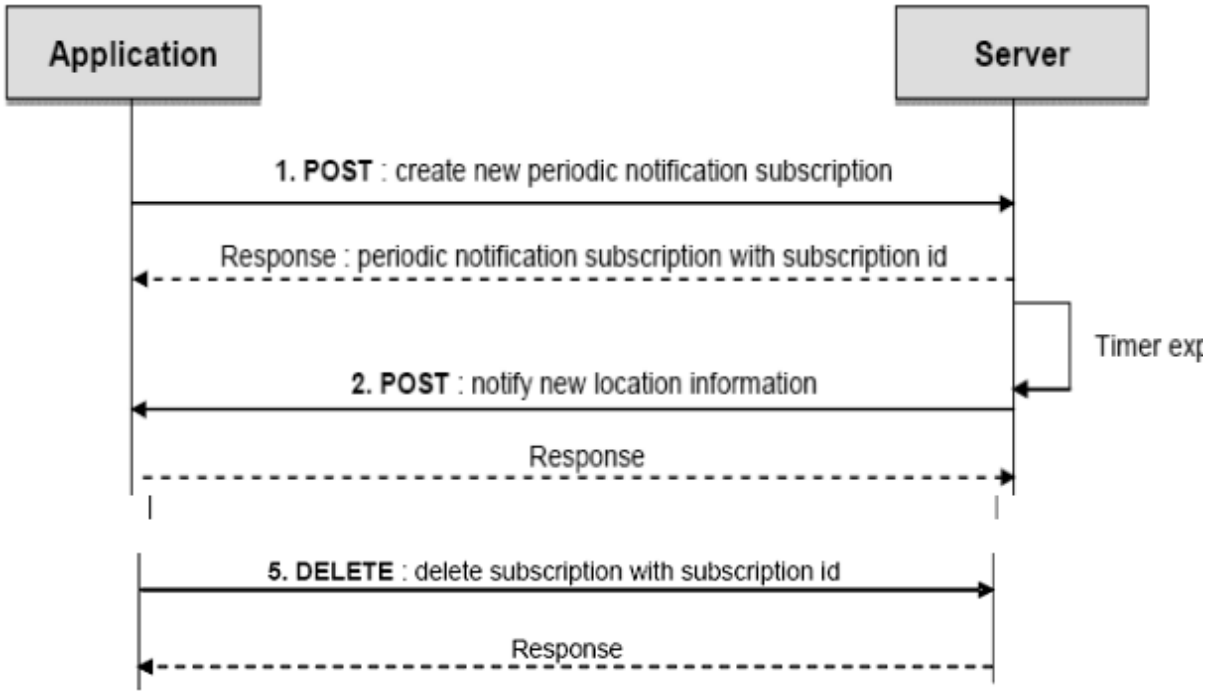
The resource:

- To start subscription to periodic notifications about terminal location for a particular client, create new resource under

http://{serverRoot}/location/{apiVersion}/subscriptions/periodic

- To delete an individual subscription for an individual subscription for periodic notifications about terminal location for a particular client, use the resource

http://{serverRoot}/location/{apiVersion}/subscriptions/periodic/{subscriptionId}



Outline of flow:

1. An application creates a new periodic notification subscription for the requesting client by using POST and receives the resulting resource URL containing subscriptionId.

2. At set-up frequency, The REST service on the server notifies the application of current location information using POST to the application supplied notifyURL.

3. An application deletes a subscription for periodic location notification and stops notifications for a particular client by using DELETE to resource URL containing subscriptionId.

9.5.2.1 *Detailed resources description*

POST Request :

This operation is used to create new periodic notification subscription for the requesting client. If correlator parameter is set, this value is used to build a predictable subscription URL with the a variable end string part of 'sub<correlator string>'

If the format of the request is not correct, a ServiceException will be returned If the requester parameter is present and the requester is not authorized, a PolicyException will be returned.

DELETE Request :

This operation is used to delete a subscription for periodic location notifications and stop notifications for a particular client. No URL parameters.

9.5.2.2 *Response Codes*

Code	Description
201	Subscription request created
204	No content
400	Request is KO

9.5.2.3 *Examples*

Application/xml format

Example 1: Add new subscription

Request:

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

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

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

```
 </tl:periodicNotificationSubscription>
```

Response:

```
 HTTP/1.1 201 Created
 Content-Type: application/xml
 Location:
http://example.com/location/v1/subscriptions/periodic/sub003
 Content-Length: nnnn
 Date: Thu, 02 Jun 2011 02:51:59 GMT

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

<tl:resourceURL>http://example.com/location/v1/subscriptions/periodi
c/sub0003</tl:resourceURL>
 <tl:callbackReference>

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

Subscription notification:

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

 <?xml version="1.0" encoding="UTF-8"?>
 <tl:subscriptionNotification
xmlns:common="urn:oma:xml:rest:netapi:common:1"
xmlns:tl="urn:oma:xml:rest:netapi:terminallocation:1">
 <tl:callbackData>4444</tl:callbackData>
 <tl:terminalLocation>
 <tl:address>tel:+19585550100</tl:address>
 <tl:locationRetrievalStatus>Retrieved</tl:locationRetrievalStatus>
```

```
 <tl:currentLocation>
 <tl:latitude>-80.86302</tl:latitude>
 <tl:longitude>41.277306</tl:longitude>
 <tl:altitude>1001.0</tl:altitude>
 <tl:accuracy>100</tl:accuracy>
 <tl:timestamp>2011-06-02T00:27:23.000Z</tl:timestamp>
 </tl:currentLocation>
 </tl:terminalLocation>
 <tl:link rel="PeriodicNotificationSubscription"
href="http:/location/v1/subscriptions/periodic/sub0003"/>
 </tl:subscriptionNotification>
```

Example 2: Delete subscription

Request:

```
DELETE /location/v1/subscriptions/periodic/sub0003 HTTP/1.1
Accept: application/xml
Host: example.com
```

Response:

```
HTTP/1.1 204 No Content
Date: Thu, 02 Jun 2011 02:51:59 GMT
```

Application/x-www-form-urlencoded format

Example 1: Add new subscription

Request:

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

 clientCorrelator=0003&

notifyURL=http%3A%2F%2Fapplication.example.com%2Fnotifications%2FLoc
ationNotification&
 callbackData=4444&
 address=tel%3A%2B19585550100&
```

```
 requestedAccuracy=10&
 frequency=10&
 duration=100
```

Response:

```
 HTTP/1.1 201 Created
 Content-Type: application/xml
 Location:
http://example.com/location/v1/subscriptions/periodic/sub003
 Content-Length: nnnn
 Date: Thu, 02 Jun 2011 02:51:59 GMT

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

<tl:resourceURL>http://example.com/location/v1/subscriptions/periodi
c/sub0003</tl:resourceURL>
 <tl:callbackReference>

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

Application/json format

Example 1: Add new subscription

Request:

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

 {"periodicNotificationSubscription": {
 "address": "tel:+19585550100",
 "callbackReference": {
 "callbackData": "4444",
```

```
 "notifyURL":
"http://application.example.com/notifications/LocationNotification"
 },
 "checkImmediate": "true",
 "clientCorrelator": "0003",
 "frequency": "10",
 "duration": "100",
 "requestedAccuracy": "10"
 }}
```

Response:

```
 HTTP/1.1 201 Created
 Content-Type: application/json
 Location:
http://example.com/location/v1/subscriptions/periodic/sub0003
 Content-Length: nnnn

 {"periodicNotificationSubscription": {
 "address": "tel:+19585550100",
 "callbackReference": {
 "callbackData": "4444",
 "notifyURL":
"http://application.example.com/notifications/LocationNotification"
 },
 "checkImmediate": "true",
 "clientCorrelator": "0003",
 "frequency": "10",
 "duration": "100",
 "resourceURL":
"http://example.com/exampleAPI/location/v1/subscriptions/periodic/su
b003",
 "requestedAccuracy": "10"
 }}
```

9.5.3 Area (Circle) Notification Subscription

Purpose: Area location subscription

Resource	HTTP Verb	Base URI	Data Structures	Description
Area (circle)notification subscriptions	POST	"http://{serverRoot}/location/{apiVersion}/subscriptions/area/circle"	CircleNotificationSubscription	create new subscription.
Area (circle)individual notification subscription	DELETE	"http://{serverRoot}/location/{apiVersion}/subscriptions/area/circle/{subscriptionid}"	None	delete one subscription.
Client notifications on terminal location changes	POST	Notification URL provided by client in notification subscription	SubscriptionNotification or SubscriptionCancellationNotification	signal notification

This figure below shows a scenario to control subscriptions for notification about terminal movement in relation to the geographic area (circle), crossing in and out, for a particular client.

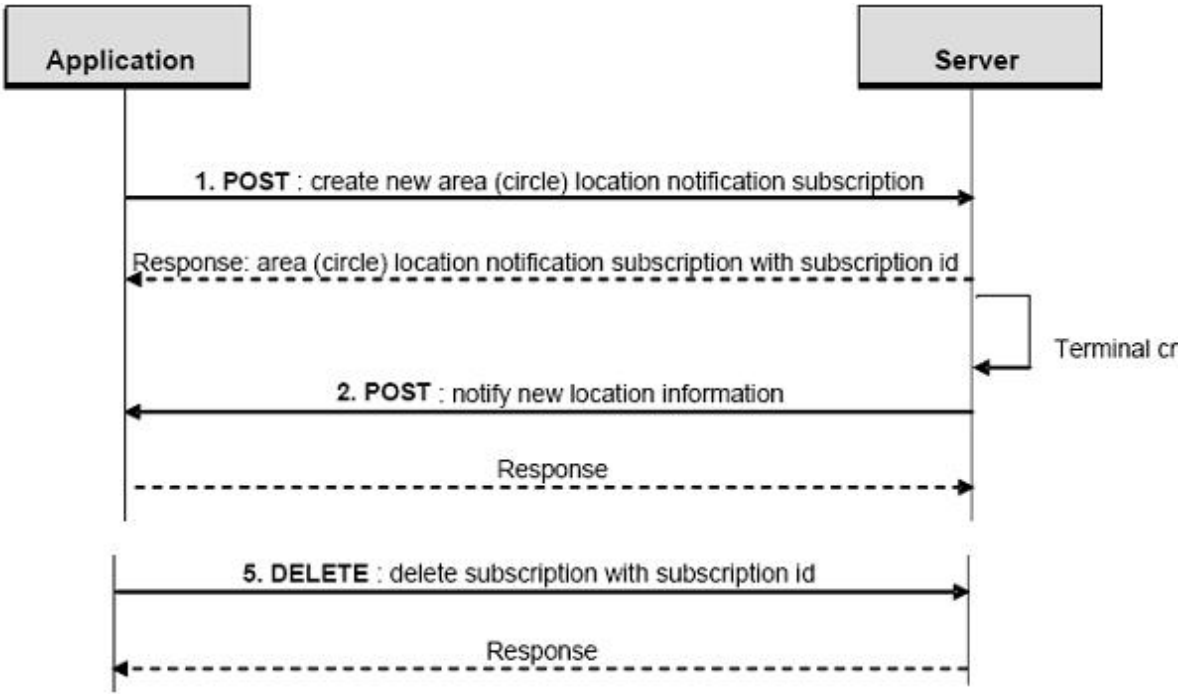
The resource:

- To start subscription to notifications about terminal movements in relation to the geographic area (circle), crossing in and out, for a particular client, create new resource under

http://{serverRoot}/location/{apiVersion}/subscriptions/area/circle

- To delete an individual subscription for notifications about terminal movements in relation to the geographic area (circle), crossing in and out, for a particular client, use the resource

http://{serverRoot}/location/{apiVersion}/subscriptions/area/circle/{subscriptionId}



Outline of flow:

1. An application creates a new periodic notification subscription for the requesting client by using POST and receives the resulting resource URL containing subscriptionId.

2. When the terminal crosses in or out the specified area (circle), The REST service on the server notifies the application of current location information using POST to the application supplied notifyURL.

3. An application deletes a subscription for periodic location notification and stops notifications for a particular client by using DELETE to resource URL containing subscriptionId.

9.5.3.1 *Detailed resources description*

POST Request :

This operation is used to create new movement notification subscription for the requesting client. If correlator parameter is set, this value is used to build a predictable subscription URL with the a variable end string part of 'sub<correlator string>'

If the format of the request is not correct, a ServiceException will be returned If the requester parameter is present and the requester is not authorized, a PolicyException will be returned.

DELETE Request :

This operation is used to delete a subscription for periodic location notifications and stop notifications for a particular client. No URL parameters.

9.5.3.2 *Response Codes*

Code	Description
201	Subscription request created
204	No content
400	Request is KO

9.5.3.3 *Examples*

Application/xml format

Example 1: Add new subscription

Request:

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

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

<tl:notifyURL>http://application.example.com/notifications/LocationN
otification</tl:notifyURL>
   <tl:callbackData>4444</tl:callbackData>
 </tl:callbackReference>
 <tl:address>tel:+19585550100</tl:address>
 <tl:latitude>100.23</latitude>
 <tl:longitude>-200.45</tl :longitude>
 <tl:radius>500</tl :radius>
 <tl:trackingAccuracy>10</tl:trackingAccuracy>
 <tl:enteringLeavingCriteria>Entering</tl:enteringLeavingCriteria>
 <tl:checkImmediate>true</tl:checkImmediate>
 <tl:frequency>10</tl:frequency>
 <tl:duration>100</tl:duration>
 <tl:count>10</tl:count>
 </tl:circleNotificationSubscription>
```

Response:

```
 HTTP/1.1 201 Created
 Content-Type: application/xml
 Location:
http://example.com/location/v1/subscriptions/area/circle/sub003
 Content-Length: nnnn
 Date: Thu, 02 Jun 2011 02:51:59 GMT

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

<tl:resourceURL>http://example.com/location/v1/subscriptions/area/ci
rcle/sub0003</tl:resourceURL>
 <tl:callbackReference>

<tl:notifyURL>http://application.example.com/notifications/LocationN
otification</tl:notifyURL>
 <tl:callbackData>4444</tl:callbackData>
 </tl:callbackReference>
 <tl:address>tel:+19585550100</tl:address>
 <tl:latitude>100.23</tl :latitude>
 <tl:longitude>-200.45</tl :longitude>
 <tl:radius>500</radius>
 <tl:trackingAccuracy>10</tl:trackingAccuracy>
 <tl:enteringLeavingCriteria>Entering</tl:enteringLeavingCriteria>
 <tl:checkImmediate>true</tl:checkImmediate>
 <tl:frequency>10</tl:frequency>
 <tl:duration>100</tl:duration>
 <tl:count>10</tl:count>
 </tl:circleNotificationSubscription>
```

Subscription notification:

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

 <?xml version="1.0" encoding="UTF-8"?>
 <tl:subscriptionNotification
xmlns:common="urn:oma:xml:rest:netapi:common:1"
xmlns:tl="urn:oma:xml:rest:netapi:terminallocation:1">
 <tl:callbackData>4444</tl:callbackData>
 <tl:terminalLocation>
```

```
 <tl:address>tel:+19585550100</tl:address>
 <tl:locationRetrievalStatus>Retrieved</tl:locationRetrievalStatus>
 <tl:currentLocation>
 <tl:latitude>-80.86302</tl:latitude>
 <tl:longitude>41.277306</tl:longitude>
 <tl:altitude>1001.0</tl:altitude>
 <tl:accuracy>100</tl:accuracy>
 <tl:timestamp>2011-06-02T00:27:23.000Z</tl:timestamp>
 </tl:currentLocation>
 </tl:terminalLocation>
 <tl:enteringLeavingCriteria>Entering</tl:enteringLeavingCriteria>
 <tl:link rel="CircleNotificationSubscription"
href="http://example.com/location/v1/subscriptions/area/circle/sub00
03"/>
 </tl:subscriptionNotification>
```

Example 2: Delete subscription

Request:

```
 DELETE /location/v1/subscriptions/area/circle/sub0003 HTTP/1.1
 Accept: application/xml
 Host: example.com
```

Response:

```
 HTTP/1.1 204 No Content
 Date: Thu, 02 Jun 2011 02:51:59 GMT
```

Application/x-www-form-urlencoded format

Example 1: Add new subscription

Request:

```
 POST /location/v1/subscriptions/area/circle HTTP/1.1
 Accept: application/xml
 Host: example.com
 Content-Type: application/x-www-form-urlencoded
 Content-Length: nnnn

 clientCorrelator=0003&

notifyURL=http%3A%2F%2Fapplication.example.com%2Fnotifications%2FLoc
ationNotification&
 callbackData=4444&
 address=tel%3A%2B19585550100&
```

```
 latitude=100.23&
 longitude=-200.45&
 radius=500&
 trackingAccuracy=10&
 enteringLeavingCriteria=Entering&
 checkImmediate=true&
 frequency=10&
 duration=100&
 count=10
```

Response:

```
 HTTP/1.1 201 Created
 Content-Type: application/xml
 Location:
http://example.com/location/v1/subscriptions/area/circle/sub003
 Content-Length: nnnn
 Date: Thu, 02 Jun 2011 02:51:59 GMT

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

<tl:resourceURL>http://example.com/location/v1/subscriptions/area/ci
rcle/sub0003</tl:resourceURL>
 <tl:callbackReference>

<tl:notifyURL>http://application.example.com/notifications/LocationN
otification</tl:notifyURL>
 <tl:callbackData>4444</tl:callbackData>
 </tl:callbackReference>
 <tl:address>tel:+19585550100</tl:address>
 <tl:latitude>100.23</tl :latitude>
 <tl:longitude>-200.45</tl :longitude>
 <tl:radius>500</radius>
 <tl:trackingAccuracy>10</tl:trackingAccuracy>
 <tl:enteringLeavingCriteria>Entering</tl:enteringLeavingCriteria>
 <tl:checkImmediate>true</tl:checkImmediate>
 <tl:frequency>10</tl:frequency>
 <tl:duration>100</tl:duration>
 <tl:count>10</tl:count>
 </tl:circleNotificationSubscription>
```

Application/json format

Example 1: Add new subscription

Request:

```
 POST /location/v1/subscriptions/area/circle HTTP/1.1
 Content-Type: application/json
 Accept: application/json
 Host: example.com
 Content-Length: nnnn

 {"circleNotificationSubscription": {
 "address": "tel:+19585550100",
 "callbackReference": {
 "callbackData": "4444",
 "notifyURL":
"http://application.example.com/notifications/LocationNotification"
 },
 "checkImmediate": "true",
 "clientCorrelator": "0003",
 "enteringLeavingCriteria": "Entering",
 "frequency": "10",
 "duration": "100",
 "count": "10",
 "latitude": "100.23",
 "longitude": "-200.45",
 "radius": "500",
 "trackingAccuracy": "10"
 }}
```

Response:

```
 HTTP/1.1 201 Created
 Content-Type: application/json
 Location:
http://example.com/location/v1/subscriptions/area/circle/sub0003
 Content-Length: nnnn

 {"circleNotificationSubscription": {
 "address": "tel:+19585550100",
 "callbackReference": {
 "callbackData": "4444",
 "notifyURL":
"http://application.example.com/notifications/LocationNotification"
 },
 "checkImmediate": "true",
 "clientCorrelator": "0003",
 "enteringLeavingCriteria": "Entering",
 "frequency": "10",
 "duration": "100",
```

```
 "count": "10",
 "latitude": "100.23",
 "longitude": "-200.45",
 "radius": "500",
 "resourceURL":
"http://example.com/exampleAPI/location/v1/subscriptions/area/circle
/sub0003",
 "trackingAccuracy": "10"
 }}
```

10 FIWARE OpenSpecification Data MetadataPreprocessing

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

Name	FIWARE.OpenSpecification.Data.MetadataPreprocessing
Chapter	Data/Context Management,
Catalogue-Link to Implementation	<Meta-data Pre-processing>
Owner	Siemens AG, Peter Amon

10.1 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.

10.1.1 Copyright

- Copyright © 2012 by SIEMENS

10.1.2 Legal Notice

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

10.2 Overview

10.2.1 Target usage

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

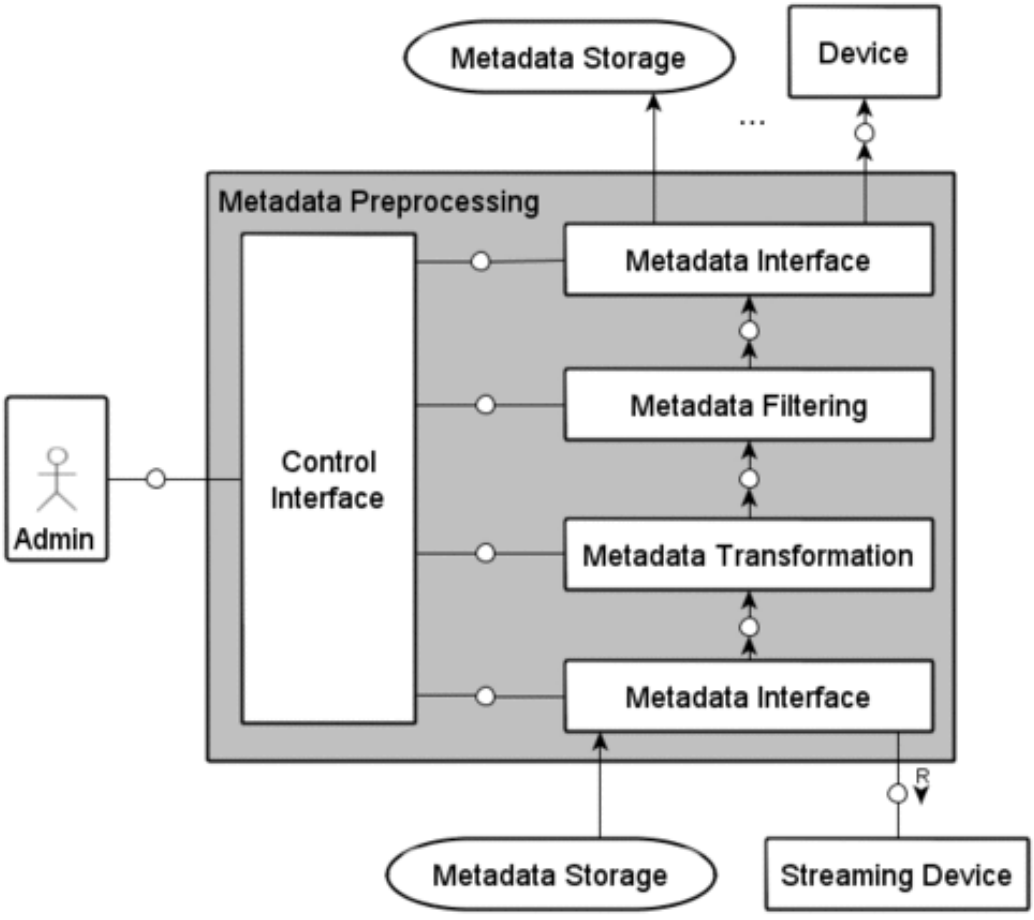
10.2.2 Example scenarios and main services exported

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

10.3 Basic Concepts

10.3.1 Functional components of the Metadata Preprocessing GE

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



Functional components of the Metadata Preprocessing GE

The functionality of the components is described in the following.

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

10.3.2 Realization by MetadataProcessor asset

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

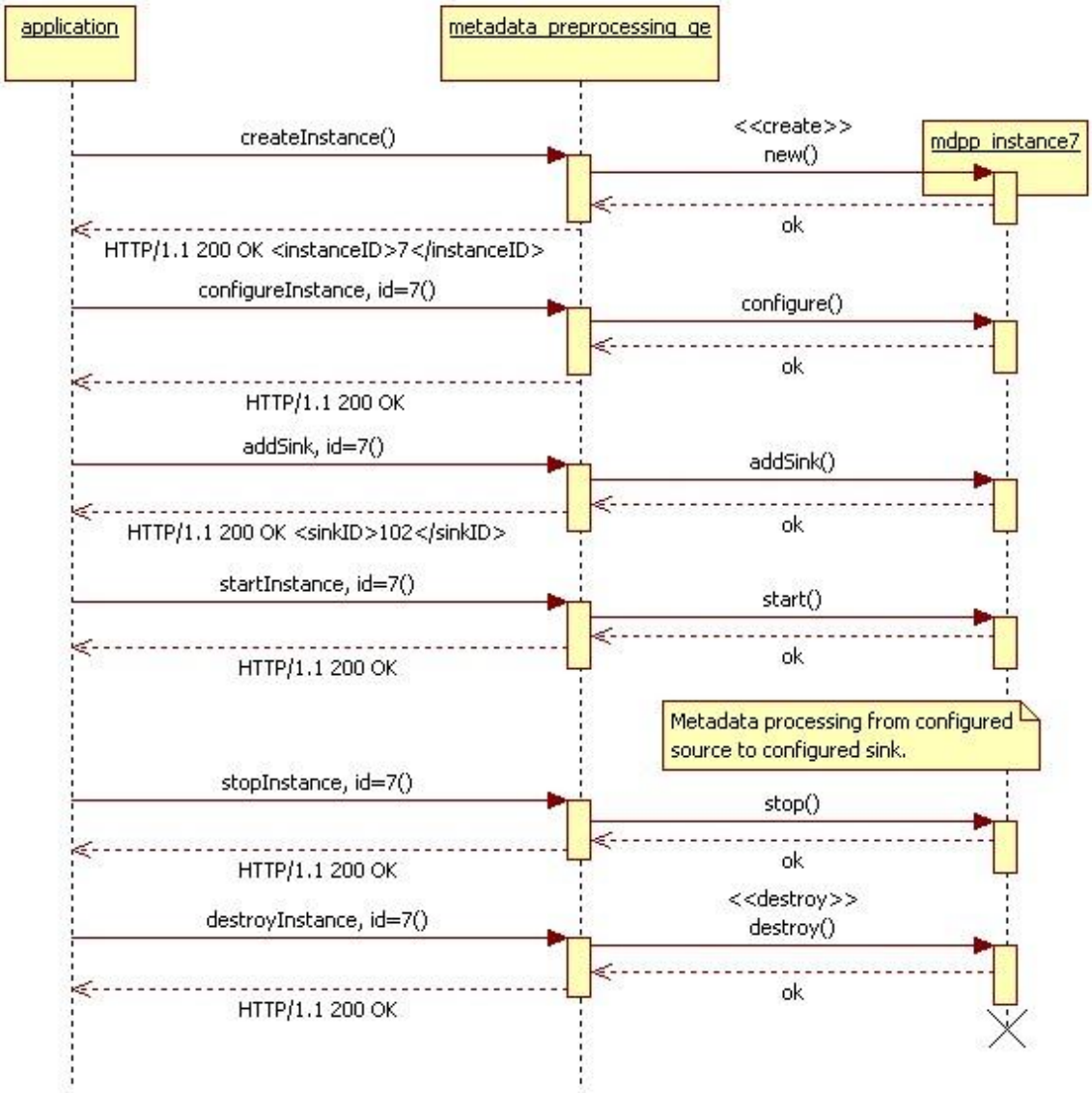
10.4 Main Interactions

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

- ***getVersion***: The version of the Metadata Preprocessing GE is returned.
- ***listInstances***: All instances (i.e., processing units) of the Metadata Preprocessing GE are listed.
- ***createInstance***: An instance for processing metadata streams/events is created.
- ***getInstanceInfo***: The information about a specific instance (i.e., processing unit) is returned.

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

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



Example scenario

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

10.5 Basic Design Principles

The following basic design principles apply:

- The Metadata Preprocessing GE realizes a generic metadata transformation approach, which is not restrictive to specific metadata schemes.
- Encapsulation of transport and metadata transformation can be implemented as-a-service, usable from other web applications or components.

- Transformation can be based on standardized and commonly used XML Stylesheet Language for Transformations (XSLT).
- In addition to encapsulation in (XML- or JSON-based) metadata formats, also incorporation of the metadata into objects (e.g., serialized Java/C++/C# classes) can be realized (by simply exchanging the stylesheet for the XSLT).

10.6 References

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

10.7 Detailed Specifications

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

10.7.1 Open API Specifications

- Metadata Preprocessing Open RESTful API Specification (PRELIMINARY)

10.8 Re-utilised Technologies/Specifications

The following technologies/specifications are incorporated in this GE:

- Extensible Stylesheet Language Transformation (XSLT) Version 1.0 as defined by the W3C,
- Real-time Transport Protocol (RTP) / RTP Control Protocol (RTCP) as defined in RFC 3550,
- Real-Time Streaming Protocol (RTSP) as defined in RFC 2326.

10.9 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.

- **Data** refers to information that is produced, generated, collected or observed that may be relevant for processing, carrying out further analysis and knowledge extraction. Data in FI-WARE has associated a **data type** and a**value**. FI-WARE will support a set of built-in **basic data types** similar to those existing in most programming languages. Values linked to basic data types supported in FI-WARE are referred as **basic data values**. As an example, basic data values like '2', '7' or '365' belong to the integer basic data type.
- A **data element** refers to data whose value is defined as consisting of a sequence of one or more <name, type, value> triplets referred as **data element attributes**, where the type and value of each attribute is either mapped to a basic data type and a basic data value or mapped to the data type and value of another data element.
- **Context** in FI-WARE is represented through **context elements**. A context element extends the concept of **data element** by associating an EntityId and EntityType to it, uniquely identifying the entity (which in turn may map to a group of entities) in the FI-WARE system to which the context element information refers. In addition, there may be some attributes as well as meta-data associated to attributes that we may define as mandatory for context elements as compared to data elements. Context elements are typically created containing the value of attributes characterizing a given entity at a given moment. As an example, a context element may contain values of some of the attributes "last measured temperature", "square meters" and "wall color" associated to a room in a building. Note that there might be many different context elements referring to the same entity in a system, each containing the value of a different set of attributes. This allows that different applications handle different context elements for the same entity, each containing only those attributes of that entity relevant to the corresponding application. It will also allow representing updates on set of attributes linked to a given entity: each of these updates can actually take the form of a context element and contain only the value of those attributes that have changed.
- An **event** is an occurrence within a particular system or domain; it is something that has happened, or is contemplated as having happened in that domain. Events typically lead to creation of some data or context element describing or representing the events, thus allowing them to processed. As an example, a sensor device may be measuring the temperature and pressure of a given boiler, sending a context element every five minutes associated to that entity (the boiler) that includes the value of these to attributes (temperature and pressure). The creation and sending of the context element is an event, i.e., what has occurred. Since the data/context elements that are generated linked to an event are the way events get visible in a computing system, it is common to refer to these data/context elements simply as "events".
- A **data event** refers to an event leading to creation of a data element.
- A **context event** refers to an event leading to creation of a context element.

- An **event object** is used to mean a programming entity that represents an event in a computing system [EPIA] like event-aware GEs. Event objects allow to perform operations on event, also known as **event processing**. Event objects are defined as a data element (or a context element) representing an event to which a number of standard event object properties (similar to a header) are associated internally. These standard event object properties support certain event processing functions.

11 Metadata Preprocessing Open RESTful API Specification (PRELIMINARY)

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

11.1 Introduction to the Metadata Preprocessing GE API

11.1.1 Metadata Preprocessing GE API Core

The Metadata Preprocessing GE API is a RESTful, resource-oriented API accessed via HTTP that uses XML-based representations for information interchange.

Please check the FI-WARE Open Specifications Legal Notice to understand the rights to use FI-WARE Open Specifications.

11.1.2 Intended Audience

This specification is intended for both software/application developers and application providers. This document provides a full specification of how to interoperate with the Metadata Preprocessing GE. To use this information, the reader should firstly have a general understanding of the Metadata Preprocessing GE (see Metadata Preprocessing GE Product Vision). You should also be familiar with:

- RESTful web services,
- HTTP/1.1,
- XML data serialization formats.

11.1.3 API Change History

This version of the Metadata Preprocessing GE API Guide replaces and obsoletes all previous versions. The most recent changes are described in the table below:

Revision Date	Changes Summary
May 1, 2012	Initial version
May 16, 2012	Revision of API to support several MDPP instances
Oct 10, 2012	Revision due to internal review
Nov 8, 2012	Revision due to 3rd internal review
...	

11.1.4 How to Read This Document

In the whole document it is taken the assumption that reader is familiarized with REST architecture style. Along the document, some special notations are applied to differentiate some special words or concepts. The following list summarizes these special notations.

- A bold, mono-spaced font is used to represent code or logical entities, e.g., HTTP method (GET, PUT, POST, DELETE).
- An italic font is used to represent document titles or some other kind of special text, e.g., URI.
- The variables are represented between brackets, e.g. {id} and in italic font. When the reader find it, can change it by any value.

For a description of some terms used along this document, see Metadata Preprocessing GE Architecture.

11.1.5 Additional Resources

You can download the most current version of this document from the FIWARE API specification website at <link to the url>. For more details about the Metadata Preprocessing GE service that this API is based upon, please refer to Metadata Preprocessing GE Product Vision.

11.2 General Metadata Preprocessing GE API Information

11.2.1 Resources Summary

The resource summary is shown in the following figure.

```
                                    |--- ?action=stop              PUT ->
stopInstance
                                    |
                                    |--- /config                   GET ->
getConfig
                                    |                              PUT ->
configureInstance
                                    |
                                    |--- /sinks                    GET ->
listSinks
                                           |                     POST ->
addSink
                                           |
                                           |--- /{sinkID}          GET ->
getSinkInfo
                                                                DELETE ->
removeSink

Sink (client)
-------------
//{sinkNotificationURI}
```

11.2.2 Authentication

Authentication is not supported in Version 1 of the Metadata Preprocessing GE.

11.2.3 Representation Format

The Multimedia Analysis GE API supports XML-based representation formats for both requests and responses. This is specified by setting the Content-Type header to application/xml, if the request/response has a body. Note: In addition, the Metadata Preprocessing GE API supports XML-based representations for the payload metadata to be processed (i.e., transformed and/or filtered).

11.2.4 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. Note: In addition, payload metadata is transmitted between the Metadata Preprocessing GE and connected components by using RTP, as defined by IETF RFC-3550. In future versions, resource representation might also be transmitted by using HTTP 1.1 protocol, as defined by IETF RFC-2616.

11.2.5 Resource Identification

The resource identification for HTTP transport is made using the mechanisms described by HTTP protocol specification as defined by IETF RFC-2616.

11.2.6 Links and References

Request forwarding is not supported in Version 1 of the Metadata Preprocessing GE.

11.2.7 Limits

Limits are not yet specified for Version 1 of the Metadata Preprocessing GE.

11.2.8 Versions

Querying the version is not yet specified for Version 1 of the Metadata Preprocessing GE.

11.2.9 Extensions

Querying extensions is not supported in Version 1 of the Metadata Preprocessing GE.

11.2.10 Faults

Fault elements and their associated error codes are described in the following table.

Fault Element	Error Code	Reason Phrase	Description	Expected in All Requests?
POST, **GET**, **PUT**, **DELETE**	400	Bad Request	The client sent a request the server is not able to process. The message body may contain a detailed description of this error.	[YES]
POST, **GET**, **PUT**, **DELETE**	404	Not Found	The requested URI does not map any resource.	[YES]
POST, **GET**, **PUT**, **DELETE**	405	Method Not Allowed	The used HTTP method is not allowed for the requested resource. The message body may contain a detailed description of this error.	[YES]
POST, **GET**, **PUT**, **DELETE**	500	Internal Server Error	An unforeseen error occurred at the server. The message body may contain a detailed description of this error.	[YES]

11.3 API Operations

11.3.1 Version

Verb	URI	Description
GET	//{serverRoot}/{assetName}/version	***getVersion***: returns the current version of the Metadata Preprocessing GE realization/asset (e.g., MetadataProcessor)

11.3.1.1 *getVersion*

Example:

```
GET //198.51.100.24/mdp/version HTTP/1.1
Accept: application/xml
```

Sample result:

```
HTTP/1.1 200 OK
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<version>1.02</version>
```

11.3.2 Management of instances

Verb	URI	Description
GET	//{serverRoot}/{assetName}/instances	***listInstances***: lists all instances (i.e., processing units) of the Metadata Preprocessing GE
POST	//{serverRoot}/{assetName}/instances	***createInstance***: creates an instance (i.e., a processing unit) of the Metadata Preprocessing GE
DELETE	//{serverRoot}/{assetName}/instance/{instanceID}	***destroyInstance***: destroys a specific instance (i.e., processing unit)
PUT	//{serverRoot}/{assetName}/instance/{instanceID}?action=start	***startInstance***: starts the processing of the processing unit
PUT	//{serverRoot}/{assetName}/instance/{instanceID}?action=stop	***stopInstance***: stops the processing of the processing unit

11.3.2.1 *listInstances*

Example:

```
GET //198.51.100.24/mdp/instances HTTP/1.1
Accept: application/xml
```

Sample result:

```
HTTP/1.1 200 OK
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<instances>
  <instance id="7" sourceURI="rtsp://203.0.113.1/stream1"
activeSinks="1"/>
  <instance id="89" sourceURI="rtsp://203.0.113.15/stream5"
activeSinks="3"/>
</instances>
```

11.3.2.2 *createInstance*

Example:

```
POST //198.51.100.24/mdp/instances HTTP/1.1
Accept: application/xml
```

Sample result:

```
HTTP/1.1 200 OK
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<instanceID>7</instanceID>
```

11.3.2.3 *destroyInstance*

Example:

```
DELETE //198.51.100.24/mdp/instances/7 HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

11.3.2.4 *startInstance*

Example:

```
PUT //198.51.100.24/mdp/instances/7?action=start HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

11.3.2.5 *stopInstance*

Example:

```
PUT //198.51.100.24/mdp/instances/7?action=stop HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

11.3.3 Configuration of Instances

Verb	URI	Description
GET	//{serverRoot}/{assetName}/instances/{instanceID}/config	***getConfig***: returns the configuration of an existing processing unit
PUT	//{serverRoot}/{assetName}/instances/{instanceID}/config	***configureInstance***: configures an existing processing unit
GET	//{serverRoot}/{assetName}/instances/{instanceID}/sinks	***listSinks***: lists all connected sinks of a specific processing unit
POST	//{serverRoot}/{assetName}/instances/{instanceID}/sinks	***addSink***: adds a sink for receiving the transformed/filtered metadata (e.g., another GE)
GET	//{serverRoot}/{assetName}/instances/{instanceID}/sinks/{sinkID}	***getSinkInfo***: returns the information about a specific sink

DELETE	//{serverRoot}/{assetName}/instances/{instanceID}/sinks/{sinkID}	***removeSink***: removes a specific sink

11.3.3.1 *getConfig*

Example:

```
GET //198.51.100.24/mdp/instances/7/config HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<configurationInstance>
  <source>
    <sourceURI>rtsp://203.0.113.1/stream1</sourceURI>
  </source>
  <processing>
    <plugin position="1" type="xslt">
      <xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:template match="/person_list">
          <object_list>
            <xsl:apply-templates />
          </object_list>
        </xsl:template>
        <xsl:template match="person">
          <object type="person">
            <id>
              <xsl:apply-templates select="id" />
            </id>
            <label>
              <xsl:apply-templates select="name" />
            </label>
          </object>
        </xsl:template>
      </xsl:stylesheet>
    </plugin>
  </processing>
</configurationInstance>
```

11.3.3.2 *configureInstance*

The following example demonstrates the transformation of a person list into a more generic object list. In order to configure the Metadata Preprocessing GE, a stylesheet is sent to the GE.

```
PUT //198.51.100.24/mdpp/instances/7/config HTTP/1.1
```

```
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<configurationInstance>
  <source>
    <sourceURI>rtsp://203.0.113.1/stream1</sourceURI>
  </source>
  <processing>
    <plugin position="1" type="xslt">
      <xsl:stylesheet                                    version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:template match="/person_list">
          <object_list>
            <xsl:apply-templates />
          </object_list>
        </xsl:template>
        <xsl:template match="person">
          <object type="person">
            <id>
              <xsl:apply-templates select="id" />
            </id>
            <label>
              <xsl:apply-templates select="name" />
            </label>
          </object>
        </xsl:template>
      </xsl:stylesheet>
    </plugin>
  </processing>
</configurationInstance>
```

Sample result:

```
HTTP/1.1 200 OK
```

With this configuration, incoming XML metadata is transformed. This is illustrated in the following example, which is kept simple for demonstration purposes.

Example input metadata stream:

```
<?xml version="1.0" encoding="UTF-8"?>
<person_list>
  <person>
    <id>09</id>
    <name>Guard01</name>
    <status>ClearanceLevel04</status>
  </person>
</person_list>
```

Example output metadata stream:

```
<?xml version="1.0" encoding="UTF-8"?>
<object_list>
  <object type="person">
    <id>09</id>
    <label>Guard01</label>
  </object>
</object_list>
```

As can be seen from the example, the transformation changes the label of the metadata and adds options to the XML elements. Furthermore, some metadata is filtered since it might not be needed by subsequent components.

11.3.3.3 *listSinks*

Example:

```
GET //198.51.100.24/mdp/instances/7/sinks HTTP/1.1
Accept: application/xml
```

Sample result:

```
HTTP/1.1 200 OK
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<sinks>
  <sink id="101" sinkURI="http://192.0.2.11/metadata1"/>
  <sink id="103" sinkURI="http://192.0.3.65/source234"/>
</sinks>
```

11.3.3.4 *addSink*

Example:

```
POST //198.51.100.24/mdp/instances/7/sinks HTTP/1.1
Content-Type: application/xml
Accept: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<configurationSink>
  <sinkURI>http://192.0.2.11/metadata1</sinkURI>
</configurationSink>
```

Sample result:

```
HTTP/1.1 200 OK
Content-Type: application/xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<sinkID>102</sinkID>
```

Sample message to listener (call-back):

```
PUT //192.0.2.11/metadata1 HTTP/1.1
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<configurationListener>
  <streamURI>rtsp://198.51.100.24/mdp/7/stream1</streamURI>
</configurationListener>
```

11.3.3.5 *getSinkInfo*

Example:

```
GET //198.51.100.24/mdp/instances/7/sinks/102 HTTP/1.1
Accept: application/xml
```

Sample result:

```
HTTP/1.1 200 OK
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<configurationSink>
  <sinkURI>http://192.0.2.11/metadata1</sinkURI>
</configurationSink>
```

11.3.3.6 *removeSink*

Example:

```
DELETE //198.51.100.24/mdp/instances/7/sinks/102 HTTP/1.1
```

Sample result:

```
HTTP/1.1 200 OK
```

12 FIWARE OpenSpecification Data PubSub Context Broker

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

Name	FIWARE.OpenSpecification.Data.PubSub
Chapter	Data/Context Management,
Catalogue-Link to Implementation	http://catalogue.fi-ware.eu/enablers/publishsubscribe http://catalogue.fi-ware.eu/enablers/publishsubscribe-samson-broker
Owner	Telecom Italia, Boris Moltchanov Telefónica I+D, Juanjo Hierro, Carlos Ralli, Fermín Galán, Ken Gunnar Zangelin

12.1 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.

12.1.1 Copyright

12.1.2 Legal Notice

12.2 Overview

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

Generally speaking, the Context Broker GE will enable publication of context information by entities, referred as Context Producers, so that published context information becomes available to other entities, referred as Context Consumers, which are interested in processing the published context information. Applications or even other GEs in the FI-WARE platform may play the role of Context Producers, Context Consumers or both. Events

in FI-WARE based systems refer to something that has happened, or is contemplated as having happened. As such, they map to updates or notification on updates on context information that can be handled by applications or FI-WARE GEs.

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

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

12.2.2 Target usage

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

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

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

12.2.3 Example Scenarios

The number of potential context sources permanently connected through 3G links, e.g. mobile user terminals, embedded sensors, microphones and cameras, is expected to increase significantly. By processing and inferring this raw information, a variety of useful information will be available in future communication and content management systems. It is likely for smart spaces to grow from smart homes/offices to urban smart spaces in which

plenty of artifacts and people are interconnected over distance. This will enable all sorts of innovative interactive pervasive applications as perceived by Weiser [1]

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

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

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

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

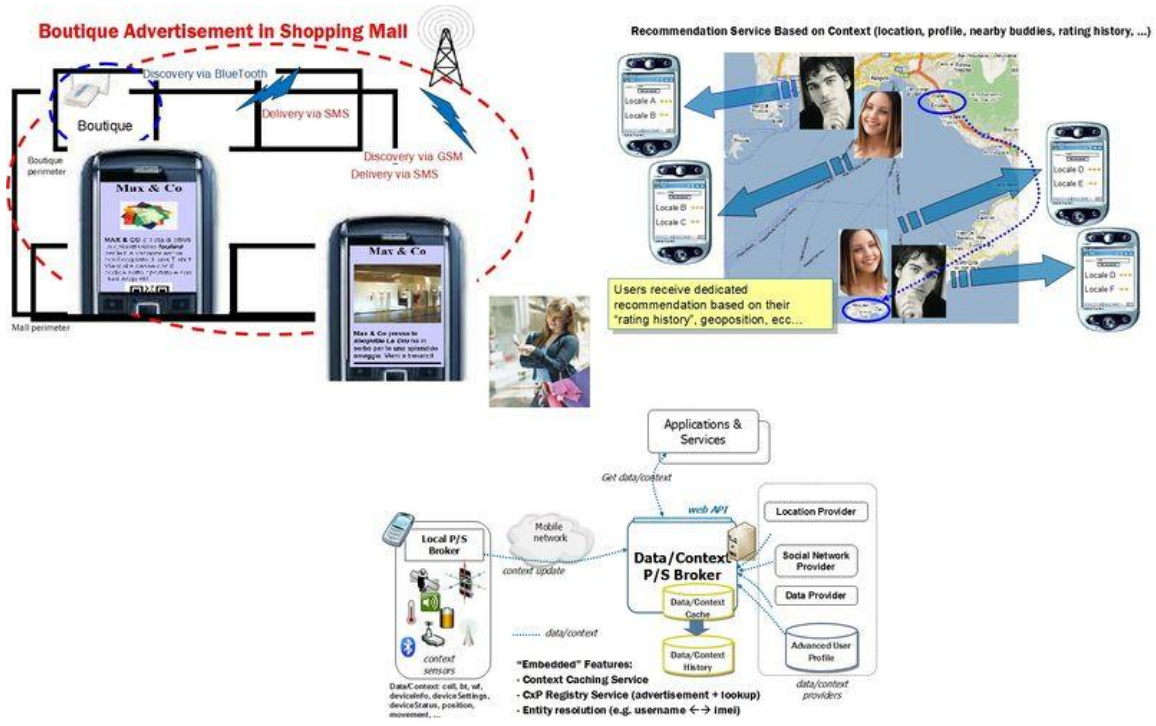


Figure: Context-aware Applications and Service enabled by the Publish/Subscribe GE

12.3 Basic Concepts

12.3.1 Context Elements

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

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

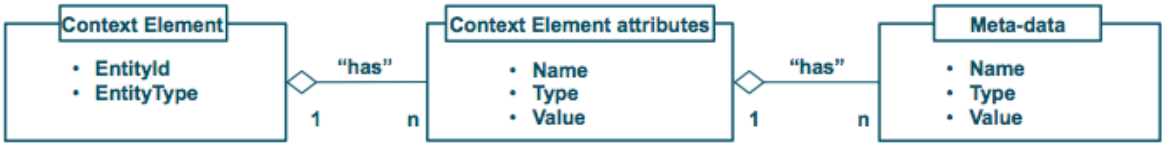
- An EntityId and EntityType, uniquely identifying the entity to which context data refers.
- A sequence of one or more data element attributes (<name, type, value> triplets)
- Optional meta-data linked to attributes (also <name, type, value> triplets)

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

- the “last measured temperature”, “square meters” and “wall color” attributes associated to a “room” in a building
- attributes “speed”, “geolocation”, “current established route” of a “car”
- attributes “message geolocation”, “message contents” of a “user message”
- attributes “creation time”, “priority” of an “alarm”
- attributes “last launch time”, “last shutdown” of a “process”

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

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



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

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

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

12.3.2 Basic Entities of the GE Model

12.3.2.1 *Context Broker*

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

12.3.2.2 *Context Provider*

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

12.3.2.3 *Context Source*

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

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

12.3.2.4 *Context Consumer*

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

12.3.2.5 *Entity*

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

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

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

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

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

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

12.3.2.6 *Context scopes*

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

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

12.3.3 Features and Functionalities

12.3.3.1 *Context caching*

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

12.3.3.2 *Context validity*

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

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

12.3.3.3 *Context history*

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

12.3.4 Fi-WARE NGSI Specification

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

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

12.4 Main Interactions using the FI-WARE NGSI Restful API

12.4.1 OMA NGSI Basics

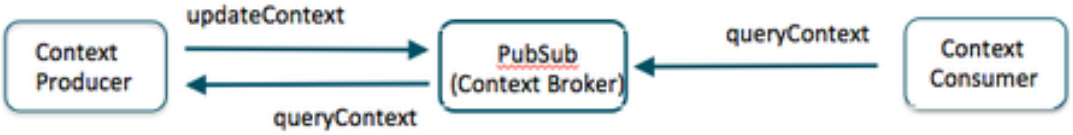
OMA NGSI (Next Generation Service Interface) Operations are grouped into two major interfaces:

- NGSI-10
  - updateContext
  - queryContext
  - subscribeContext / unsubscribeContext / updateContextSubscription
  - notifyContext
- NGSI-9
  - registerContext
  - discoverContextAvailability
  - subscribeContextAvailability / unsubscribeContextAvailability / updateContextAvailabilitySubscription
  - notifyContextAvailability

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

12.4.2 Basic Interactions and related Entities

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



The basic interactions are:

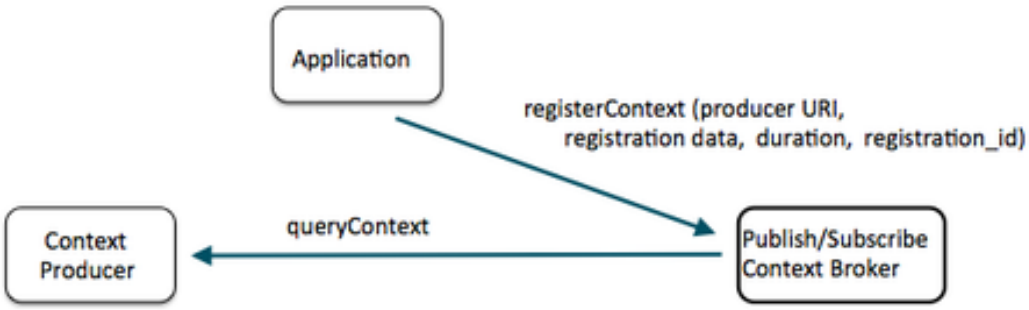
- Context Producers playing the role of Context Sources publish data/context elements by invoking the updateContext operation on a Context Broker.
- Some Context Producers may also play the role of Context Providers and export a queryContext operation Context Brokers may invoke at any given time to query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

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

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

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

- Third applications register information about a given Context Provider by means of invoking the registerContext operation that the Context Broker exports. Information about what entities the Context Provider can inform about is provided in the request.
- Context Brokers can retrieve data/context elements by invoking the queryContext operation on a Context Provider

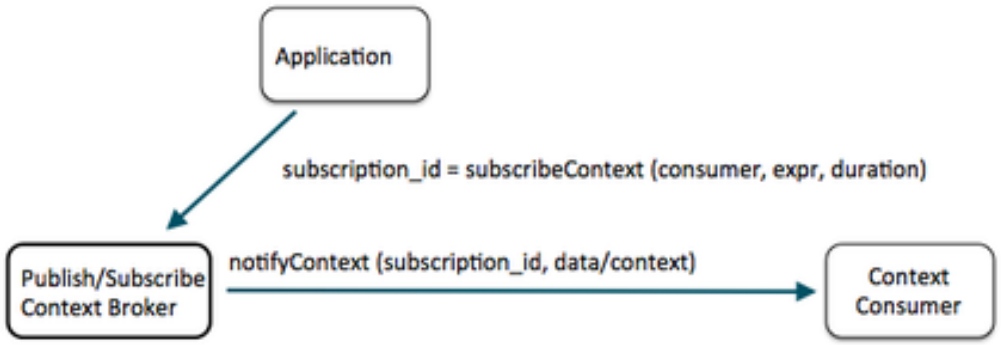


12.4.4 Interactions to subscribe Context Consumers to specific notifications

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

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

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



12.4.5 Extended Operations: Registering Entities & Attributes availability

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

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

12.4.6 Extended Operations: Applications subscription to Entities/Attributes registration

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

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

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

12.5 Main interactions using ContextML/CQL

12.5.1.1 *ContextML Basics*

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

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

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

ContextML allows the following features:

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

The ContextML schema is composed by:

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

12.5.1.2 *Context Data*

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

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

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

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

Figure 12: *civilAddres* Scope Example

12.5.1.3 *ContextML Naming Conventions*

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

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

12.5.2 ContextML API

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

12.5.3 ContextQL (CQL)

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

12.5.3.1 *Context Query*

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

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

The following actions can be represented in CQL:

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

The following conditions can be expressed in CQL:

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

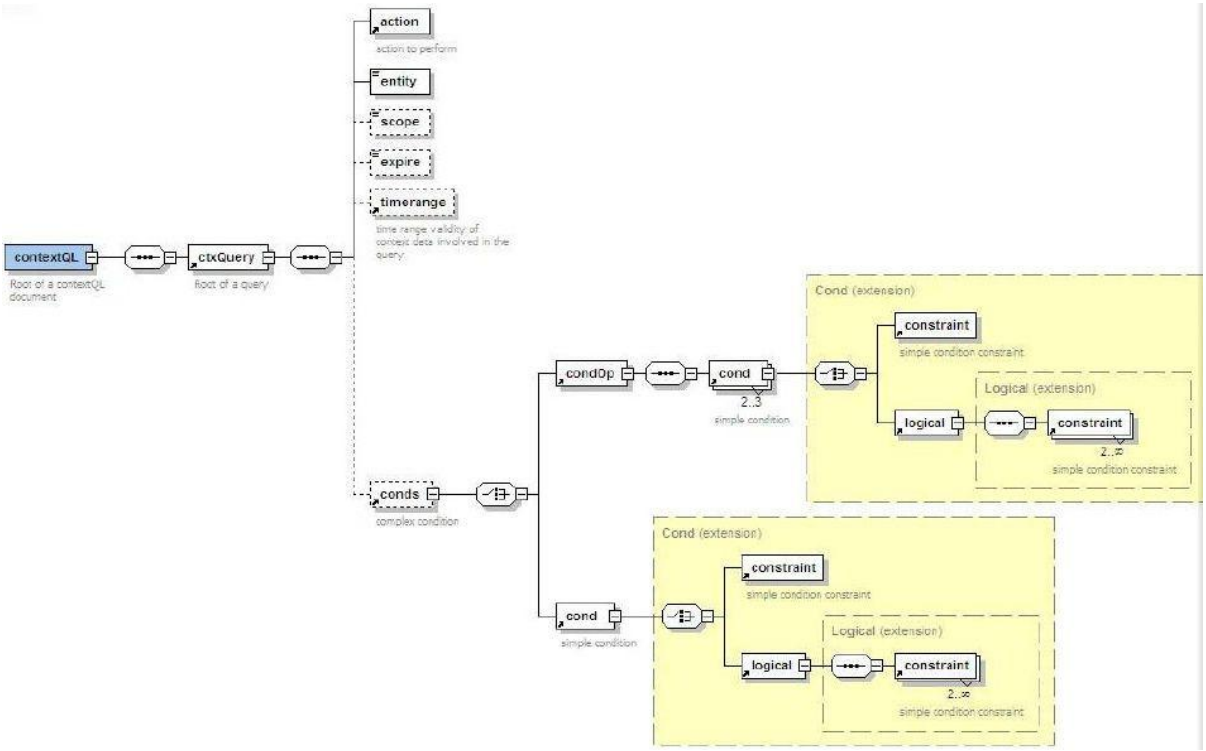


Figure 13: XSD schema of a ContextML query

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

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

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

Table 1: Combinations of possible conditions in the broker

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

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

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

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

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

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

Table 2: ContextQL operators

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

12.5.4 CQL API

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

12.6 Basic Design Principles

12.6.1 Conceptual Decoupling

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

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

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

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

12.7 References

[1]	Weiser, M., "The computer for the 21st century", Human-computer interaction: toward the year 2000, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.
[2]	Lamorte L., Licciardi C. A., Marengo M., Salmeri A., Mohr P., Raffa G., Roffia L., Pettinari M. & Salmon Cinotti T., 2007, “A platform for enabling context aware telecommunication services”, 3rd Workshop on Context Awareness for Proactive Systems, University of Surrey, UK, June 2007.
[3]	Moltchanov B., Knappmeyer M., Licciardi C. A., Baker N., 2008, “Context-Aware Content Shariing and Castiing”, ICIN2008, Bordeaux, France, UK, October 2008.
[4]	Open Mobile Alliance (OMA) Next Generation Services Interface (NGSI) Specification http://www.openmobilealliance.org/Technical/release_program/docs/NGSI/V1_0-20101207-C/OMA-TS-NGSI_Context_Management-V1_0-20100803-C.pdf
[5]	Moltchanov B., Knappmeyer M., Liaquat Kiani S., Fra' C., Baker N., 2010, “ContextML: A Light-Weight Context Representation and Context Management Schema”, IEEE International Symposium on Wireless Pervasive Computing, Modena, Italy, May 2010.
[6]	Sumi, Y., Etani, T., Fels, S., Simonet, N., Kobayashi, K. & Mase, K., 1998, “C-map: Building a context-aware mobile assistant for exhibition tours”, Community Computing and Support Systems, Social Interaction in Networked Communities, Springer-Verlag, UK, pp.137–154.
[7]	Cheverst, K., Davies, N., Mitchell, K., Friday, A. & Efs-tratiou, C., 2000, “Developing a context-aware electronic tourist guide: some issues and experiences”, Proceedings of the SIGCHI conference on Human Factors in Computing Systems, ACM Press, New York, USA, pp.17–24.
[8]	Moltchanov B., Fra' C., Valla, M., Licciardi C. A., 2011, “Context Management Framework and Context Representation for MNO”, Activity Context Workshop / AAAI2012, San Francisco, USA, August 2012.
[9]	Gu, T., Pung, H.K. & Zhang, D.Q., 2004, “A middleware for building context-aware

	mobile services", Proceedings of IEEE Vehicular Technology Conference (VTC), Milan, Italy
[10]	Fahy, P. & Clarke, S., 2004, "CASS – a middleware for mobile context-aware applications", Workshop on Context Awareness, MobiSys 2004.
[11]	MobiLife, an Integrated Project in European Union's IST 6th Framework Programme, http://www.ist-mobilife.org
[12]	Service Platform for Innovative Communication Environment (SPICE), An Integrated Project in European Union's IST 6th Framework Programme, http://www.ist-spice.org
[13]	Open Platform for User-centric service Creation and Execution (OPUCE), An Integrated Project in European Union's IST 6th Framework Programme.
[14]	Context-aware Content Casting (C-CAST), A Research Project in European Union's ICT 7th Framework Programme. Detailed Specifications

12.8 Detailed Specifications

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

12.8.1 Open API Specifications

- FI-WARE NGSI Open RESTful API Specification (PRELIMINARY)
- ContextML/CQL over HTTP Open RESTlike API Specification (PRELIMINARY)

12.8.2 Other Specifications

- ContextML Specification (PRELIMINARY)
- Context Query Language (CQL) Specification (PRELIMINARY)

12.9 Re-utilised Technologies/Specifications

The following technologies used for Pub/Sub GE implementation:

- JBoss
- J2EE
- JAX-RS

- MySQL

12.10 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.

- **Data** refers to information that is produced, generated, collected or observed that may be relevant for processing, carrying out further analysis and knowledge extraction. Data in FI-WARE has associated a **data type** and a**value**. FI-WARE will support a set of built-in **basic data types** similar to those existing in most programming languages. Values linked to basic data types supported in FI-WARE are referred as **basic data values**. As an example, basic data values like '2', '7' or '365' belong to the integer basic data type.
- A **data element** refers to data whose value is defined as consisting of a sequence of one or more <name, type, value> triplets referred as **data element attributes**, where the type and value of each attribute is either mapped to a basic data type and a basic data value or mapped to the data type and value of another data element.
- **Context** in FI-WARE is represented through **context elements**. A context element extends the concept of **data element** by associating an EntityId and EntityType to it, uniquely identifying the entity (which in turn may map to a group of entities) in the FI-WARE system to which the context element information refers. In addition, there may be some attributes as well as meta-data associated to attributes that we may define as mandatory for context elements as compared to data elements. Context elements are typically created containing the value of attributes characterizing a given entity at a given moment. As an example, a context element may contain values of some of the attributes "last measured temperature", "square meters" and "wall color" associated to a room in a building. Note that there might be many different context elements referring to the same entity in a system, each containing the value of a different set of attributes. This allows that different applications handle different context elements for the same entity, each containing only those attributes of that entity relevant to the corresponding application. It will also allow representing updates on set of attributes linked to a given entity: each of these updates can actually take the form of a context element and contain only the value of those attributes that have changed.
- An **event** is an occurrence within a particular system or domain; it is something that has happened, or is contemplated as having happened in that domain. Events typically lead to creation of some data or context element describing or representing the events, thus allowing them to processed. As an example, a sensor device may be measuring the temperature and pressure of a given boiler, sending a context element every five minutes associated to that entity (the boiler) that includes the value of these to attributes (temperature and pressure). The creation and sending of the context element is an event, i.e., what has occurred. Since the data/context elements that are generated linked to an event are the way events get visible in a computing system, it is common to refer to these data/context elements simply as "events".

- A **data event** refers to an event leading to creation of a data element.
- A **context event** refers to an event leading to creation of a context element.
- An **event object** is used to mean a programming entity that represents an event in a computing system [EPIA] like event-aware GEs. Event objects allow to perform operations on event, also known as **event processing**. Event objects are defined as a data element (or a context element) representing an event to which a number of standard event object properties (similar to a header) are associated internally. These standard event object properties support certain event processing functions.

13 FI-WARE NGSI Open RESTful API Specification (PRELIMINARY)

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

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 (PRELIMINARY)*
- FI-WARE NGSI-10 Open RESTful API Specification (PRELIMINARY)*

Following chapters comprise the specification of both interfaces.

14 FI-WARE NGSI-9 Open RESTful API Specification (PRELIMINARY)

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

14.1 Introduction to the FI-WARE NGSI-9 API

14.1.1 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)

14.1.2 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/NGSI-10 specification and binding documents for details on the resource structure and message formats.

14.1.3 Change history

This version of the <Name>API Guide 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

14.1.4 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.

14.1.5 Legal Notice

Please check the FI-WARE Open Specifications Legal Notice to understand the rights to use FI-WARE Open Specifications.

15 FI-WARE NGSI-9 Open RESTful API Specification (PRELIMINARY)

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

15.1 Introduction to the FI-WARE NGSI-9 API

15.1.1 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)

15.1.2 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/NGSI-10 specification and binding documents for details on the resource structure and message formats.

15.1.3 Change history

This version of the <Name>API Guide 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

15.1.4 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.

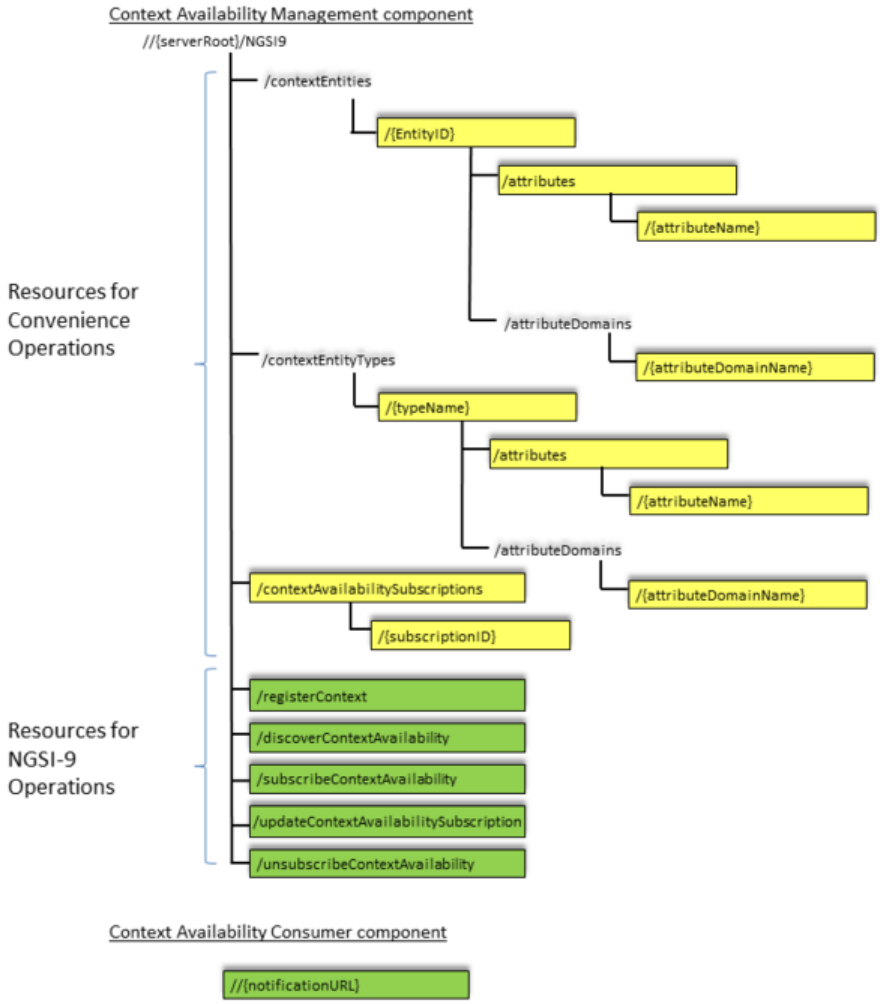
OMA NGSI-10 and OMA NGSI-9 share the same NGSI-9/NGSI-10 information model. Be sure to have read it before continuing on this page.

15.1.5 Legal Notice

Please check the FI-WARE Open Specifications Legal Notice to understand the rights to use FI-WARE Open Specifications.

15.2 General NGSI-9 API information

15.2.1 Resources Summary



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.

15.2.2 Representation Format

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

15.2.3 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.

15.2.4 API Operations on Context Management Component

15.2.4.1 *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 unsubscribeContextAvailabilityRequest; the

resource		response body is an instance of unsubscribeContextAvailabilityResponse.

15.2.4.2 *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 values from the domain	-	Register a provider of information about attributes from the domain	-
Context entity type	/contextEntityTypes/ {typeName}	Retrieve information on providers of any information about context	-	Register a provider of information about context entitie of the	-

		entities of the type		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 subscription container	/contextAvailabilitySubscriptions	-	-	Create a new availability subscription	-
Availability subscription	/contextAvailabilitySubscriptions/ {subscriptionID}	-	Update subscription	-	Cancel subscription

15.2.5 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.

16 FI-WARE NGSI-10 Open RESTful API Specification (PRELIMINARY)

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

16.1 Introduction to the FI-WARE NGSI 10 API

Please check the FI-WARE Open Specifications Legal Notice to understand the rights to use FI-WARE Open Specifications.

16.1.1 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)

16.1.2 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.

16.1.3 Change history

This version of the <Name>API Guide 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

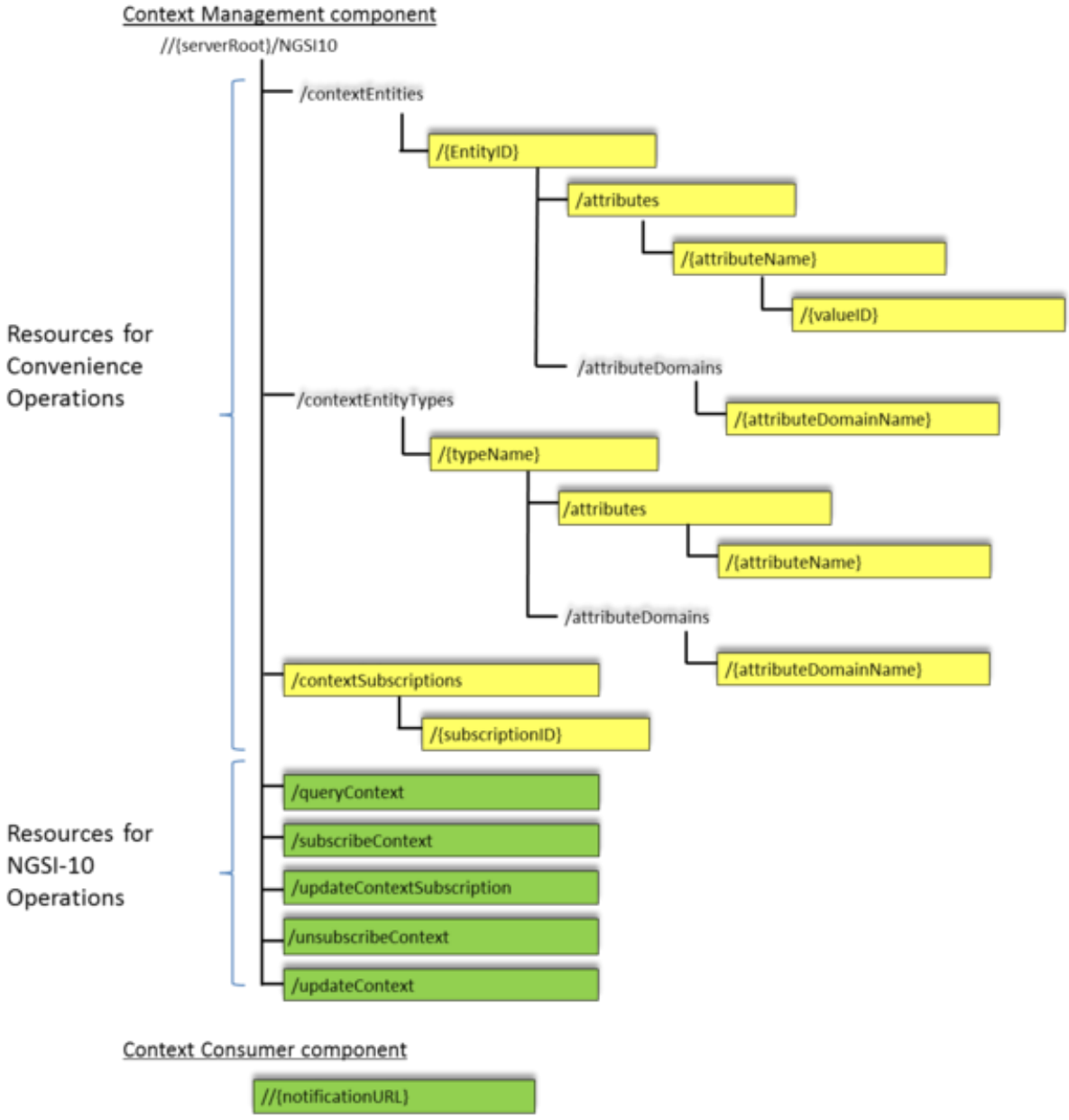
16.1.4 Additional Resources

The formal specification of OMA NGSI 10 can be downloaded from the website of theOpen 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.

16.2 General NGSI 10 API information

16.2.1 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.

16.2.2 Representation Format

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

16.2.3 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.

16.2.4 API Operations on Context Management Component

16.2.4.1 *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	/contextQuery	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	/updateContextSubscription	Generic update of context subscriptions. The

context subscription resource		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.

16.2.4.2 *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}/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 individual	/contextEntities/{EntityID}/ attributes/ {attributeName}/{attributeID}	Retrieve specific attribute value	Replace attribute value	-	Delete attribute value

context entity					
Attribute domain of individual context entity	/contextEntities/{EntityID}/ attributeDomains/ {attributeDomainName}	Retrieve all attribute information belonging to attribute domain	-	-	-
Context entity type	/contextEntityTypes/{typeName}	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

16.2.5 API operation on Context Consumer Component

This section describes the resource that has to be provided by the context consumer in order to receive 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.

17 ContextML API

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

17.1 Using ContextML to interact with the Publish/Subscribe GE

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

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

The returned data are formatted according to the Context Management Language (ContextML) proposed for this architecture.

17.2 ContextML Basics

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

ContextML allows the following features:

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

The ContextML schema is composed by:

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

17.2.1 Context Data

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

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

The below example reports context data provided from a CP that supports the *civilAddress* scope.

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

17.2.2 ContextML Naming Conventions

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

- names should be **lower case**, with the capital letters if composed by more than one word
    - example: *cell*, *longitude*, *netType*
- special chars like *_:/ must be avoided

- MAC addresses or Bluetooth IDs should be written without ':' separator, using capital letters
    - example: MAC address *00:22:6B:86:85:E3* should be represented: *00226B8685E3*

17.3 ContextML API

In the following paragraphs a description of the available method is given.

17.3.1 Announcement of a Context Provider: providerAdvertising method

A Context Provider (CP) that provides context information about one or more scope has to publish its presence to the Context Broker (CB). When a CP starts it has to send to the CB a ContextML document in which it specifies its name, its version, the entity types and the context scopes that are supported. Moreover the CP has to publish the URL to invoke and the input parameters it needs for context computation (if necessary). In this way when the CB receives a request for context information of a specific entity and a specific scope, it knows which one is the right CP to be invoked.

The context provider shall be announced invoking '**providerAdvertising(*ctxData*)** method within HTTP POST request where content type is set to *"application/x-www-form-urlencoded"*. **ctxData** = *"<?xml...> <contextML> ..... </contextML>"* or with a content type set to *"text/xml"* with body containing only ContextML document of the request. The URL is:

```
http://[server]/CB/ContextBroker/providerAdvertising
```

Here is an example of Context Provider announcement (the response will be an ACK, as described in a previous paragraph).

```
<contextML>
    <ctxAdvs>
        <ctxAdv>
            <contextProvider id="LP" v="1.1.0"/>
            <urlRoot>ca_example.tilab.com/LP</urlRoot>
            <scopes>
                <scopeDef n="position">
                    <url>/loc/getLocation</url>
                    <entityTypes>username,mobile</entityTypes>
                    <inputDef>
                        <inputEl name="phone" type="currentSettings:mobile"/>
                        <inputEl name="cgi" type="cell:cgi"/>
                        <inputEl name="btList" type="bt:btList"/>
                        <inputEl name="wfList" type="wf:wfList"/>
                    </inputDef>
                </scopeDef>
                <scopeDef n="civilAddress">
                    <url>/locInfo/getCivilAddress</url>
                    <entityTypes>username</entityTypes>
                    <inputDef>
                        <inputEl name="latitude" type="position:latitude"/>
                        <inputEl name="longitude" type="position:longitude"/>
                    </inputDef>
                </scopeDef>
            </scopes>
        </ctxAdv>
    </ctxAdvs>
</contextML>
```

17.3.2 Description of Context Providers: getContextProviders method

This methods allows to retrieve the list of Context Providers providing the specified scope. The HTTP GET request is as follows:

```
http://[server]/CB/ContextBroker/getContextProviders?scope=[scopeNam
e]
```

The following is an example with a description of available Context Providers.

```
<contextML>
  <ctxPrvEls>
    <ctxPrvEl>
      <par n="scope">position</par>
      <parA n="contextProviders">
        <parS n="contextProvider">
          <par n="id">LP</par>
          <par n="url">mobilife2.tilab.com/LP/loc/getLocation</par>
        </parS>
      </parA>
    </ctxPrvEl>
  </ctxPrvEls>
</contextML>
```

17.3.3 List of Available Context Scopes: getAvailableAtomicScopes method

The ContextML language allows the retrieval of the list of scope which is available to the Context Broker, with an HTTP GET request of the following type:

```
http://[server]/CB/ContextBroker/getAvailableAtomicScopes
```

The following is an example of a description of available context scopes.

```
<contextML>
   <scopeEls>
      <scopeEl>
         <parA n="scopes">
               <par n="scope">position</par>
               <par n="scope">civilAddress</par>
               <par n="scope">userProfile</par>
         </parA>
      </scopeEl>
   </scopeEls>
</contextML>
```

17.3.4 Context Update

In order to send a context element to context broker the method **"contextUpdate(ctxData)"** where the ctxData are specified as in the context provided announcement described above.

An HTTP POST data request should be sent as:

```
http://[server]/CB/ContextBroker/contextUpdate
```

The POST body should contain a ContextML message containing the context elements to be updated. The request's contentType must be set to *"text/xml"* and the Http content length must be set accordingly. Since the ContextML information is contained in the POST body, no request parameters are needed.

Here is an example, for a request of **"position"** for a device:

```
POST /CB/ContextBroker/contextUpdate HTTP/1.1
User-Agent: Mozilla/4.0
Host: prova
Content-Type: text/xml
Content-Length: 671
Connection: Keep-Alive
Cache-Control: no-cache

<?xml version="1.0" encoding="UTF-8"?>
<contextML xmlns="http://ContextML/1.7"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ContextML/1.7 ../ContextML-1.7.xsd">
   <ctxEls>
      <ctxEl>
       <contextProvider id="MyClient" v="1.2.1"/>
       <entity id="123456789123" type="imei"/>
       <scope>position</scope>
       <timestamp>2008-05-20T11:12:19+01:00</timestamp>
       <expires>2008-05-20T11:21:22+01:00</expires>
```

```
        <dataPart>
               <par n="latitude">45.11045277777778</par>
               <par n="longitude">7.675251944444445</par>
               <par n="accuracy">50</par>
               <par n="locMode">GPS</par>
        </dataPart>
       </ctxEl>
    </ctxEls>
</contextML>
```

Normally the context broker response is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<contextML xmlns="http://ContextML/1.7"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ContextML/1.7 .../ContextML-1.7.xsd">
 <ctxResp>
       <contextProvider id="CB" v="1.4.3"/>
       <timestamp>2008-05-20T17:55:42+02:00</timestamp>
       <entity id="123456789123" type="imei"/>
       <method>contextUpdate</method>
       <resp status="OK" code="200"/>
 </ctxResp>
</contextML>
```

If some error has occurred, the message describes the problem:

```
<?xml version="1.0" encoding="UTF-8"?>
<contextML xmlns="http://ContextML/1.7"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ContextML/1.7 .../ContextML-1.7.xsd">
 <ctxResp>
       <contextProvider id="CB" v="1.4.3"/>
       <timestamp>2008-05-20T16:11:56+02:00</timestamp>
       <entity id="123456789123" type="imei"/>
       <scope>position</scope>
       <method>contextUpdate</method>
       <resp status="ERROR" code="456" msg="Scope not defined"/>
 </ctxResp>
</contextML>
```

17.3.5 Get context

This method allows the retrieval of context elements from the CMF. The platform searches for valid context elements in cache, otherwise tries to update them with the help of context providers. Updated context information is stored into the cache. An HTTP GET data request should be sent to the server, containing the entity and the comma separated list of required scopes (scopeList parameter). Here is an example to require the scope “position” of a device:

```
http://[server]/CB/ContextBroker/getContext?entity=imei|123456789123
&scopeList=position
```

The context broker answer with the required context element, if available, as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<contextML xmlns="http://ContextML/1.7"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ContextML/1.7 .../ContextML-1.7.xsd">
<ctxEls>
       <ctxEl>
              <contextProvider id="MyClient" v="1.2.1"/>
              <entity id="123456789123" type="imei"/>
              <scope>position</scope>
              <timestamp>2008-05-20T11:12:19+01:00</timestamp>
              <expires>2008-05-20T11:21:22+01:00</expires>
              <dataPart>
                     <par n="latitude">45.11045277777778</par>
                     <par n="longitude">7.675251944444445</par>
                     <par n="accuracy">50</par>
                     <par n="locMode">GPS</par>
              </dataPart>
       </ctxEl>
</ctxEls>
</contextML>
```

If the required context element is not available, the response will be similar to:

```
<?xml version="1.0" encoding="UTF-8"?>
<contextML xmlns="http://ContextML/1.7"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ContextML/1.7 .../ContextML-1.7.xsd">
<ctxResp>
       <contextProvider id="CB" v="1.4.3"/>
       <timestamp>2008-05-20T16:11:56+02:00</timestamp>
       <entity id="123456789123" type="imei"/>
       <scope>cell</scope>
       <method>getContext</method>
       <resp status="ERROR" code="460" msg="Provider LP Returned:
404 - Not found: location not possible"/>
</ctxResp>
</contextML>
```

18 CQL API

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

18.1 ContextQL (CQL)

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

18.1.1 Context Query

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

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

The following actions can be represented in CQL:

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

request. An unsubscribe request can be implemented by a subscription with a certain subId of an existing subscription setting the validity period to zero.

The following conditions can be expressed in CQL:

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

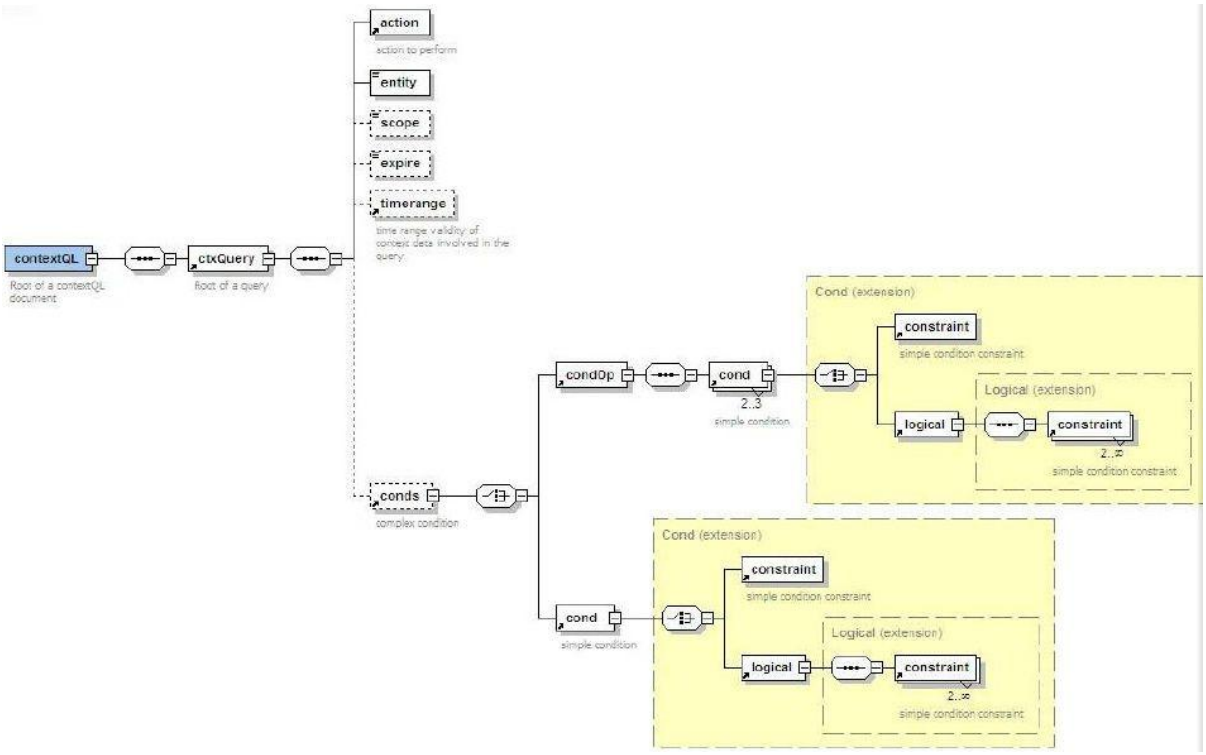


Figure: XSD schema of a ContextML query

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

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

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

Table 1: Combinations of possible conditions in the broker

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

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

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

- *param* – iodentifies parameters to which refers a condition and its value shall be the same context type to match e.g. *scope.par*, *scope.parS.par*, *scope.parA[n].par*. This attribute does not exist if the condition ONCLOCK
- *op* – identifies operator to apply to a parameter. This attribute exist only in the conditions ONVALUE. Currently defined attributes are of arithmetic and string-based types, which are listed in the below Table 2

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

Table 2: ContextQL operators

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

18.2 CQL API

In the following paragraphs a description of the available method is given.

18.2.1 Examples of Context Queries

18.2.1.1 *SELECT ONVALUE*

The SELECT ONVALUE allows to retrieve context data present at the time of the request, only if a context parameter has a specific value. Condition could be related also to context scopes not included in the returned data set.

```
<contextQL xmlns="http://ContextQL/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ContextQL/1.0
http://cark3.cselt.it/schemas/ContextQL-1.0.xsd">
  <ctxQuery>
    <action type="SELECT"/>
    <entity>username|sergio</entity>
    <scope>civilAddress</scope>
    <timerange from="2007-10-12T00:00:00+02:00"
               to="2007-10-12T23:59:00+02:00"/>
    <conds>
      <cond type="ONVALUE">
        <constraint param="activity.status" op="CONT" value="with_friends"/>
      </cond>
    </conds>
  </ctxQuery>
</contextQL>
```

The answer is a ContextML message, as for a standard ContextML getContext request.

18.2.1.2 *SUBSCRIBE ONVALUE*

The SUBSCRIBE ONVALUE allows to subscribe to context data notification if a context parameter has a specific value. Condition could be related also to context scope not included in the returned data set.

```
<contextQL xmlns="http://ContextQL/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ContextQL/1.0
http://cark3.cselt.it/schemas/ContextQL-1.0.xsd">
  <ctxQuery>
    <action type="SUBSCRIBE"/>
    <entity>username|cristinaF</entity>
    <scope>position</scope>
    <expire>36000</expire>
    <conds>
      <cond type="ONVALUE">
        <logical op="AND">
          <constraint param="position.locMode" op="EQ" value="GPS"/>
          <constraint param="cell.cgi" op="NSTW" value="222-1-"/>
        </logical>
      </cond>
    </conds>
  </ctxQuery>
</contextQL>
```

The response will contain the expiration time (sec) and the subscription Id, useful to renew or delete subscription:

```
<contextML xmlns="http://ContextML/1.3"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ContextML/1.3
http://cark3.cselt.it/schemas/ContextML-1.3.xsd ">
  <ctxResp>
    <contextProvider id="CB" v="1.2"/>
    <method>subscribe</method>
    <resp code="200" status="OK"/>
    <expire>7200</expire>
    <subId>5672</subId>
  </ctxResp>
</contextML>
```

18.2.1.3 *SUBSCRIBE ONCLOCK*

The SUBSCRIBE ONCLOCK allows to subscribe to context data notification which are sent at a specific time interval.

```
<contextQL xmlns="http://ContextQL/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ContextQL/1.0
http://cark3.cselt.it/schemas/ContextQL-1.0.xsd">
  <ctxQuery>
    <action type="SUBSCRIBE"/>
    <entity>imei|358361005978493</entity>
    <scope>deviceStatus</scope>
    <expire>3600</expire>
    <conds>
      <cond type="ONCLOCK">
        <constraint value="300"/>
      </cond>
    </conds>
  </ctxQuery>
</contextQL>
```

18.2.1.4 *SUBSCRIBE ONCLOCK/ONCHANGE*

Also subscription on mixed conditions could be specified in CQL (not supported in current release):

```
<contextQL xmlns="http://ContextQL/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ContextQL/1.0
http://cark3.cselt.it/schemas/ContextQL-1.0.xsd">
  <ctxQuery>
    <action type="SUBSCRIBE"/>
    <entity>imei|358361005978493</entity>
    <scope>cell</scope>
    <expire>3600</expire>
    <conds>
      <condOp op="OR">
        <cond type="ONCLOCK">
          <constraint value="300"/>
        </cond>
        <cond type="ONCHANGE">
          <constraint param="cell.cgi"/>
        </cond>
      </condOp>
    </conds>
  </ctxQuery>
</contextQL>
```

18.2.1.5 *SELECT COUNT*

The SELECT COUNT returns the number of entities which have context data in cache verifying the required condition:

```
<contextQL xmlns="http://ContextQL/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ContextQL/1.0
http://cark3.cselt.it/schemas/ContextQL-1.0.xsd">
  <ctxQuery>
    <action type="SELECT" option="COUNT"/>
    <entity>username|*</entity>
    <timerange from="2007-10-24T12:00:00+02:00"
               to="2007-10-24T12:00:00+02:00"/>
    <conds>
      <cond type="ONVALUE">
        <constraint param="civilAddress.city" op="EQ" value="Torino"/>
      </cond>
    </conds>
  </ctxQuery>
</contextQL>
```

The response is as follows:

```
<contextML xmlns="http://ContextML/1.3"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ContextML/1.3
http://cark3.cselt.it/schemas/ContextML-1.3.xsd ">
  <ctxResp>
    <contextProvider id="CB" v="1.2"/>
    <method>select</method>
    <resp code="200" status="OK"/>
    <dataPart>
      <count>7</count>
    </dataPart>
  </ctxResp>
</contextML>
```

18.2.1.6 *SELECT LIST*

The SELECT LIST returns the list of entities which have context data in cache verifying the required condition:

```
<contextQL xmlns="http://ContextQL/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ContextQL/1.0
http://cark3.cselt.it/schemas/ContextQL-1.0.xsd">
  <ctxQuery>
    <action type="SELECT" option="LIST"/>
    <entity>username|*</entity>
    <conds>
      <cond type="ONVALUE">
        <logical op="AND">
          <constraint param="activity.status" op="CONT" value="working"/>
          <constraint param="activity.status" op="CONT" value="own_office"/>
        </logical>
      </cond>
    </conds>
  </ctxQuery>
</contextQL>
```

The response is as follows:

```
<contextML xmlns="http://ContextML/1.3"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ContextML/1.3
http://cark3.cselt.it/schemas/ContextML-1.3.xsd">
  <ctxResp>
    <contextProvider id="CB" v="1.2"/>
    <method>select</method>
    <resp code="200" status="OK"/>
    <dataPart>
      <parA n="entities">
        <par n="entity">cristinaF</par>
        <par n="entity">sergio</par>
        <par n="entity">mvalla</par>
      </parA>
    </dataPart>
  </ctxResp>
</contextML>
```

19 FIWARE OpenSpecification Data QueryBroker

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

Name	FIWARE.OpenSpecification.Data.QueryBroker
Chapter	Data/Context Management,
Catalogue-Link to Implementation	<Query Broker>
Owner	Siemens AG, Thomas Riegel

19.1 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.

19.1.1 Copyright

- Copyright © 2012 by Siemens AG

19.1.2 Legal Notice

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

19.2 Overview

19.2.1 Introduction to the Media-enhanced Query Broker GE

Multimedia data is produced at an immense rate. By investigating solutions and approaches for storing and archiving the produced data, one rapidly ends up in a highly heterogeneous environment of data stores. Usually, the involved domains feature individual sets of metadata formats for describing content, technical or structural information of multimedia data [Stegmaier 09a]. Furthermore, depending on the management and retrieval requirements, these data sets are accessible in different systems supporting a multiple set of retrieval models and query languages. By summing up all these obstacles, easy and efficient access and retrieval across those system borders is a very cumbersome task [Smith 08]. Standards are one way to introduce interoperability among different peers. Recent developments and achievements in the domain of multimedia retrieval concentrated on the establishment of a multimedia query language (MPEG) [Döller 08a], standardized image retrieval (JPEG) and the heterogeneity problem between metadata formats (JPEG) [Döller 10]. Another approach

for interoperable media retrieval is the introduction of a mediator or middleware system abstracting the communication: a Media-enhanced Query Broker. Acting as middleware and mediator between multimedia clients and retrieval systems, collaboration can be remarkably improved. A Media-enhanced Query Broker accepts complex multi-part and multimodal queries from one or more clients and maps/distributes those to multiple connected Multimedia Retrieval Systems (MMRS). Consequently, implementation complexity is reduced at the client side as only one communication partner needs to be addressed. Result aggregation and query distribution is also accommodated, further easing client development. However, the actual retrieval process of the multimedia data is performed inside the connected data stores.

19.2.2 Target usage

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

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

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

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

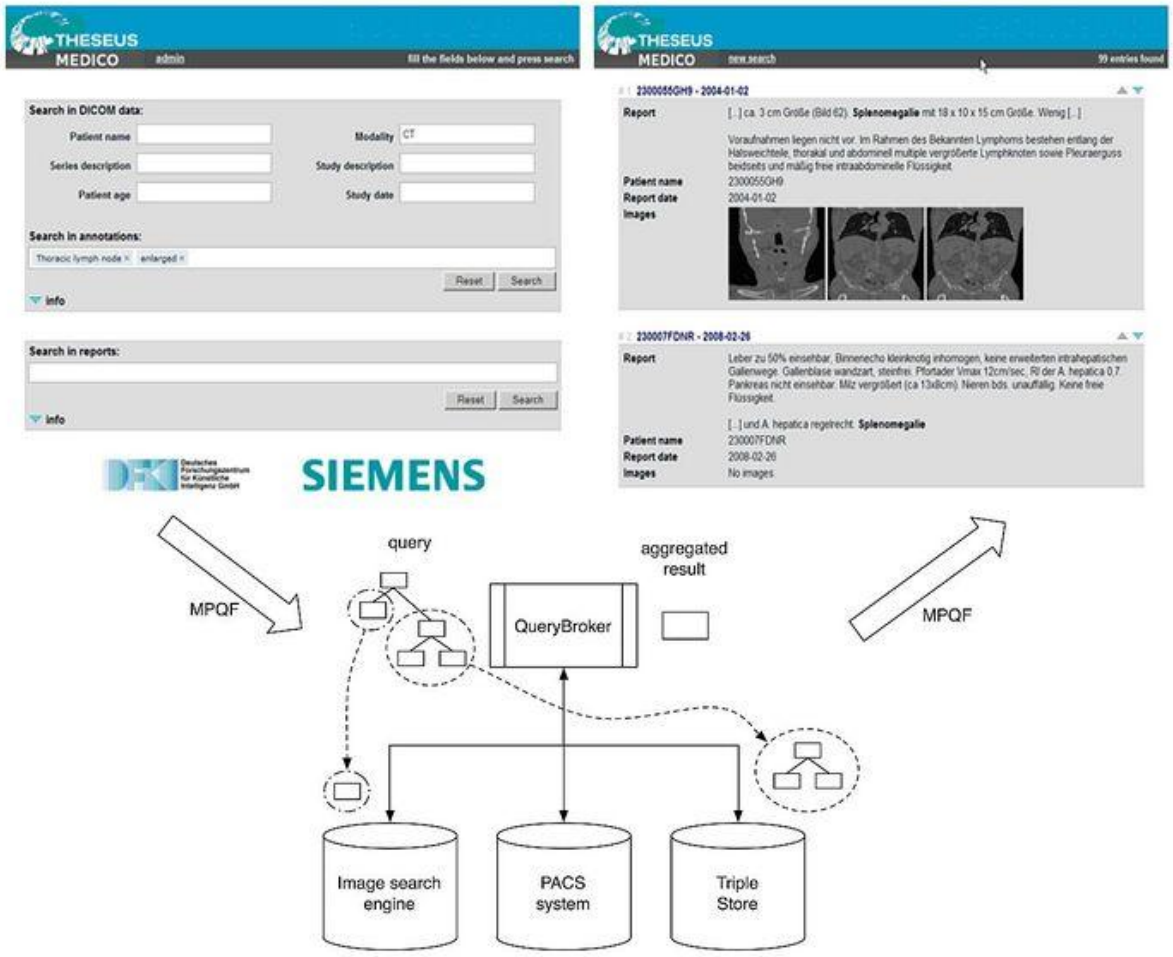
The question about how non-relational or "NoSQL" databases, which are becoming an increasingly important part of the database landscape, can be integrated, is one of the open points to be addressed during the FI-WARE project.

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

19.2.3 Example Scenario

The already identified issues of heterogeneity can be also found in the current diagnostic process at hospitals. The workflow of a medical diagnosis is mainly based on reviewing and

comparing images coming from multiple time points and modalities in order to monitor disease progression over a certain period of time. For ambiguous cases the radiologist deeply relies on reference literature or second opinion. Beside textual data stored in appraisals, a vast amount of images (e.g., CT scans) is stored in Picturing Archive and Communications Systems (PACS), which could be reused for decision support. Unfortunately efficient access to this information is not available due to weak search capabilities.
The mission of the MEDICO application scenario is to establish an intelligent and scalable search engine for the medical domain by combining medical image processing and semantically rich image annotation vocabularies.



Search infrastructure: end-to-end workflow in MEDICO

The figure above sketches an end-to-end workflow inside the MEDICO system. It provides the user with an easy-to-use web-based form to describe the desired search query. Currently, this user interface utilizes a semantically rich data set composed of DICOM tags, image annotations, text annotations and gray-value based (3D) CT images. This leads to a heterogeneous multimedia retrieval environment with multiple query languages: DICOM tags as well as the raw image data are stored in a PACS, annotations describing images, doctor´s letter as well as laboratory exminations are saved in a triple store. Finally, a similarity search can be conducted by the use of an image search engine, which operates on top of extracted image features. Apparently, all these retrieval services are using their own query languages

for retrieval (e.g., SPARQL) as well as the actual data representation for annotation storage (e.g., RDF/OWL). To fulfill a sophisticated semantic search, the present interoperability issues have to be solved. Furthermore, it is essential to enable federated search functionalities in this environment. These requirements have been taken into account in the design and implementation of the QueryBroker. An overview of the architecture can be found in [Stegmaier 10] and [Stegmaier 09b].

19.3 Basic Concepts

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

19.3.1 Design Principles

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

- Query language abstraction:

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

- Multiple retrieval paradigms:

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

- Metadata format interoperability:

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

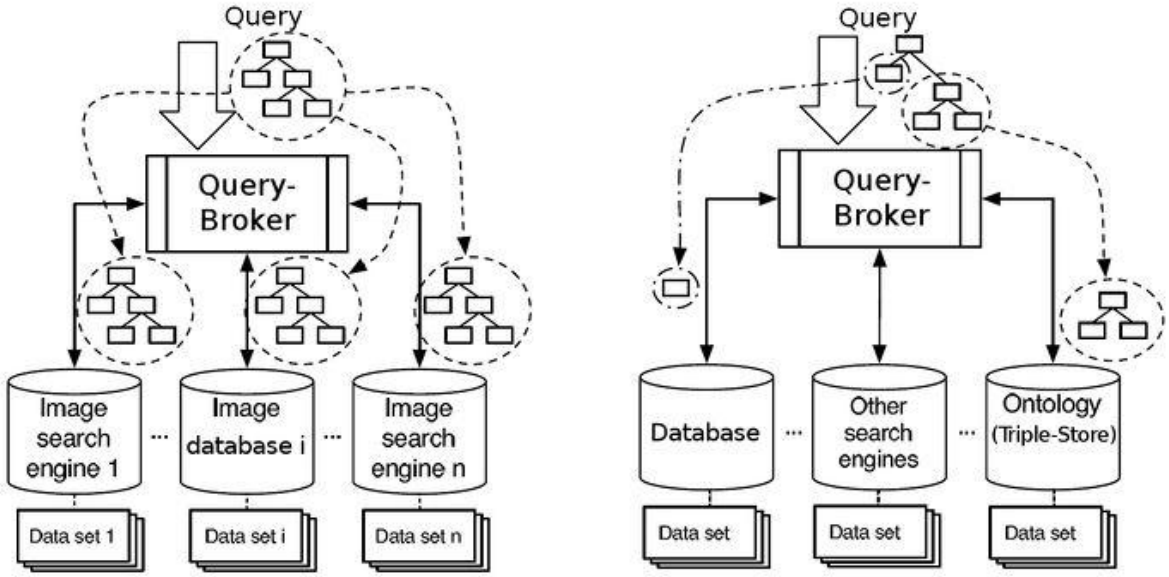
- Modular architectural design:

  A modular architectural design should always be striven for in software development. The central aspects in these topics are convertibility, extensibility and reusability. These ensure that the components are loosely coupled in the overall system

supporting an easy extension of the provided functionality of components, or even the replacement of these by new implementations.

19.3.2 Query Processing Strategies

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



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

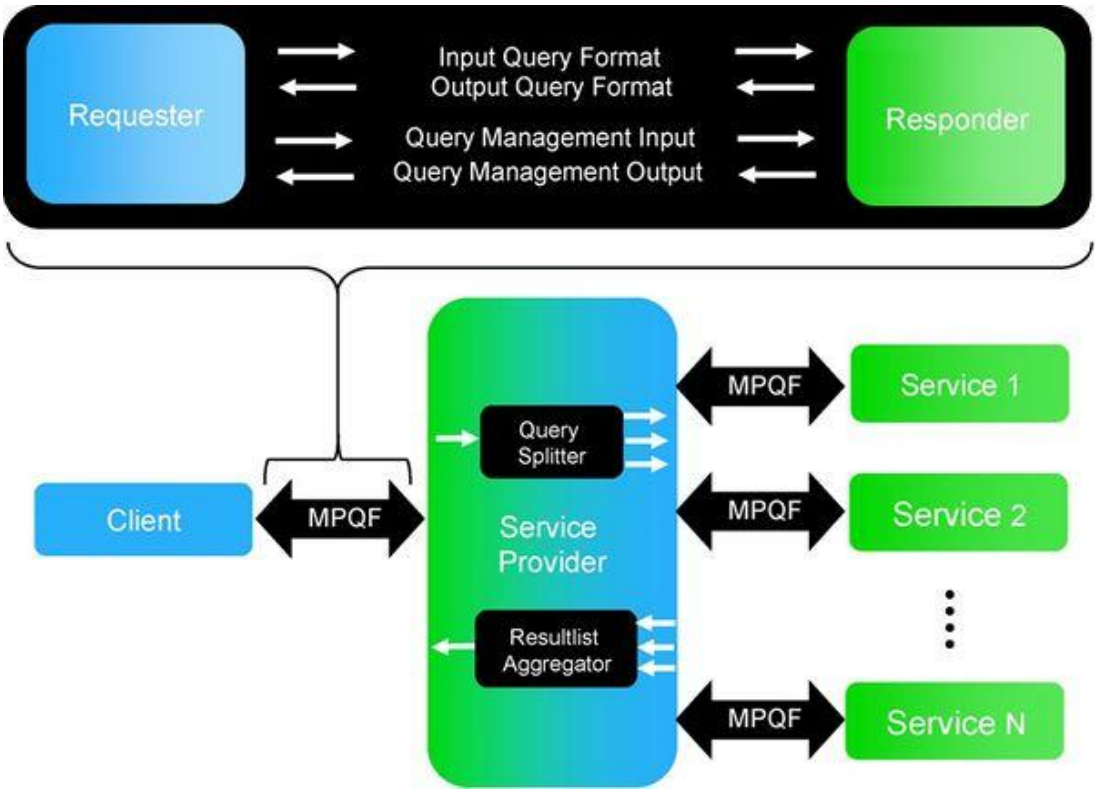
The first paradigm deals with registered and participating retrieval systems that are able to process the whole query locally, see the left side (a) of the figure above. In this sense, those heterogeneous systems may provide their local metadata format and a local / autonomous data set. A query transmitted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

The second paradigm deals with registered and participating retrieval systems that allow distributed processing on the basis of a global data set as illustrated in the right side (b) of the figure above. The involved heterogeneous systems may depend on different data representation (e.g., ontology based semantic annotations and XML-based feature values) and query interfaces (e.g., SPARQL and XQuery) but describe a common (linked) global data set. In this context, a query transmitted to the Media-enhanced Query Broker needs to

be evaluated and optimized resulting in a specific query execution plan. Segments of the query are forwarded to the respective engines to be executed in parallel. Subsequently, the result aggregation has to deal with the correct consolidation and (if required) format conversion of the partial result sets. In this context, the Media-enhanced Query Broker behaves like a federated Database Management System.

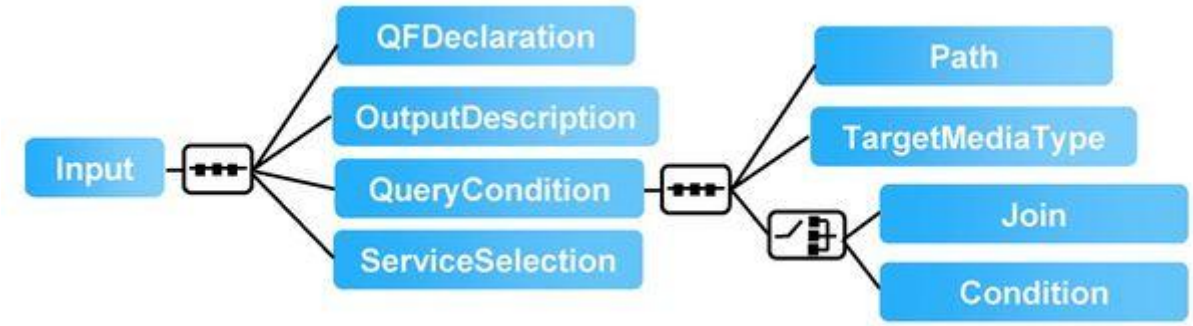
19.3.3 MPEG Query Format (MPQF)

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



Possible scenario for the use of MPQF

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



Structure of the Input Query Format

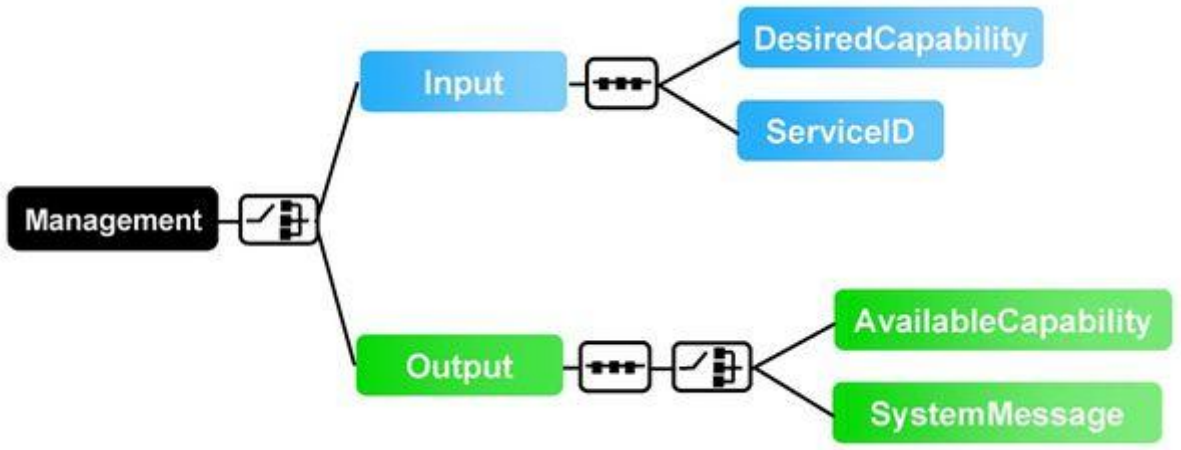
In detail, the IQF (see the figure above) can be composed of three different parts. The first is a declaration part pointing to resources (e.g., image file or its metadata description, etc.) that are used within the query condition or output description part. The output description part allows, by using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

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

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

Available MPQF query types

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



Structure of the Query Management Tools

The QMT of MPQF cope with the task of searching for and choosing desired multimedia services for retrieval. This includes service discovery, querying for service capabilities and service capability descriptions. The figure above depicts the element hierarchy of the

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

19.3.4 Federated Query Evaluation Workflow

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

- Query analysis

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

- Query segmentation

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

- Generation of a query execution plan

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

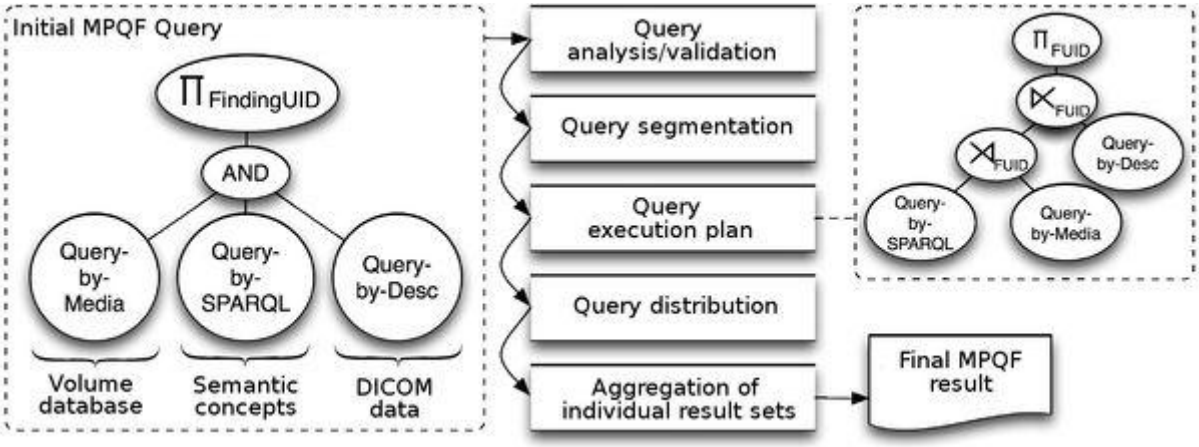
- Distribution of query

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

- Consolidation of partial results

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

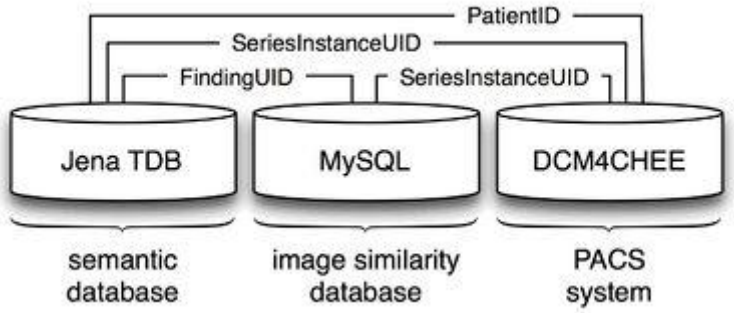
The described workflow of the federated query processing can best be illustrated using the example scenario, as depicted in the figure below.



Central steps of the query execution plan

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

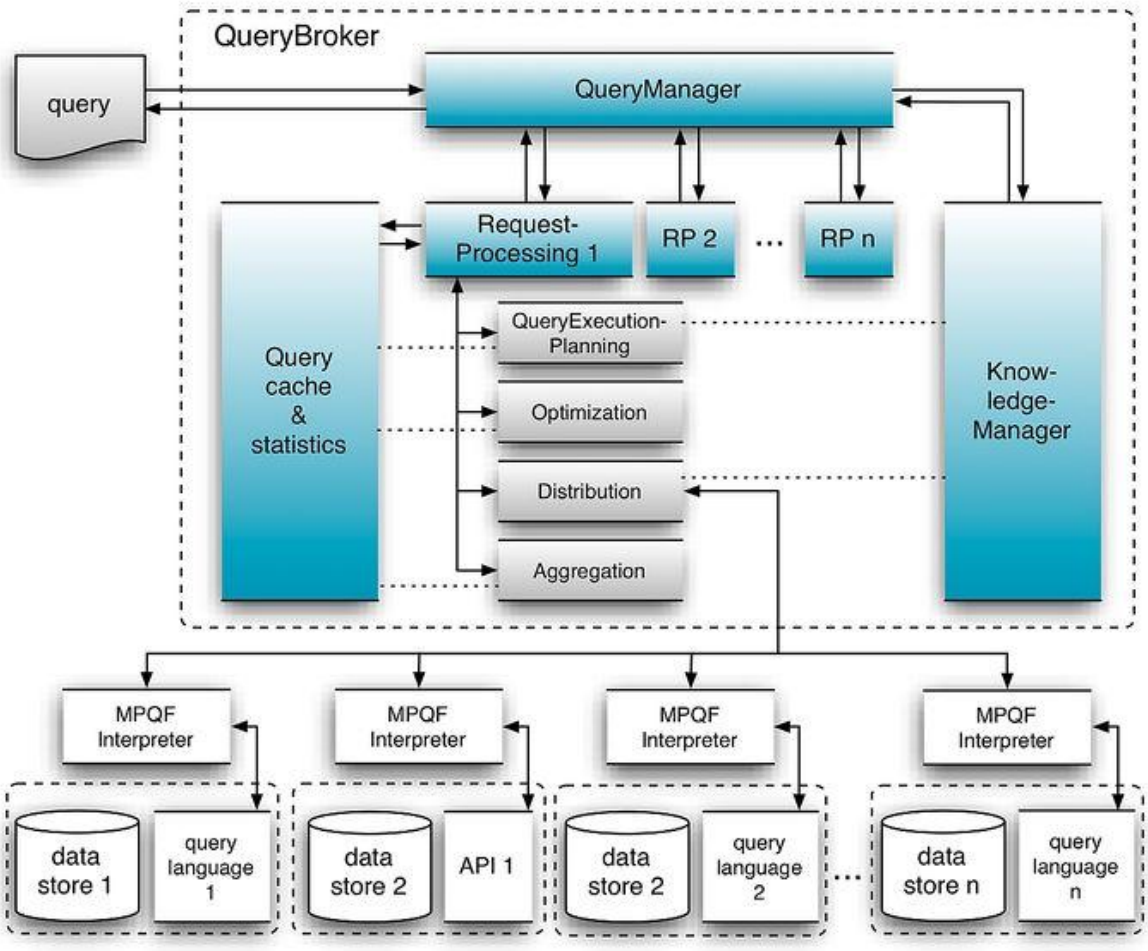
This principle is called semantic links, for a definition and examples see Creating a Semantic Link. The following figure depicts for the example scenario the diverse data sources forming a common (semantically linked) global data set.



Semantic Link between knowledge bases

19.4 QueryBroker Architecture

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



Architecture of the QueryBroker

- **QueryManager:**

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

- **KnowledgeManager:**

  The main functionalities of the KnowledgeManager are the (de-) registration of data stores with their capability descriptions and the service discovery as an input for the

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

- **RequestProcessing:**

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

- **Query cache and statistics:**

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

- **MPQF interpreter:**

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

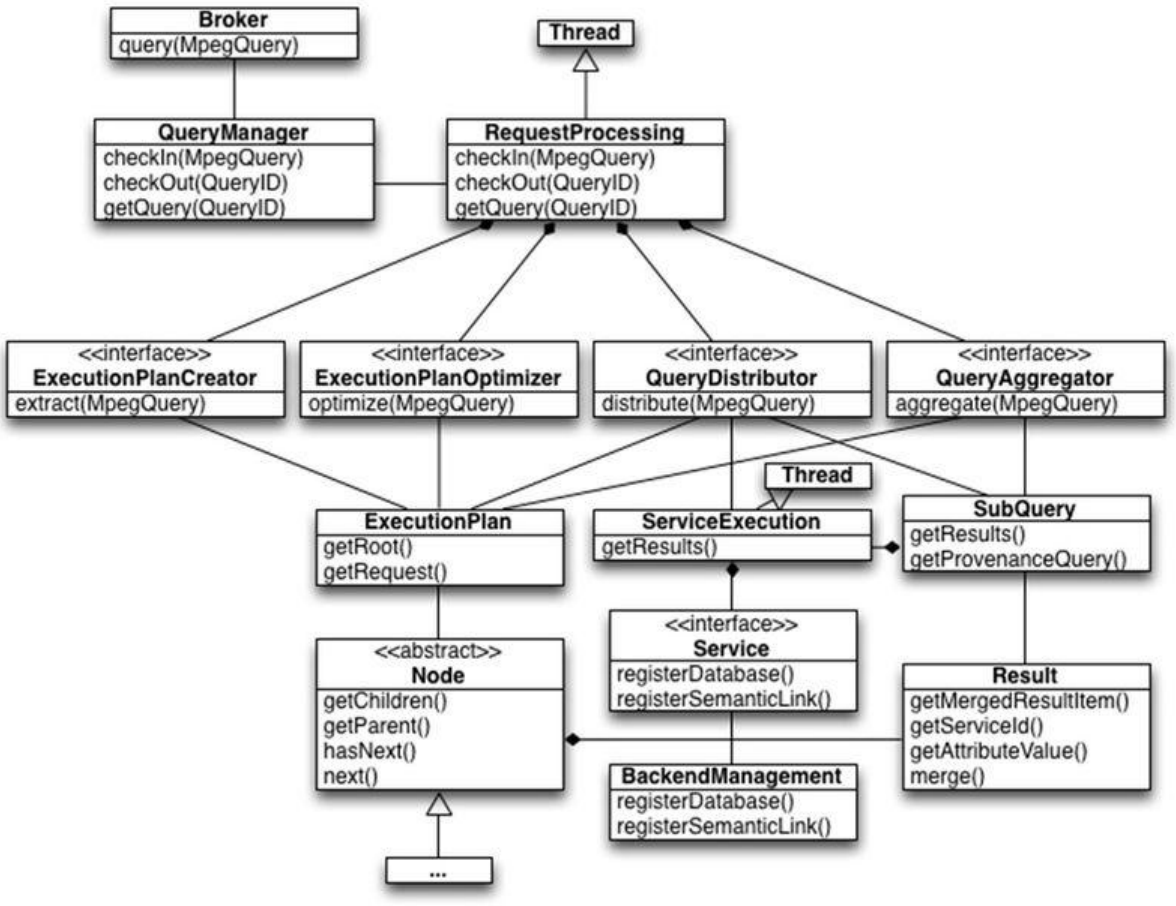
19.5 Main Interactions

19.5.1 Modules and Interfaces

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

19.5.2 Architecture

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



QueryBroker software architecture

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

    - **Service** interface has to be implemented by any service endpoint. A service endpoint connects a database or another dataset to the multimedia query framework.

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

19.5.3 Backend Functionality

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

19.5.3.1 *(De-)Register a Service*

Executable by BackendManagement.getInstance().registerDatabase(mqt); Service endpoints are able to execute sub trees of the query execution plan. At the moment only single leaves are supported as sub trees. These can be Query-By-Media or Query-by-Description. To register a service endpoint, which has to implement the Service Interface

(de.uop.dimis.air.backend.Service), a valid MPQF message needs to be formulated like the following:

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

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

Java example:

```
// get the register query xml file as stream

InputStream stream = ...

// unmarshal the xml file

Unmarshaller                              u                                  =
NamespaceHelper.getInstance().getJAXBContextMpqfJPSearch().createUnm
arshaller();

MpegQueryType mpqfQuery = (MpegQueryType) u.unmarshal(stream);

// register the database

BackendManagement.getInstance().registerDatabase(mpqfQuery);
```

19.5.3.2 *Creating a Semantic Link*

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

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

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

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

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

Java example:

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

19.5.4 Frontend Functionalities

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

Querying

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

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

Synchronous Query

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

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

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

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

Java example:

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

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

Java example:

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

```
List<ResultItemType> results =
response.getQuery().getOutput().getResultItem();
```

Asynchronous Query

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

Java Example:

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

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

Java Example:

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

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

Complex Query Example

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

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

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

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

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

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

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

Query Execution Tree Evaluation

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

19.5.4.1 *Code Example*

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

Java example:

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

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

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

        // create a result container for the computed results
        ObjectFactory mpqfObjFac = new ObjectFactory();
        MpegQueryType result = mpqfObjFac.createMpegQueryType();
        Query qry = mpqfObjFac.createMpegQueryTypeQuery();
        result.setQuery(qry);
```

```
        OutputQueryType oqt = mpqfObjFac.createOutputQueryType();
        qry.setOutput(oqt);
        List<ResultItemType> resultItems = oqt.getResultItem();

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

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

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

```
</mpqf:MpegQuery>
```

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

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

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

Java example:

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

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

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

ExampleService.class.getResourceAsStream("SemanticLinks.xml");
// unmarshal the xml file
Unmarshaller u =
NamespaceHelper.getInstance().getJAXBContextsSemanticLinks().createU
nmarshaller();
KeyMatchesType link = (KeyMatchesType) ((JAXBElement<?>)
u.unmarshal(stream)).getValue();
```

```
// register the semantic link
BackendManagement.getInstance().registerSemanticLink(link);
// the queryBroker is now ready for queries
```

19.6 References

[DICOM]	Digital Imaging and Communications in Medicine; The DICOM Standard
[Döller 08a]	M. Döller, R. Tous, M. Gruhne, K. Yoon, M. Sano, and I. S. Burnett, “The MPEG Query Format: On the Way to Unify the Access to Multimedia Retrieval Systems,” IEEE Multimedia, vol. 15, no. 4, pp. 82–95, 2008.
[Döller 08b]	Mario Döller, Kerstin Bauer, Harald Kosch and Matthias Gruhne. Standardized Multimedia Retrieval based on Web Service technologies and the MPEG Query Format. Journal of Digital Information, 6(4):315-331,2008.
[Döller 10]	M. Döller, F. Stegmaier, H. Kosch, R. Tous, and J. Delgado, “Standardized Interoperable Image Retrieval,” in Proceedings of the ACM Symposium on Applied Computing, Track on Advances in Spatial and Image-based Information Systems, (Sierre, Switzerland), pp. 881–887, 2010.
[DublinCore]	Dublin Core Metadata Initiative. Dublin Core metadata element set – version 1.1: Reference description. http://dublincore.org/documents/dces/, 2008.
[Gruhne 08]	Matthias Gruhne, Peter Dunker, Ruben Tous and Mario Döller. Distributed Cross-Modal Search with the MPEG Query Format. In 9th International Workshop on Image Analysis for Multimedia Interactive Services, pages 211-224, Klagenfurt, Austria, May 2008. IEEE Computer Society.
[JAXB]	Java Architecture for XML Binding (JAXB), Metro Projekt, http://jaxb.dev.java.net/
[Matinez 02]	J. M. Matinez, R. Koenen and F. Pereira. MPEG-7. IEEE Multimedia, 9(2):78-87, April-June 2002.
[MAVEN]	Apache Maven, Apache Software Foundation, http://maven.apache.org/
[MPEG-7]	ISO/IEC 15938-1:2002 - Information technology -- Multimedia content description interface -- http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=46427
[Smith 08]	J. R. Smith, “The Search for Interoperability,” IEEE Multimedia, vol. 15, no. 3, pp. 84–87, 2008.
[Spring]	The Spring Framework, SpringSource, 2012, http://www.springsource.org/
[Stegmaier 09a]	F. Stegmaier, W. Bailer, T. Bürger, M. Döller, M. Höffernig, W. Lee, V. Malaisé, C. Poppe, R. Troncy, H. Kosch, and R. V. de Walle, “How to Align Media Metadata Schemas? Design and Implementation of the Media Ontology,” in Proceedings of the 10th International Workshop of the Multimedia Metadata Community on Se- mantic Multimedia Database

	Technologies in conjunction with SAMT, vol. 539, (Graz, Austria), pp. 56–69, December 2009.
[Stegmaier 09b]	Florian Stegmaier, Udo Gröbner, Mario Döller. “Specification of the Query Format for medium complexity problems (V1.1)”, Deliverable CTC 2.5.15 of Work-Package 2 (“Video, Audio, Metadata, Platforms”) of THESEUS Basic Technologies, 2009.
[Stegmaier 10]	Florian Stegmaier, Mario Döller, Harald Kosch, Andreas Hutter and Thomas Riegel. “AIR: Architecture for Interoperable Retrieval on distributed and heterogeneous Multimedia Repositories”. 11th International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS2010), Desenzano del Garda, Italy, p 1-4.
[XPath]	XML Path Language (XPath), 2.0 (Second Edition, W3C Recommendation, 14. December 2010, http://www.w3.org/TR/xpath20/

19.7 Detail Specifications

The 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 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.

19.7.1 Open API Specifications

- Query Broker Open RESTful API Specification (PRELIMINARY)

19.8 Re-utilised Technologies/Specifications

At its core, the QueryBroker utilizes the MPEG Query Format (MPQF) [ISO/IEC 15938-12:2008] as common internal representation for input and output query description and managing the backend search services. A comprehensive overview can be found in the papers [1,2].

The GE itself is implemented in Java.

Standards

[ISO/IEC 15938-12:2008] "Information Technology - Multimedia Content Description Interface - Part 12: Query Format". Editors: Kyoungro Yoon, Mario Doeller, Matthias Gruhne, Ruben Tous, Masanori Sano, Miran Choi, Tae-Beom Lim, Jongseol James Lee, Hee-Cheol Seo.

[ISO/IEC 15938-12:2008/Cor.1:2009] "Information Technology - Multimedia Content Description Interface - Part 12: Query Format, TECHNICAL CORRIGENDUM 1". Editors: Kyoungro Yoon, Mario Doeller.

Take a look at the latest version of the MPEG Query Format XML Schema at https://svn-itec.uni-klu.ac.at/repos/MPEGSchema/trunk/mpeg7/mpqf.xsd

References

[1] Mario Döller, Ruben Tous, Matthias Gruhne, Kyoungro Yoon, Masanori Sano, and Ian S Burnett, “The MPEG Query Format: On the way to unify the access to Multimedia Retrieval Systems”, IEEE Multimedia, vol. 15, no. 4, pp. 82–95, 2008.

[2] Ruben Tous and Jaime Delgado (2008). *Semantic-driven multimedia retrieval with the MPEG Query Format*. 3rd International Conference on Semantic and Digital Media Technologies SAMT 3 - 5 Dec 2008, Koblenz, Germany. Lecture Notes in Computer Science. ISSN 0302-9743. Volume 5392/2008. ISBN 978-3-540-92234-6. Pag. 149-163.

19.9 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.

- **Data** refers to information that is produced, generated, collected or observed that may be relevant for processing, carrying out further analysis and knowledge extraction. Data in FI-WARE has associated a **data type** and a**value**. FI-WARE will support a set of built-in **basic data types** similar to those existing in most programming languages. Values linked to basic data types supported in FI-WARE are referred as **basic data values**. As an example, basic data values like ‘2’, ‘7’ or ‘365’ belong to the integer basic data type.
- A **data element** refers to data whose value is defined as consisting of a sequence of one or more <name, type, value> triplets referred as **data element attributes**, where the type and value of each attribute is either mapped to a basic data type and a basic data value or mapped to the data type and value of another data element.
- **Context** in FI-WARE is represented through **context elements**. A context element extends the concept of **data element** by associating an EntityId and EntityType to it, uniquely identifying the entity (which in turn may map to a group of entities) in the FI-WARE system to which the context element information refers. In addition, there may be some attributes as well as meta-data associated to attributes that we may define as mandatory for context elements as compared to data elements. Context elements are typically created containing the value of attributes characterizing a given entity at a given moment. As an example, a context element may contain values of some of the attributes “last measured temperature”, “square meters” and “wall color” associated to a room in a building. Note that there might be many different context elements referring to the same entity in a system, each containing the value of a different set of attributes. This allows that different applications handle different context elements for the same entity, each containing only those attributes of that entity relevant to the corresponding application. It will also allow representing updates on set of attributes linked to a given entity: each of these updates can actually take the form of a context element and contain only the value of those attributes that have

changed.

- An **event** is an occurrence within a particular system or domain; it is something that has happened, or is contemplated as having happened in that domain. Events typically lead to creation of some data or context element describing or representing the events, thus allowing them to processed. As an example, a sensor device may be measuring the temperature and pressure of a given boiler, sending a context element every five minutes associated to that entity (the boiler) that includes the value of these to attributes (temperature and pressure). The creation and sending of the context element is an event, i.e., what has occurred. Since the data/context elements that are generated linked to an event are the way events get visible in a computing system, it is common to refer to these data/context elements simply as "events".
- A **data event** refers to an event leading to creation of a data element.
- A **context event** refers to an event leading to creation of a context element.
- An **event object** is used to mean a programming entity that represents an event in a computing system [EPIA] like event-aware GEs. Event objects allow to perform operations on event, also known as **event processing**. Event objects are defined as a data element (or a context element) representing an event to which a number of standard event object properties (similar to a header) are associated internally. These standard event object properties support certain event processing functions.

20 Query Broker Open RESTful API Specification (PRELIMINARY)

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

20.1 Introduction to the REST-Interface of the QueryBroker

Please check the FI-WARE Open Specifications Legal Notice to understand the rights to use FI-WARE Open Specifications.

20.1.1 QueryBroker REST-API Core

The QueryBroker REST-API is a RESTful, resource-oriented API accessed via HTTP that uses XML-based representations for information interchange. It offers a convenient way to manage a QueryBroker instance and to submit complex multi-part and multimodal queries to multiple connected MMRS by sending appropriate MPQF expressions.

20.1.2 Intended Audience

This specification is intended for both software developers and Cloud Providers. For the former, this document provides a specification of how to interoperate with Cloud Platforms that implements the QueryBroker REST API. For the latter, this specification indicates the interface to be provided in order for clients to interoperate with Cloud Platform to provide the described functionalities. To use this information, the reader should firstly have a general understanding of the Media-enhanced Query Broker GE. You should also be familiar with:

- RESTful web services
- HTTP/1.1
- MPQF (MPEG Query Format,[Tous 2008]).

20.1.3 API Change History

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

Revision Date	Changes Summary
Apr 24, 2012	• initial version
July 4, 2012	• version 0.5
Oct 4, 2012	• version 0.6
...	• ...

20.1.4 How to Read This Document

In the whole document it is taken the assumption that reader is familiarized with REST architecture style. Along the document, some special notations are applied to differentiate some special words or concepts. The following list summarizes these special notations.

- A bold, mono-spaced font is used to represent code or logical entities, e.g., HTTP method (GET, PUT, POST, DELETE).
- An italic font is used to represent document titles or some other kind of special text, e.g., URI.
- The variables are represented between brackets, e.g. {id} and in italic font. When the reader find it, can change it by any value.

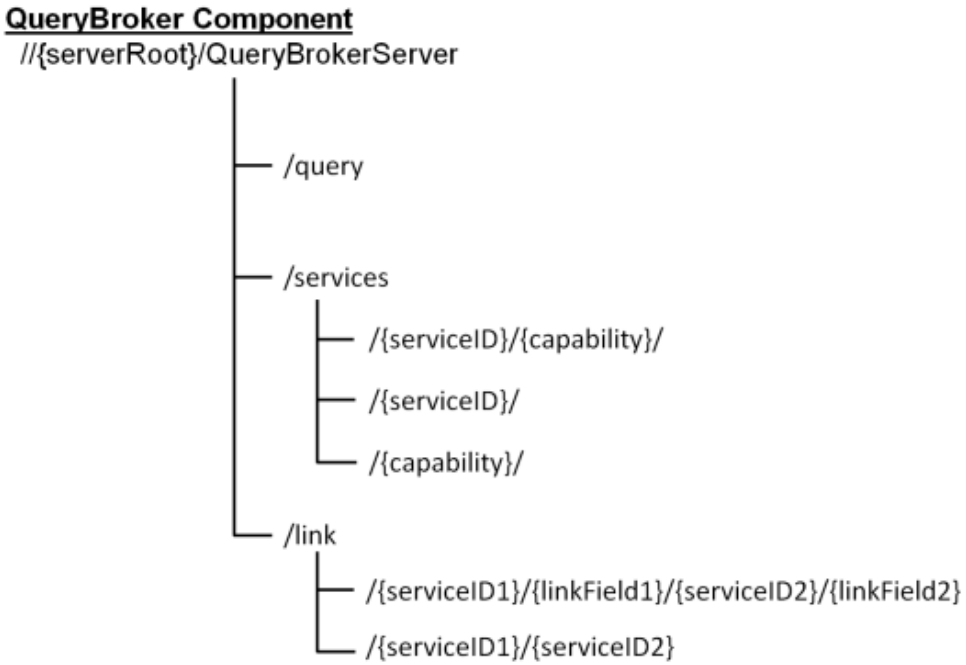
For a description of some terms used along this document, see High Level Description of Query Broker GE.

20.1.5 Additional Resources

You can download the most current version of this document from the FIWARE API specification website at the Summary of FI-WARE Open Specifications. For more details about the Media-enhanced Query Broker GE that this API is based upon, please refer to "Architecture Description of Media-enhanced Query Broker GE". Related documents, including an Architectural Description, are available at the same site."

20.2 General QueryBroker REST API Information

20.2.1 Resources Summary



20.2.2 Authentication

Authentication is NOT supported in Version 1 - This feature is foreseen to be incorporated in the future.

At that time each HTTP request against the *QueryBroker* will require the inclusion of specific authentication credentials. The specific implementation of this API may support multiple authentication schemes (OAuth, Basic Auth, Token) and will be determined by the specific provider that implements the GE. Please contact them to determine the best way to authenticate against this API. Remember that some authentication schemes may require that the API operate using SSL over HTTP (HTTPS).

20.2.3 Representation Format

The QueryBroker REST API supports XML based representation formats, namely MPQF and XPath. The request format is specified using the Content-Type header and is required for operations that have a request body. The response format is in the initial version MPQF. (Later it may also be specified in requests using either the Accept header. Note that it is possible for a response to be serialized using a format different from the request.

20.2.4 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.

20.2.5 Resource Identification

The resource identification used by the API in order to identify unambiguously the resource will be provided over time. For HTTP transport, this is made using the mechanisms described by HTTP protocol specification as defined by IETF RFC-2616.

20.2.6 Links and References

None

20.2.7 Paginated Collections

The MPQF allows the specification of limits on the number of elements to return; if desired it is recommended to use this feature to control/reduce the load on the service(s) as it provides more sophisticated configuration options.

20.2.8 Limits

Please have in mind, that in the current version the processing is done in-memory, i.e. many very large intermediate results delivered simultaneously be the registered data repositories may exhausts the available RAM (min. 1 GB, but 4 GB preferred) causing an fault.

20.2.9 Versions

Querying the version is NOT supported in Version 1 - This feature is foreseen be incorporated in the future.

20.2.10 Faults

Please find below a list of possible fault elements and error codes

Fault Element	Associated Error Codes	Description	Expected in All Requests?
POST	400 (“Bad Request”)	The document in the entity-body, if any, contains an error message. Hopefully the client can understand the error message and use it to fix the problem.	[YES]
POST	404 (“Not Found”)	The requested URI doesn’t map to any resource. The server has no clue what the client is asking for.	[YES]
POST	500 (“Internal Server Error”)	There’s a problem on the server side. The document in the entity-body, if any, is an error message. The error message probably won’t do much good, since the client can’t fix the server problem.	[YES]

20.3 API Operations

The QueryBroker is implemented as a middleware to establish unified retrieval in distributed and heterogeneous environments with extension functionalities to integrate multimedia specific retrieval paradigms in the overall query execution plan, e.g., multimedia fusion technique. To ensure interoperability between the query applications and the registered database services, the QueryBroker uses as internal query representation format the MPEG Query Format (MPQF). MPQF is an XML-based (multimedia) query language which defines the format of queries and replies to be interchanged between clients and servers in a (multimedia) information search and retrieval environment.

The normative parts of the MPEG Query Format define three main components:

- The Input Query Format provides means for describing query requests from a client to a information retrieval system.
- The Output Query Format specifies a message container for the connected retrieval systems responses and finally
- The Query Management Tools provide means for functionalities such as service discovery, service aggregation and service capability description (e.g.,which query types or formats are supported).

Therefore MPQF can be and is used for managing all essential tasks submitting complex multi-part and multimodal queries to multiple connected data resources, namely:

- (de-)register a retrieval system/service,
- creating a semantic link in case of an included join operation, and
- the actual query.

As appropriate MPQF expressions can be lengthy this is done by POST allowing the data to be transmitted in the body of the http request.

Important: A serviceID is the fully qualified class name (e.g., de.uop.dimis.test.Service) of the implemented service. Hence, all services which will be registered at the QueryBroker have to be in the classpath of the QueryBroker-Server WAR-file.

20.3.1 QueryBroker operations

20.3.1.1 ***Submit MPQF query***

Verb	URI	Description
POST	//{serverRoot}/QueryBrokerServer/query	Submit a valid MPQF query (The request body must contain a valid MPQF query in XML serialization.)

Request example:

```
POST //localhost/QueryBrokerServer/query HTTP/1.1
Content-Type: multipart/form-data
Accept: multipart/form-data
MIME-Version: 1.0
Host: localhost:8080
Content-Type: multipart/form-data
Content-Length: 425

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

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

Response example:

```
HTTP/1.1 200 OK
```

20.3.1.2 *(De-)Register Database*

Verb	URI	Description
POST	/services/{serviceID}/{capability}/	Registers the service at the QueryBroker endpoint e.g., /services/de.uop.dimis.FlickrService/QueryByMedia/ (remark: a serviceID is the full qualified class name (e.g., de.uop.dimis.test.Service) of the implemented service.)
DELETE	/services/{serviceID}/	Deregister the service at the QueryBroker e.g., /services/de.uop.dimis.FlickrService/
GET	/services/{capability}/	Service discovery for given capability. Returns a semicolon separated list of all registered services which will handle the given capability. e.g., /services/QueryByMedia/

Request example:

```
POST
//localhost/QueryBrokerServer/services/de.uop.dimis.FlickrService/Qu
eryByMedia HTTP/1.1
Content-Type: multipart/form-data
Accept: multipart/form-data
MIME-Version: 1.0
Host: localhost:8080
Content-Type: multipart/form-data
Content-Length: 0
```

Response example:

```
HTTP/1.1 200 OK
```

20.3.1.3 *Creating a Semantic Link*

Verb	URI	Description
POST	//{serverRoot}/QueryBrokerServer/link/{serviceID1}/{linkField1}/{serviceID2}/{linkField2}	Registers a semantic link between two registered endpoints.
GET	//{serverRoot}/QueryBrokerServer/link/{serviceID1}/{serviceID2}	Returns information about the registered semantic link between the given services.

Request example:

```
POST
//localhost/QueryBrokerServer/link/de.uop.dimis.FlickrService/identi
fier/de.uop.dimis.FBService/decription/ HTTP/1.1
Content-Type: multipart/form-data
Accept: multipart/form-data
MIME-Version: 1.0
Host: localhost:8080
Content-Type: multipart/form-data
Content-Length: 0
```

Response example:

```
HTTP/1.1 200 OK
```

21 FIWARE OpenSpecification Data SemanticAnnotation

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

Name	FIWARE.OpenSpecification.Data.SemanticAnnotation
Chapter	Data/Context Management,
Catalogue-Link to Implementation	<Semantic Annotation>
Owner	Telecom Italia, Mondin Fabio Luciano

21.1 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.

21.1.1 Copyright

- Copyright © 2012 by Telecom Italia

21.1.2 Legal Notice

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

21.2 Overview

21.2.1 Introduction to the Semantic Annotation GE

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

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

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

This GE provides a general purpose text analyzer to identify and disambiguate LOD (Linked Open Data) resources related to the entities in the text. It is build following a modular approach to optimize and distribute text processing & LOD sources (plug-in). Also it allows RDF triple generation that easily links to LOD resources.

The main conceptual idea of the SA GE is shown in the Figure 1 below.

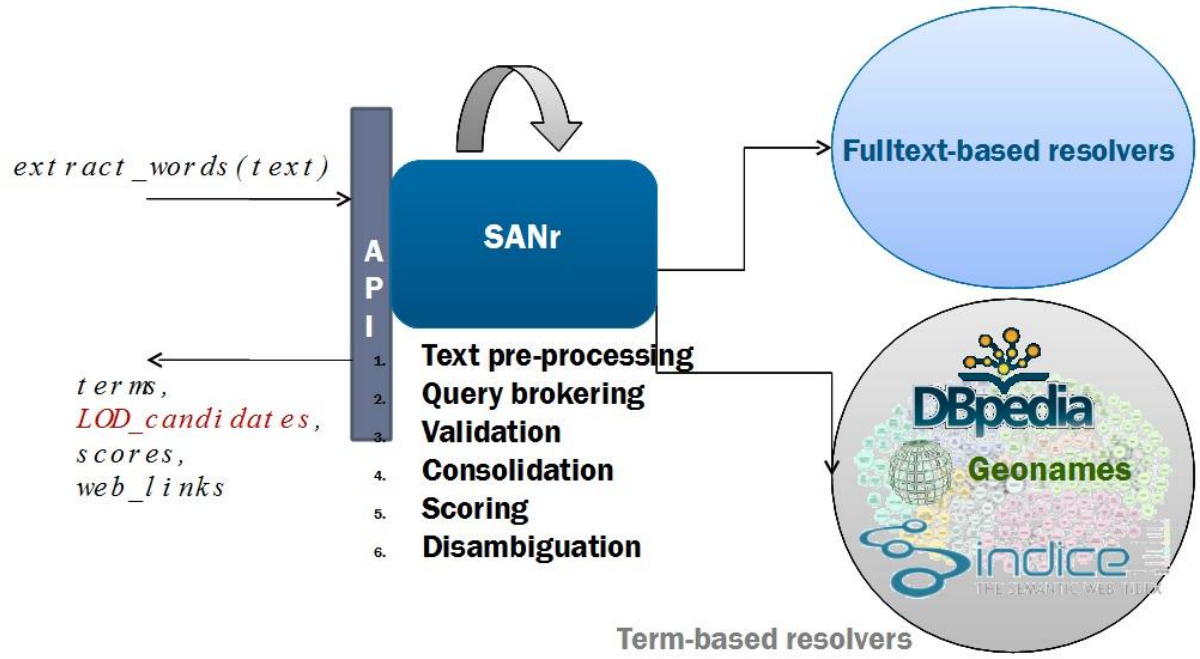


Figure 1: Conceptual Model of Semantic Annotation GE

21.2.2 Target usage

This GE may be used in the augmenting of content (news, books, etc) with additional information and links to LOD. It provides filtering and search based on LOD resources used as categories/tags.

Target users are all stakeholders that want to enrich textual data (tags or text) with meaningful and external content.

In the media era of the web, much content is text-based or partially contains text, either as media itself or as metadata (e.g. title, description, tags, etc). Such text is typically used for searching and classifying content, either through folksonomies (tag-based search), predefined categories, or through full-text based queries. To limit information overload with meaningless results there is a clear need to assist this searching process with semantic knowledge, thus helping in clarifying the intention of the user. This knowledge can be further exploited not only to provide the requested content, but also to enrich results with additional , yet meaningful content, which can further satisfy the user needs.

Semantics, and in particular Linked Open Data (LOD), is helpful in both annotating & categorizing content, but also in providing additional rich information that can improve the user experience.

As end-user content can be of any type, and in any language, such enabler requires a general purpose & multilingual approach in addressing the annotation task.

Typical users or applications can be thus found in the area of eTourism or eReading, where content can benefit from such functionality when visiting a place or reading a book, for example being provided with additional information regarding the location or cited characters.

The pure semantic annotation capabilities can be regarded as helpful for editors to categorize content in a meaningful manner thus limiting ambiguous search results (e.g. an article wouldn't be simply tagged with apple, but with its exact concept, i.e. a fruit, New York City or the brand)

21.2.3 Basic Design Principles

The Enabler has been designed following a modular approach, as it is shown in Figure 1. This way each component in the enabler can be developed or changed, given that it gives the same input/output format.

This leaves open the road to change data sources in order to have other data sources than dbpedia or geonames or to change the process standing behind the candidate's choice for each entity.

21.3 Basic Concepts

The GE has a web API, supports multilingual texts (it, en, es, pt) and includes "candidate" LOD resources and performs disambiguation. As result the GE created external links and HTML snippets (LOD information mashup).

The following modules listed in the Figure 2 are supposed to be included into the SA GE for its features and functionalities.

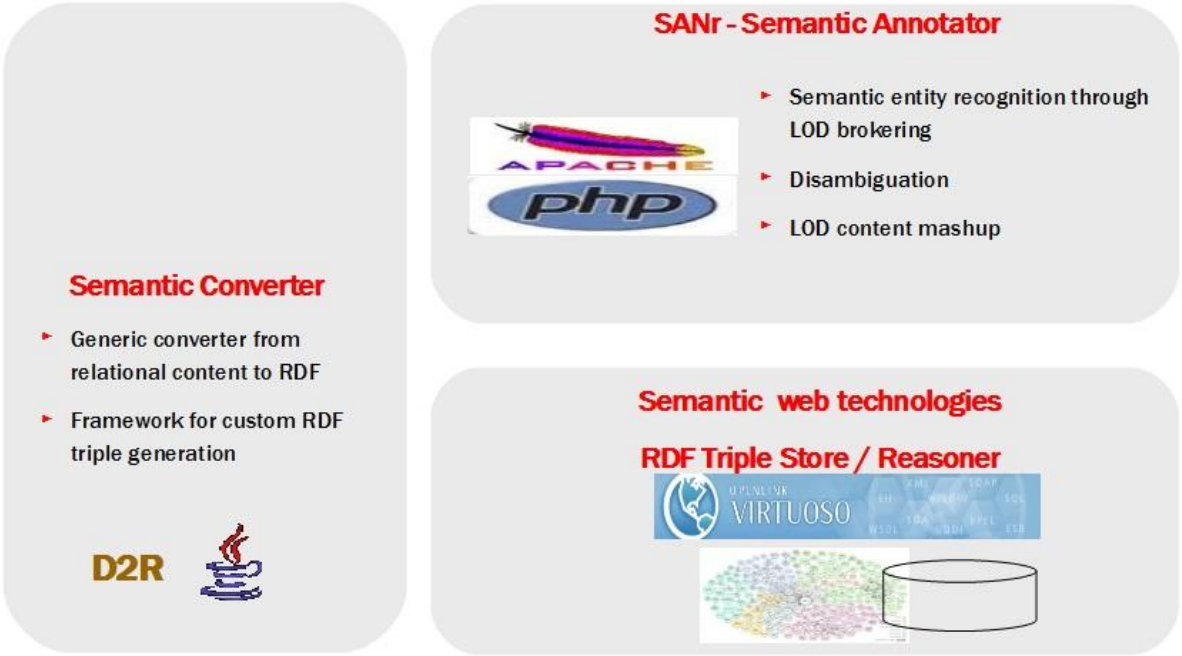


Figure 2: Semantic Annotation GE Modules

21.4 Main Interactions

The enabler basically consists of an API, which can be called by a simple HTTP GET request to this URL.

http://semantican.lab.fi-ware.eu/ajax/extract_words.php?text=

with a text to analyze as input which has to be passed as "text" parameter as shown above.

This API will:

1. Identigy Text Language

2. Identify Entities (People, Places, Organizations) in the Text

3. For each found entity It searches over Semantic Data Sources (DBPedia and Geonames) for related Linked Open Data Objects.

4. The found LOD objects for each entity are returned in JSON Format as "candidates". Eache candidate has a score. The candidate with the highest score is flagged as "preferred".

5. The query is logged into a DB with an ID.

Here's an example of the JSON return.


```
{
    "queryId": "12143",
    "lang": "it",
    "keywords": "Mario+Monti",
    "extags": "Mario Monti",
    "freeling": "Mario_Monti",
    "proc_time": "13",
    "terms": [
        {
            "id": "tc-Mario+Monti",
            "term": "Mario Monti",
            "candidates": [
                {
                    "id": "tag--Mario_Monti--
http://dbpedia.org/resource/Mario_Monti",
                    "label": "Mario Monti",
                    "uri":
"http://dbpedia.org/resource/Mario_Monti",
                    "type": "user",
                    "ext": "Mario Monti",
                    "extra": [],
                    "wrapper": "dbpedia",
                    "lev": "2",
                    "sim": "0.909090909091",
                    "sis": "1",
                    "jw": "0.963636363636",
                    "sc": "1",
                    "class": "empty",
                    "preferred": "true"
                }
            ],
```

```
            "html": "<fieldset><div class=panel><div class=header>A
proposito di <b>Mario Monti</b></div><div
class=panel_body></div></div><div class=panel><div
class=panel_body><img
src='http://upload.wikimedia.org/wikipedia/commons/thumb/3/33/Il_Pre
sidente_del_Consiglio_incaricato_Mario_Monti_(cropped).jpg/200px-
Il_Presidente_del_Consiglio_incaricato_Mario_Monti_(cropped).jpg'
height=160 /><br><div class=info>Ã^ senatore a vita dal 9 novembre
2011 e dal successivo 16 novembre assume, per la prima volta,
l'incarico di Presidente del Consiglio dei Ministri della Repubblica
Italiana e allo stesso tempo di Ministro dell'Economia e delle
Finanze dello stesso governo. Presidente dell'UniversitÃ  Bocconi
dal 1994, Monti Ã¨ stato c...<ul><li><a
href='http://www.guardian.co.uk/world/mario-monti'
target='_blank'>Link
utile</a></li></ul></div></div></div></fieldset><fieldset><legend>Co
ncetti associati a <strong>Mario Monti</strong></legend><ul><li><img
src='img/user.png' alt='user' title='user'> <a
href='http://dbpedia.org/resource/Mario_Monti' target='_blank'
title='[2-0.909090909091-0.963636363636/1]' >Mario Monti</a>
(dbpedia)</li></ul></fieldset>",
            "class": "empty"
        }
    ]
}
```


Moreover, by setting the 'html_snippet=on' parameter in the request URL, an HTML snippet for the preferred DBPedia entry is returned if possible. The HTML Snippet contains a Picture and Short Abstract for the resource.

FIWARE.OpenSpecification.Details.Data.SemanticAnnotation

21.5 Re-utilised Technologies/Specifications

Here is a list of Re-utilised Technologies for the enabler:

- **Freeling 2.2** The enabler uses Freeling as a language processing tool in order to perform Named Entity Recongition. [1]

- **Dbpedia** One of the most important general data sources used by the enabler. [2]

- **Geonames** Reference data source for places [3]

21.6 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.

- **Data** refers to information that is produced, generated, collected or observed that may be relevant for processing, carrying out further analysis and knowledge extraction. Data in FI-WARE has associated a **data type** and a**value**. FI-WARE will support a set of built-in **basic data types** similar to those existing in most programming languages. Values linked to basic data types supported in FI-WARE are referred as **basic data values**. As an example, basic data values like '2', '7' or '365' belong to the integer basic data type.
- A **data element** refers to data whose value is defined as consisting of a sequence of one or more <name, type, value> triplets referred as **data element attributes**, where the type and value of each attribute is either mapped to a basic data type and a basic data value or mapped to the data type and value of another data element.
- **Context** in FI-WARE is represented through **context elements**. A context element extends the concept of **data element** by associating an EntityId and EntityType to it, uniquely identifying the entity (which in turn may map to a group of entities) in the FI-WARE system to which the context element information refers. In addition, there may be some attributes as well as meta-data associated to attributes that we may define as mandatory for context elements as compared to data elements. Context elements are typically created containing the value of attributes characterizing a given entity at a given moment. As an example, a context element may contain values of some of the attributes "last measured temperature", "square meters" and "wall color" associated to a room in a building. Note that there might be many different context elements referring to the same entity in a system, each containing the value of a different set of attributes. This allows that different applications handle different context elements for the same entity, each containing only those attributes of that entity relevant to the corresponding application. It will also allow representing updates on set of attributes linked to a given entity: each of these updates can actually take the form of a context element and contain only the value of those attributes that have changed.
- An **event** is an occurrence within a particular system or domain; it is something that has happened, or is contemplated as having happened in that domain. Events typically lead to creation of some data or context element describing or representing the events, thus allowing them to processed. As an example, a sensor device may be measuring the temperature and pressure of a given boiler, sending a context element every five minutes associated to that entity (the boiler) that includes the value of these to attributes (temperature and pressure). The creation and sending of the context element is an event, i.e., what has occurred. Since the data/context elements that are generated linked to an event are the way events get visible in a computing system, it is common to refer to these data/context elements simply as "events".
- A **data event** refers to an event leading to creation of a data element.
- A **context event** refers to an event leading to creation of a context element.
- An **event object** is used to mean a programming entity that represents an event in a computing system [EPIA] like event-aware GEs. Event objects allow to perform operations on event, also known as **event processing**. Event objects are defined as a data element (or a context element) representing an event to which a number of standard event object properties (similar to a header) are associated internally. These standard event object properties support certain event processing functions.

22 FIWARE OpenSpecification Data SemanticSupport

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

Name	FIWARE.OpenSpecification.Data.SemanticSupport
Chapter	Data/Context Management,
Catalogue-Link to Implementation	<Semantic Application Support>
Owner	Atos Origin, Jose Maria Fuentes Lopez

22.1 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.

22.1.1 Copyright

22.1.2 Legal Notice

22.2 Overview

22.2.1 Introduction to the Semantic Application Support GE

The Semantic Web Application Support enabler aims at providing an effective environment for developers to implement and deploy high quality Semantic Web-based applications. The Semantic Web was first envisioned more than a decade ago by Tim Berners-Lee, as a way of turning the Web into a set of resources understandable not only for humans, but also by machines (software agents or programs), increasing its exploitation capabilities. The Semantic Web has focused the efforts of many researchers, institutions and IT practitioners, and received a fair amount of investment from European and other governmental bodies. As a result of these efforts, a large amount of mark-up languages, techniques and applications,

ranging from semantic search engines to query answering system, have been developed. Nevertheless the adoption of Semantic Web from the IT industry is still following a slow and painful process.

In recent years, several discussions had taken place to find out the reasons preventing Semantic Web paradigm adoption. There is a general agreement that those reasons range from technical (lack of infrastructure to meet industry requirements in terms of scalability, performance. distribution, security, etc.) to engineering (not general uptake of methodologies, lack of best practices and supporting tools), and finally commercial aspects (difficulties to penetrate in the market, lack of understanding of the main strengths and weaknesses of the semantic technologies by company managers, no good sales strategies, etc.).

The Semantic Application Support enabler addresses part of the abovementioned problems (engineering and technical) from a data management point of view, by providing:

- An infrastructure for metadata publishing, retrieving and subscription that meets industry requirements like scalability, distribution and security. From now and so on, we will refer to this infrastructure as SWAS Infrastructure.
- A set of tools for infrastructure and data management, supporting most adopted methodologies and best practices. From now and so on, we will refer to this tools as SWAS Engineering Environment.

22.2.2 Target usage

Target users are mainly ontology engineers and developers of semantically-enabled applications that need RDF storage and retrieval capabilities. Other GE from the FI-WARE, such as for example the GE for semantic service composition or the query broker, or from the usage areas of the PPP that need semantic infrastructure for storage and querying are also target users of this GE.

22.2.3 Example Scenario

There is a need for semantically-enabled applications in many fields and domains, ranging from research projects to enterprise intranets or public web sites. Semantic applications often rely on ontologies and knowledge bases to develop business functionality such as discovery, composition, annotation, etc., with the aim of enhancing exploitation capabilities of resource (services, text documents, media documents, etc.). The need of an infrastructure that ease the development, storage and use of ontologies and allow practitioners to efficiently manage their knowledge bases, providing the means to manage metadata effectively is therefore of paramount interest.

The TaToo (http://www.tatoo-fp7.eu/tatooweb/) project can be taken as an example in order to show how this generic enabler can help to future internet application developers. TaToo is a research project on the environmental domain with the goal of developing tools to facilitate the discovery of environmental resources. In order to enhance the discovery process, the application stores annotations (metadata) of existing environmental resources by tagging them with ontology terms. Therefore, a ontology framework [Pariente 2011] has been developed including three domain ontologies that describe three different environmental domains plus a bridge ontology that allow cross domain interoperability. Moreover, TaToo ontologies are not built from scratch but by reusing, complete or partially, existing ontologies publicly available over the internet. Nowadays the TaToo ontology framework is the result of the integration of more than 15 ontologies. The development of such a framework is a complex task, involving several domain experts and ontology developers. To achieve a

common understanding, is imperative the usage of appropriate methodology and tools. In TaToo the NeOn Methodology [Suarez-Figueroa 2008] and the NeOn Toolkit [NeOn-Toolkit], one of the baseline assets of this generic enabler, have been the basis of the ontology engineering process. The Neon Toolkit helped TaToo's ontology developers to apply the NeOn methodology to develop ontologies providing several functionality such us ontology edition, ontology modularization, ontology search, etc. Besides, these ontologies are expected to evolve in time, and therefore they would need a system that helps the ontology expert to tackle the ontology evolution problems. This is not completely covered by the NeOn Toolkit, as there are aspects such as ontology versioning, knowledge base maintenance, workspace environments, etc. that are not fully covered by the tool. This functionality will be developed in the scope of FI-WARE project. Figure SWAS-1 shows a screenshot of NeOn Toolkit being used in the scope of TaToo.

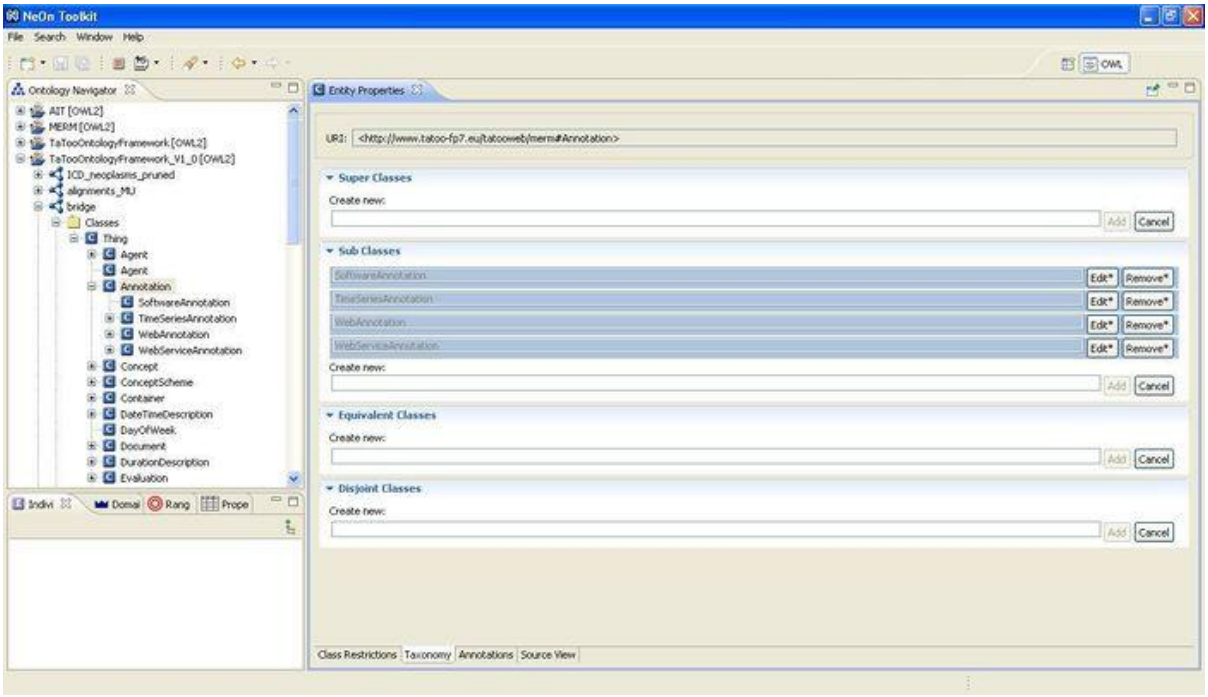


Figure SWAS-1: NeOn Toolkit screenshot

Once ontologies are developed they need to be uploaded to a knowledge base with inference capabilities to be used by business logic components. In TaToo, Sesame [Sesame] and OWLIM [OWLIM], two of the assets selected as baseline assets for this enabler, have been used as knowledge base implementation. However Sesame and OWLIM are just RDF / OWL oriented storages, so there is a lack of knowledge base management capabilities. As an example, once a ontology is loaded into a Sesame workspace it is not possible to keep track of this fact, so in case the ontology evolve in time, there is no possibility to establish a relation between the workspace and the loaded ontology in order to update the workspace. This kind of knowledge base management problems will be tackled and solved by the Semantic Support Application GE in the scope of FI-WARE.

Summarizing, a project such as TaToo might benefit from an enabler that provides an ontology and knowledge base management system integrated with an ontology engineering environment supporting strong ontology development methodology, covering the whole semantic web application lifecycle. This is clearly extensible to many different Semantic Web-based applications.

22.3 Basic Concepts

This section introduces the basic concepts related to the Semantic Support Application GE including ontologies, ontology languages and ontology development methodologies.

22.3.1 Ontologies

[Gruber 1993] introduced the concept of ontology as “a formal explicit specification of a shared conceptualization”. Thus, in the Semantic Web, ontologies play the role of a formal (machine-understandable) and shared (in a domain) backbone. Ontologies are becoming a clear way to deal with integration and exploitation of resources in the several domains. Starting from Gruber’s definition is it possible to infer some of the key features that make ontologies a valuable knowledge engineering product:

- Ontologies are formal, so they are supposed to be machine-understandable.
- Ontologies have explicit definitions, so they are storable, interchangeable, manageable, etc.
- Ontologies are shared, so they are supposed to be agreed, supported and accessible by a broad community of interest.
- Ontologies are a conceptualization, so they are supposed to be expressive enough to model wide knowledge areas.

In order to efficiently develop ontology networks that really fulfill these features, a wide range of elements are needed, ranging from appropriate methodologies, to tools supporting those methodologies and appropriate infrastructures to allow management of the ontology lifecycle. Providing such a support is the aim of this GE.

To do so, some decisions have been taken in order to limit the scope of the GE:

- To select [OWL-2 RL] as reference language for ontology formalization.
- To select NeOn Methodology as reference methodology for ontology development.

Both decisions will be discussed in the following sections.

22.3.2 OWL-2

Since the inception of the ontologies, several ontology languages, with different expressivity, serialization and underlying logic formalisms have rise and fall (OWL, WSML, F-Logic, OIL, KIF, etc.). Sometimes these languages differ in their serialization, sometimes in their background logic and sometimes they are just designed with a different purpose. Therefore, provide functionality for every single ontology language is almost an impossible task. In consequence, in the scope of the Semantic Web Application Generic Enabler OWL-RL (a decidable subset of OWL, the W3C standard and most popular ontology language) has been selected as reference for ontology definition. Some of the reasons supporting this decision are now introduced:

- Since October 2009 OWL-2 is a W3C recommendation for ontology definition.
- OWL-2 RL provides a good trade-off between expressivity and performance. Inference over OWL-2 RL guarantees that inference process will finish in a finite amount of time.
- OWL-2 RL is based in previous W3C standards such as [RDF] and [RDFs] so previous ontologies could be also managed by the proposed infrastructure.

22.3.3 Ontology Engineering

Semantic Web Application Support GE aims to provide the means for FI Applications developers to develop Semantic Web enabled applications efficiently. Ontology development, one of the key points in these applications, is a complex, expensive and time-consuming process that includes different activities, such as specifying requirements, information extraction, logical modeling, etc. In order to efficiently manage this process, it is necessary to use a methodology and its supporting tools. Due to its adoption and maturity, Semantic Application Support GE will provide the means to support the NeOn Methodology. The NeOn Methodology defines a methodology for ontology development that covers the whole ontology lifecycle. The NeOn methodology includes extracted elements of previous methodologies like METHONTOLOGY [Fernandez 1997], On-To-Knowledge [OnToKnowledge 2001] and DILIGENT [DILIGENT 2004]. The NeOn methodology increases the level of descriptive detail, and provides two new features: ontology creation from existing resources (both ontological or not) and ontology contextualization. In this way, NeOn offers a general methodology for ontology development useful across different technological platforms and that specifies each process and activity of the methodology, defining its purpose, inputs, outputs, involved actors, applicable techniques, tools and methods, when its execution is necessary, etc.

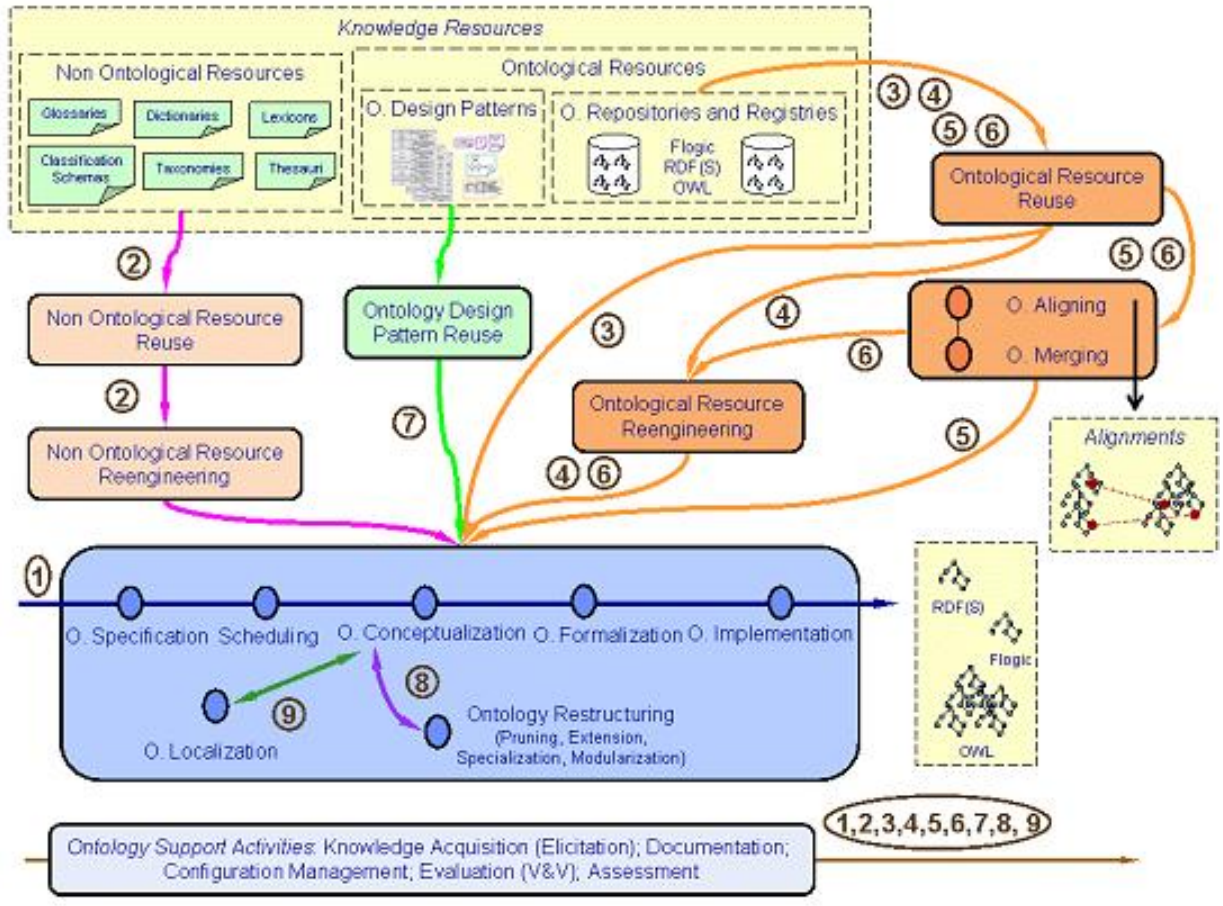


Figure SWAS-2: NeOn Methodology overview (from Suárez-Figueroa, 2008, with permission)

The NeOn methodology presents and describes nine of the most common scenarios that may arise during ontology development:

1. Specification for implementation from scratch.
2. Reusing and re-engineering non-ontological resources.
3. Reusing ontological resources.
4. Reusing and re-engineering ontological resources.
5. Reusing and merging ontological resources.
6. Reusing, merging and re-engineering ontological resources.
7. Reusing ontology design patterns.
8. Restructuring ontological resources.
9. Localizing ontological resources.

Scenario 1 represents the base case, whereas the rest of the scenarios are related to it as shown in Figure SWAS-2. For each of these scenarios, the NeOn methodology establishes detailed guidelines, tools to use, etc.

The Semantic Web Application Support GE should provide an ontology engineering environment supporting processes and activities outlined in the NeOn methodology.

22.4 Semantic Application Support GE Architecture

The objective of the Semantic Application Support GE is to facilitate the Ontology Engineering process providing a set of tools that allow the ontology reutilization using repositories to publish and share ontologies between projects. The developer can use the published ontologies to create semantic repositories to support specific needs.

In order to satisfy the previous objective, the Semantic Application Support GE is divided in a client-side Engineering Environment and a server-side Infrastructure. Figure SWAS-3 presents the SWAS Infrastructure architecture.

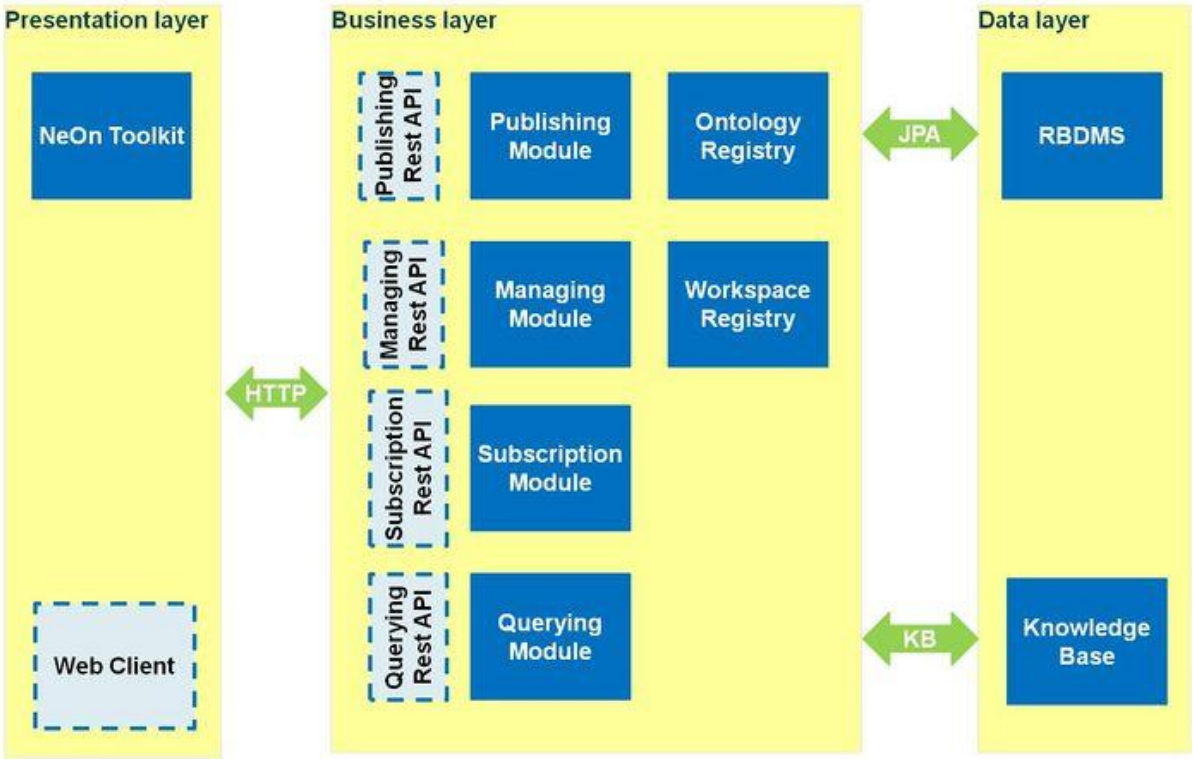


Figure SWAS-3: SWAS Infrastructure architecture

As it is shown in the diagram, it follows a typical three layer Java Enterprise Architecture. Components included in business and presentation layers are JEE based. In the data layer, two components can be found:

- A relational database, that will be used by Ontology Registry to store ontology documents loaded into the GE.
- A Knowledge Base providing OWL-2RL support. This Knowledge Base will be used by ontology and workspace registries to store ontology and workspace related metadata and by managing, querying and publishing modules to provide their functionality.

Business components will interact with data layer components by means of two different mechanisms. To interact with the relational database, business components will use JPA (Java Persistence API) that make business components database system independent. Unfortunately such an abstraction mechanism is not available for knowledge base required interaction so business components interacting with the knowledge base will be knowledge base implementation dependent. In Semantic Web Application Support reference implementation, the combination of Sesame and OWLIM has been chosen as knowledge base implementation. Knowledge base independence feature will be study for future releases.

Business Layer contains following components:

- Ontology registry that manages ontologies loaded into the system and its related metadata. Operations such us retrieving / uploading ontology, retrieving / uploading metadata, etc would be provided by this component. A description of methods provided for FI-WARE first release can be found in Backend Functionality section.
- Workspace registry that manages workspaces and their related metadata created by users to be used by their semantic enable applications. Operations such us creating /

deleting a workspace, listing ontologies loaded into the workspace, etc would be provided by this component. Description of methods belonging to this component will be described in future FI-WARE releases.

- Publishing module that allow user to publish data into the GE. Data can be either ontologies or RDF serialized content. In case of ontologies, publishing module will rely on ontology registry functionality. In case of RDF serialized content, publishing module will store the content in proper knowledge base workspace in collaboration with workspace registry. In both cases publishing module will update subscription module if needed. Description of methods belonging to this component will be described in future FI-WARE releases.
- Managing module that allow users to monitor the status of the GE. Operations such as retrieving a list of available ontologies, retrieving a list of subscriptions, etc will be provided by this module. Managing module will rely on the rest of business components to provide its functionality. Description of methods belonging to this component will be described in future FI-WARE releases.
- Subscription module that allows users to subscribe to events produced in the GE. Operations such as subscribing to ontology updates or workspace modifications will be provided by this module. Description of methods belonging to this component will be described in future FI-WARE releases.
- Querying module that allows users to query their workspace following SPARQL Query Protocol. Description of methods belonging to this component will be described in future FI-WARE releases.

In order to provide GE functionality in a platform independent way, several Rest APIs will be developed. In this first FI-WARE release, a subset of methods belonging to publish and managing APIs will be provided. Therefore, clients or presentation layer applications will interact with business components by means of HTTP requests / responses.

SWAS Engineering Environment provides comprehensive support for the ontology engineering life-cycle. Concrete details about SWAS Engineering Environment functionality will be provided in Frontend Functionality section. SWAS architecture is based on Eclipse architecture, a leading development environment providing a technical layer for easy creation of new features and supported for a huge development community. Figure SWAS-4 shows the SWAS Engineering Environment architecture.

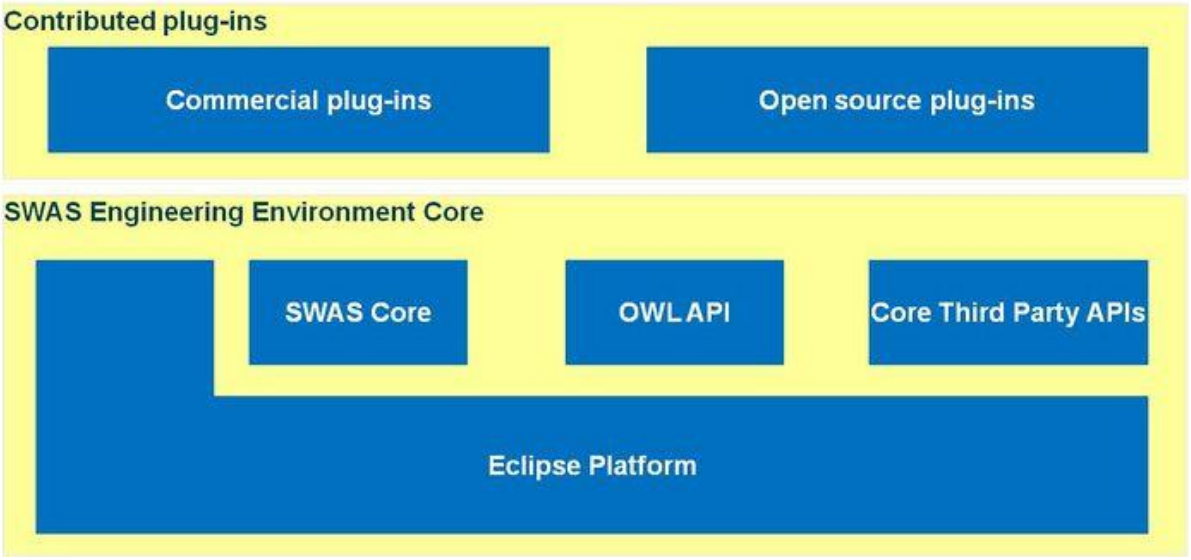


Figure SWAS-4: SWAS Engineering Environment architecture

As it shown in the diagram SWAS Engineering Environment is divided into two layers, the SWAS Engineering Environment core and the contributed plug-ins. The SWAS Engineering Core provides the core ontology editing functionality. The contributed plug-ins are extensions that provide extra functionality supporting different phases of the NeOn Methodology.

22.5 Main Interactions

22.5.1 Modules and Interfaces

This section covers the description of the Semantic Web Application Support GE main functionality. The description of this functionality is based on the functionality provided by the baseline asset in FI-WARE first release. Section Backend functionality describes functionality (methods) provided to agents in a service like style. Section Frontend functionality describes functionality provided to human users through a GUI.

22.5.2 Backend Functionality

Backend functionality describes functionality provided by the GE enabler as service invocation methods for both human or computer agents. As described in Architecture section, this functionality is accessible by means of Rest Web Services API. In this first FI-WARE release, a sub set of methods belonging to publishing and managing rest APIs will be provided:

- ***Publishing Rest API***.
    - ***Get ontology version***: Retrieves from the GE the ontology document identified by a given ontology IRI (obtained using the list ontologies service) and version IRI. To invoke the operation, a GET http request should be sent to http://<ge url location>/ontology-registry/ontologies/<ontology IRI>/<version IRI>.
    - ***Get ontology***: Similar to Get ontology version. It retrieves from the GE the latest version of the ontology document identified by a given ontology IRI

(obtained using the list ontologies service). To invoke the operation, a GET http request should be sent to http://<ge url location>/ontology-registry/ontologies/<ontology IRI>.

- ***Delete ontology version***: Removes from the GE the ontology document identified by a given ontology IRI (obtained using the list ontologies service) and version IRI. To invoke the operation, a DELETE http request should be sent to http://<ge url location>/ontology-registry/ontologies/<ontology IRI>/<version IRI>.
- ***Delete ontology***: Similar to Delete ontology version. It removes from the GE the latest version of the ontology document identified by a given ontology IRI (obtained using the list ontologies service). To invoke the operation, a DELETE http request will be sent to http://<ge url location>/ontology-registry/ontologies/<ontology IRI>.
- ***Upload ontology version***: Uploads to the GE an ontology document and identifies it with a given ontology IRI (obtained using the list ontologies service) and version IRI. To invoke the operation, a PUT http request should be sent to http://<ge url location>/ontology-registry/ontologies/<ontology IRI>/<version IRI> with an file attachement including the ontology RDF/XML serialization.
- ***Upload ontology***: Similar to Upload ontology version. Uploads an ontology document to the GE and identifies it with a given ontology IRI and with the latest version IRI available. To invoke the operation, a PUT http request should be sent to http://<ge url location>/ontology-registry/ontologies/<ontology IRI> with an file attachment including the ontology RDF/XML serialization.
- ***Get ontology version metadata***: Retrieves from the GE an ontology document containing the metadata related to an ontology document identified by a given ontology IRI (obtained using the list ontologies service) and version IRI. To invoke the operation, a GET http request should be sent to http://<ge url location>/ontology-registry/metadata/<ontology IRI>/<version IRI>.
- ***Get ontology metadata***: Similar to Get ontology version metadata. It retrieves from the GE an ontology document containing the metadata related to the latest version of the ontology document identified by a given ontology IRI (obtained using the list ontologies service). To invoke the operation, a GET http request should be sent to http://<ge url location>/ontology-registry/metadata/<ontology IRI>.
- ***Delete ontology version metadata***: Removes from the GE the metadata related to an ontology document identified by a given ontology IRI and version IRI. To invoke the operation, a DELETE http request will be sent to http://<ge url location>/ontology-registry/metadata/<ontology IRI>/<version IRI>.
- ***Delete ontology metadata***: Similar to Delete ontology version metadata. Removes from the GE the metadata related to the latest version of the ontology document identified by a given ontology IRI. To invoke the operation, a DELETE http request should be sent to http://<ge url location>/ontology-registry/metadata/<ontology IRI>.
- ***Upload ontology version metadata***: Uploads to the GE an ontology document containing metadata related to an ontology document identified by a given ontology IRI (obtained using the list ontologies service) and version IRI.

To invoke the operation, a PUT http request should be sent to http://<ge url location>/ontology-registry/metadata/<ontology IRI>/<version IRI> with an file attachement including the metadata RDF/XML serialization. Metadata uploaded must complain to OMV (Ontology metadata vocabulary).

- ***Upload ontology metadata***: Similar to Upload ontology version metadata. It uploads to the GE an ontology document containing metadata related to the latest version of an ontology document identified by a given IRI (obtained using the list ontologies service). To invoke the operation, a PUT http request should be sent to http://<ge url location>/ontology-registry/metadata/<ontology IRI> with an file attachement including the metadata RDF/XML serialization. Metadata uploaded must complain to OMV (Ontology metadata vocabulary).

- ***Managing Rest API***.
  - ***List ontologies***: Retrieves an XML document containing the list of ontology documents and their versions loaded into the GE. To invoke the operation, a GET http request should be sent to http://<ge url location>/ontology-registry/mgm/list. As a result, an xml encoding the requested information will be sent as response.
  - ***List ontology versions***: Similar to List ontologies. Retrieves an XML document containing the versions of an ontology document identified by a given ontology IRI loaded into the GE. To invoke the operation, a GET http request should be sent to http://<ge url location>/ontology-registry/mgm/<ontology IRI>. As a result, an xml encoding the requested information will be sent as response.

All methods described can be invoked by mean of regular HTTP requests either using a web browser (for those ones who rely on GET requests) or by a programmatic APIs such as Jersey. The query and subscription modules will be provided in the next releases of the Semantic Web Application Support GE.

22.5.3 Frontend Functionality

SWAS Engineering Environment functionality is based on the functionality provided by the baseline asset NeOn Toolkit. The NeOn Toolkit is a state-of-the-art, eclipse based, open source multi-platform ontology engineering environment, which provides comprehensive support for the ontology engineering life-cycle. Due to its nature, it wouldn't be possible to describe all SWAS Engineering Environment provided functionality in a service like manner. Anyway, an overview of the functionality required for the SWAS Engineering Environment is now introduced. To try to better describe functionality, some screenshots from baseline asset NeOnt Toolkit will be used in this section. Figure SWAS-6 presents an overview of NeOn Toolkit GUI.

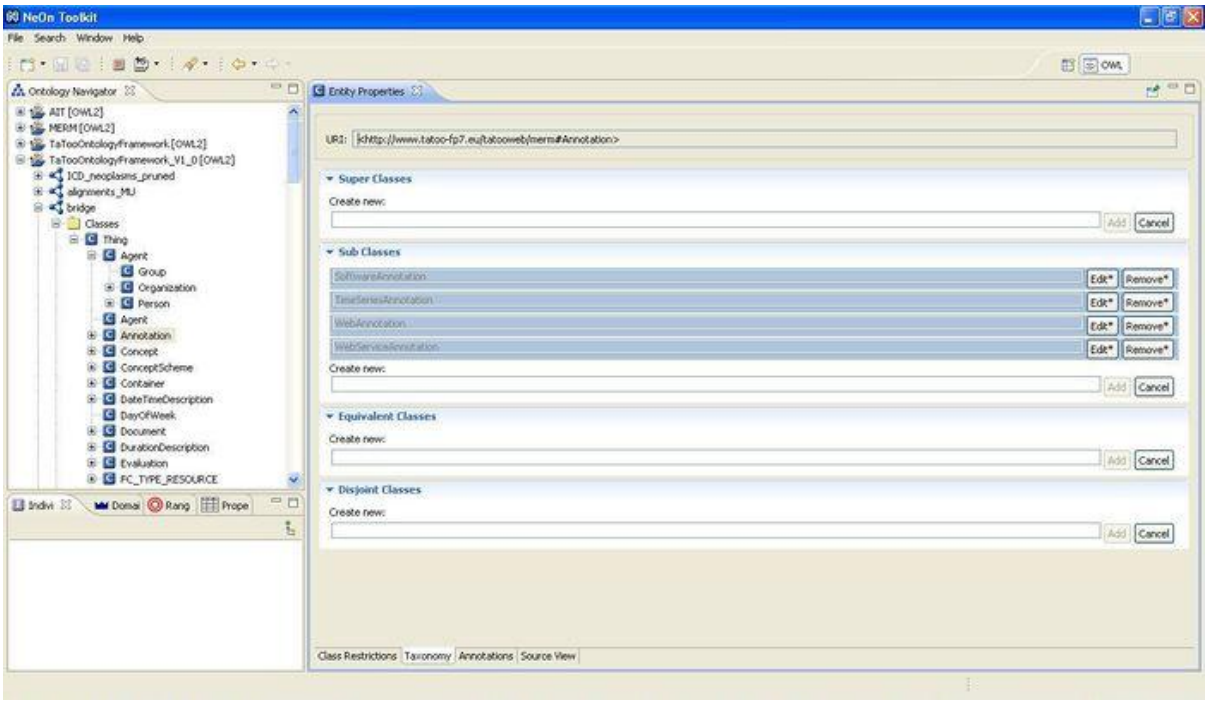


Figure SWAS-6: NeOn Toolkit main window

SWAS GE Engineering Environment will follow and take advantage some of the paradigms introduced by [Eclipse], one of the leading development environments, including:

- Using workspaces, projects, folders and files as containers to organize and store development artifacts.
- Using workbench, editors, views and perspectives to provide functionality to the user by means of GUI.

Therefore, most of the functionality provided by SWAS Engineering Environment is provided as editors, views and perspectives. Figure SWAS-7 presents the Ontology navigation perspective.

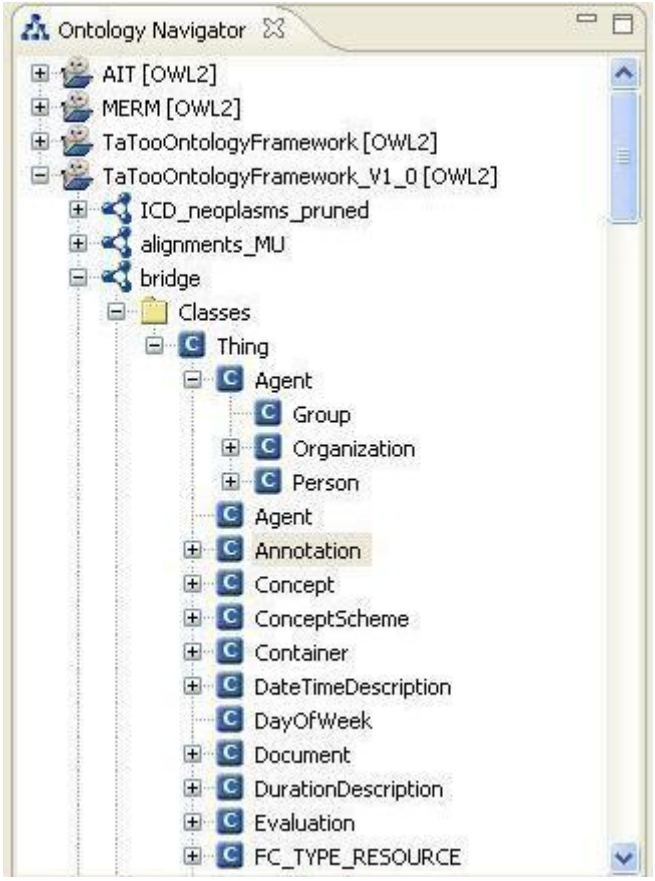


Figure SWAS-7: Ontology navigation perspective

Under this perspective users are able to manage their projects and ontologies, creating or removing projects, loading or creating new ontologies, etc. In the scope of a given ontology, users are able to manage (adding, removing, etc) main ontology contents such as classes, object properties and data properties. Once selected, ontology contents can be edited by means of a proper editor. Figure SWAS-8 presents the class editor.

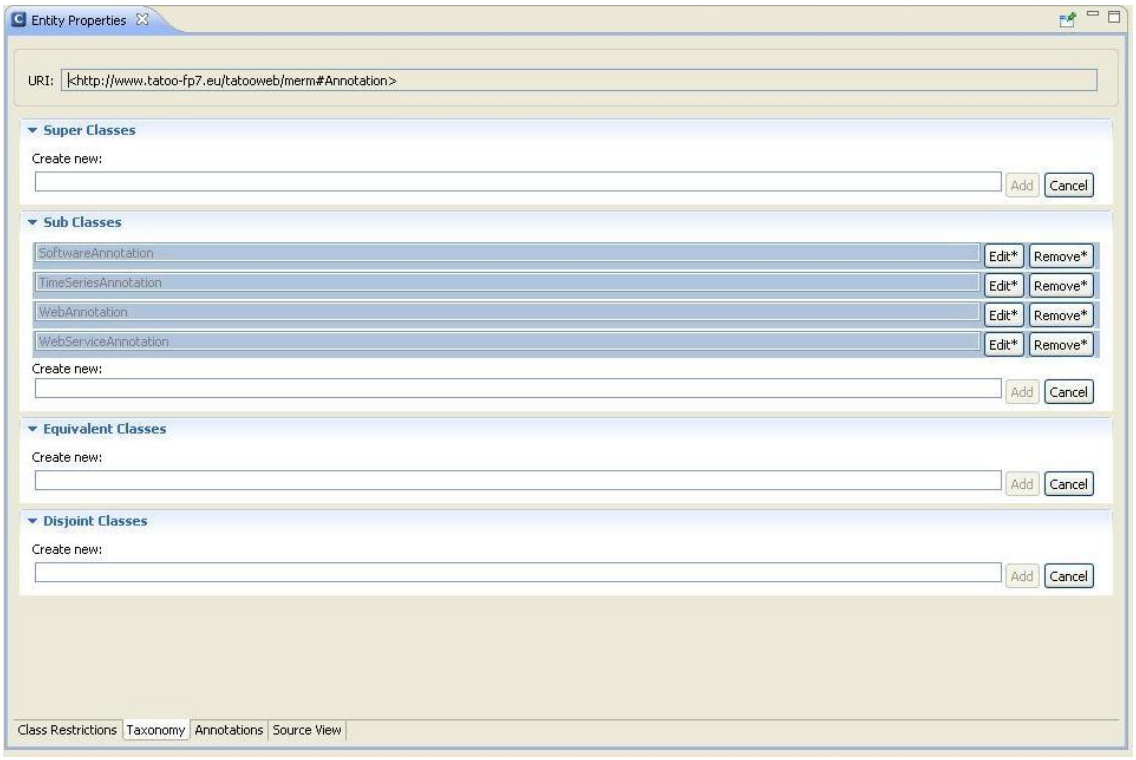


Figure SWAS-8: Class editor

Class editor is composed of four tabs:

- Class restrictions tab that allow the user to modify restrictions applicable to the class.
- Taxonomy tabs that allow the user to modify the class ancestors, successors or siblings.
- Annotation tab that allows the user to annotate the class con textual descriptions.
- Source tab that presents the user the OWL code generated for the described class.

Data property and object property editor provide similar functionality for data and object properties. Finally, views present additional information about the items selected in the ontology navigation perspective. Figure SWAS-9 presents the range view.

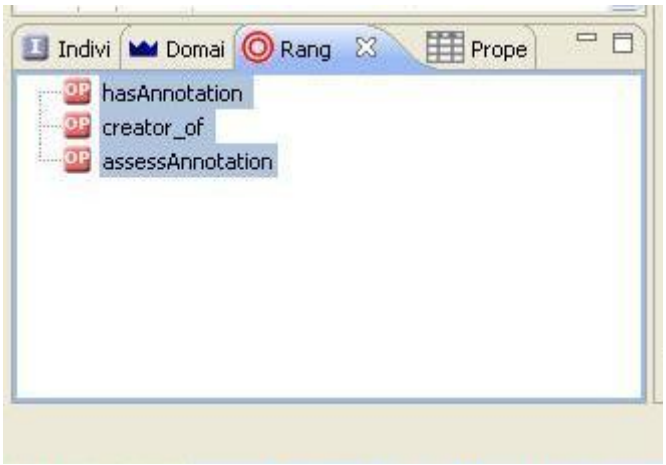


Figure SWAS-9: Range view

Range view presents for each class, the set of object properties that has the selected class as range. As mentioned in Semantic Application Support GE Architecture section, Engineering Environment functionality can be extended by means of plug-ins. Nowadays there are more than 30 active plug-ins for NeOn Toolkit covering a wide range of functionality covering several steps of the NeOn Methodology. Some of this plug-ins functionality may inspire Engineering Environment functionality in the future if needed.

22.6 Design Principles

The main goal of the Semantic Web Application Enabler is to provide a framework for ontology engineers and developers of semantically-enabled applications offering RDF/OWL management, storage and retrieval capabilities. This goal will be achieved by providing an infrastructure for metadata publication, retrieval and subscription that meets industry requirements like scalability, distribution and security, plus a set of tools for infrastructure and metadata-data management, supporting most adopted methodologies and best practices.

The Semantic Web Application enabler is based on the following design principles:

- Support standards: Support for RDF/OWL, the most common standards used in Semantic Web applications.
- Methodological approach: GE is strongly influenced by methodological approaches, so it will adopt and support, as far as possible, most adopted methodologies to achieve its goals.
- Semantic repository features: Provide high-level common features valid for most of the existing solutions in the semantic web in terms of RDF / OWL storage and inference functionalities.
- Ontology management: The enabler will provide an ontology registry and the API to control it, including some high-level ontology management functionalities.
- Knowledge Base management: The enabler will provide a knowledge base registry and the API to control it, including some high level knowledge base management functionalities.
- Extensibility: The most important part of the architecture design of the enabler is to define interfaces that allow the extensibility of the system. Where applicable the design should also be modular, to facilitate future extensions and improvements. The reference implementations should comply with this common design.

In the scope of FI-WARE first release, the work in the Semantic Application Support GE enabler has been mostly related to the ontology registry component. Some decisions regarding this component design has been taken, including: selection of an ontology metadata format, definition of a format for ontology identifiers, definition of an interface for exposing ontology registry functionality and decision on how to storage ontologies.

In order to provide advanced ontology management functionalities, ontologies should be annotated with extended metadata. In order to do so, the selection of a suitable ontology metadata format is needed. In this case, the Ontology Metadata Vocabulary [OMV] has been selected. Some of its key features are:

- OWL-2 ontology developed following NeOn Methodology by consortium members.
- Designed to meet NeOn Methodology reusability use case requirements.
- Extensible, reusable, accessible and interoperable.

OMV describes some metadata regarding ontologies that should be provided by users while loading ontologies into the GE. This metadata include information about ontology developers, ontology language, ontologies imported by the ontology, etc. A class diagram showing OMV main classes and attributes can be found in **Figure X**.

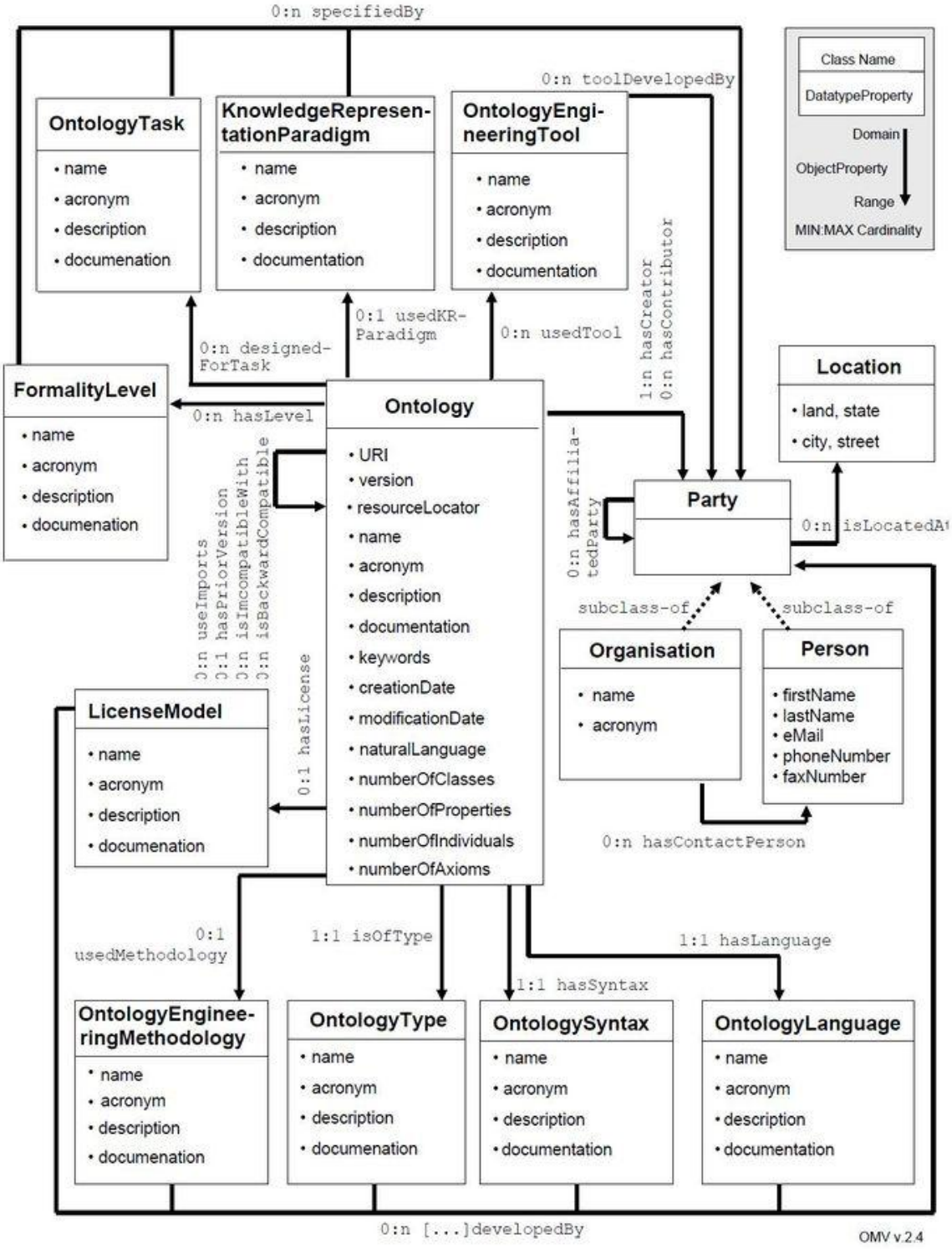


Figure X: Ontology Metadata Vocabulary UML diagram

In order to be stored into the ontology registry, it would be needed to assign to the ontology a unique identifier. Identifying ontologies may look an easy task but it is not even completely tackle even in OWL-2 specification. Taking a look to OWL-2 specification it can be found that:

- *Each ontology may have ontology IRI, which is used to identify an ontology. If an ontology has an ontology IRI, the ontology may additionally have a version IRI, which is used to identify the version of the ontology*
- *The ontology document of an ontology O should be accessible via the IRIs determined by the following rules:*
    - *If O does not contain an ontology IRI (and, consequently, it does not contain a version IRI either), then the ontology document of O may be accessible via any IRI.*
    - *If O contains an ontology IRI OI but no version IRI, then the ontology document of O should be accessible via the IRI OI.*
    - *If O contains an ontology IRI OI and a version IRI VI, then the ontology document of O should be accessible via the IRI VI; furthermore, if O is the current version of the ontology series with the IRI OI, then the ontology document of O should also be accessible via the IRI OI.*

For the sake of the implementation, in the scope of the Semantic Application Support, ontologies must have ontology IRI and version IRI. Ontology IRI must be provided by the user while version IRI may be provided by the GE in some cases. Moreover, ontology documents will be accessible using its ontology IRI plus version IRI, being this last one optional. In case no version IRI is provided, latest version of the ontology identified by the ontology IRI will be provided while accessing.

The Semantic Application Support GE will need to store then two kinds of resources: ontologies and ontology metadata. Having selected OMV as ontology metadata format, ontology metadata would need to be stored into a RDF triple store with OWL capabilities. In case of ontologies, they will be managed as plain text objects and stored in a regular relational data base. This would avoid potential problems of performance while serving ontologies for developers with editing purposes.

Finally, an interface for accessing the ontology registry should be provided. In this case, Semantic Application Support GE will follow [SPRQL Query protocol]: *If a service supports HTTP bindings, it must support the bindings as described in specification. A SPARQL Protocol service may support other interfaces such as SOAP*. In the case of this GE, a RESTful based service will implement the interface of the ontology registry. This interface is described in main interactions section.

22.7 References

[Pariente 2011]	Lobo, T. P., Lopez J. M. F., Sanguino M. A., Yurtsever S., Avellino G., Rizzoli A. E., et al. (2011). A Model for Semantic Annotation of Environmental Resources: The TaToo Semantic Framework. ISESS 2011.
[Suarez-Figueroa 2008]	Suárez-Figueroa, M. C., et al. NeOn D5.4.1. NeOn Methodology for Building Contextualized Ontology Networks. February 2008.
[NeOn-Toolkit]	The NeOn Toolkit, http://neon-toolkit.org/

[Sesame]	Sesame RDF framework, http://www.openrdf.org/
[OWLIM]	OWL Semantic Repository, http://www.ontotext.com/owlim/
[Gruber 1993]	Gruber, T.: A translation approach to portable ontology specifications. Knowledge Acquisition 5, 1993
[OWL-2 RL]	http://www.w3.org/TR/owl-profiles/#OWL_2_RL
[RDF]	Beckett D., RDF/XML Syntax Specification (Revised). W3C Recommendation, 10 February 2004
[RDFs]	Dan Brickley D, Guha R.V., RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, 10 February 2004
[Fernandez 1997]	Fernández-López M., Gómez-Pérez, A. & Juristo, N.: Methontology: from ontological art towards ontological engineering. Proc. Symposium on Ontological Engineering of AAAI, 1997
[OnToKnowledge 2001]	Broekstra, J., Kampman, A., Query Language Definition. On-To-Knowledge (IST-1999-10132), 2001
[DILIGENT 2004]	Pinto H.S., Tempich C., Staab S., Sure Y.: Diligent: Towards a fine-grained methodology for distributed, loosely-controlled and evolving engineering of ontologies. In de M´antaras L.R., Saitta L., editors, Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004), August 22nd - 27th, pages 393–397, Valencia, Spain, AUG 2004. IOS Press.
[OMV]	Jens Hartmann, Raúl Palma, York Sure, María del Carmen Suárez-Figueroa, Peter Haase, Asunción Gómez-Pérez, Rudi Studer: Ontology Metadata Vocabulary and Applications. OTM Workshops 2005: 906-915
[SPRQL Query protocol]	http://www.w3.org/TR/rdf-sparql-protocol/
[Eclipse]	http://www.eclipse.org/

22.8 Detailed Specifications

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

22.8.1 Open API Specifications

- Semantic Support Open RESTful API Specification (PRELIMINARY)

22.8.2 Other Open Specifications

- FIWARE.ArchitectureDescription.Data.SemanticSupport.OMV_Open_Specification_(DRAFT)

22.9 Re-utilised Technologies/Specifications

The Semantic Application Support GE uses a set the well know specifications in the ontology engineering domain as also in software engineering, these specifications are:

- Resource Description Framework – RDF (W3C standard).
- W3C Web Ontology Language – OWL (W3C Recommendation).
- SPARQL Query Language for RDF (W3C Recommendation).

In addition a set of APIs and tools have been used in development of GE:

- Java API for RESTful Web Services - JAX-RS
- Java Persistence API – JSR 317
- OpenRDF Sesame Semantic Repository
- Ontotext OWLIM Semantic Reasoner

22.10 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.

- **Data** refers to information that is produced, generated, collected or observed that may be relevant for processing, carrying out further analysis and knowledge extraction. Data in FI-WARE has associated a **data type** and a**value**. FI-WARE will support a set of built-in **basic data types** similar to those existing in most programming languages. Values linked to basic data types supported in FI-WARE are referred as **basic data values**. As an example, basic data values like ‘2’, ‘7’ or ‘365’ belong to the integer basic data type.
- A **data element** refers to data whose value is defined as consisting of a sequence of one or more <name, type, value> triplets referred as **data element attributes**, where the type and value of each attribute is either mapped to a basic data type and a basic data value or mapped to the data type and value of another data element.
- **Context** in FI-WARE is represented through **context elements**. A context element extends the concept of **data element** by associating an EntityId and EntityType to it,

uniquely identifying the entity (which in turn may map to a group of entities) in the FI-WARE system to which the context element information refers. In addition, there may be some attributes as well as meta-data associated to attributes that we may define as mandatory for context elements as compared to data elements. Context elements are typically created containing the value of attributes characterizing a given entity at a given moment. As an example, a context element may contain values of some of the attributes “last measured temperature”, “square meters” and “wall color” associated to a room in a building. Note that there might be many different context elements referring to the same entity in a system, each containing the value of a different set of attributes. This allows that different applications handle different context elements for the same entity, each containing only those attributes of that entity relevant to the corresponding application. It will also allow representing updates on set of attributes linked to a given entity: each of these updates can actually take the form of a context element and contain only the value of those attributes that have changed.

- An **event** is an occurrence within a particular system or domain; it is something that has happened, or is contemplated as having happened in that domain. Events typically lead to creation of some data or context element describing or representing the events, thus allowing them to processed. As an example, a sensor device may be measuring the temperature and pressure of a given boiler, sending a context element every five minutes associated to that entity (the boiler) that includes the value of these to attributes (temperature and pressure). The creation and sending of the context element is an event, i.e., what has occurred. Since the data/context elements that are generated linked to an event are the way events get visible in a computing system, it is common to refer to these data/context elements simply as "events".
- A **data event** refers to an event leading to creation of a data element.
- A **context event** refers to an event leading to creation of a context element.
- An **event object** is used to mean a programming entity that represents an event in a computing system [EPIA] like event-aware GEs. Event objects allow to perform operations on event, also known as **event processing**. Event objects are defined as a data element (or a context element) representing an event to which a number of standard event object properties (similar to a header) are associated internally. These standard event object properties support certain event processing functions.

23 Semantic Support Open RESTful API Specification (PRELIMINARY)

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

23.1 Introduction to the Ontology Registry API

Please check the FI-WARE Open Specifications Legal Notice to understand the rights to use FI-WARE Open Specifications.

23.1.1 Ontology Registry API Core

The Ontology Registry API is a RESTful, resource-oriented API accessed via HTTP that uses XML-based representations for information interchange. This API provides the means to effectively manage ontolgies and their related metadata, enhancing ontology development lifecycle. This API is part of the set of APIs provided by the Semmantic Application Support GE.

23.1.2 Intended Audience

This specification is intended for ontology practitioners or ontology engineering application developers. For the former, this document provides a full specification of how to interoperate with Ontology Registries that implements Ontology Registry API. To use this information, the reader should firstly have a general understanding of the Semmantic Application Support GE.

23.1.3 API Change History

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

Revision Date	Changes Summary
Apr 24, 2012	Initial API version

23.1.4 How to Read this Document

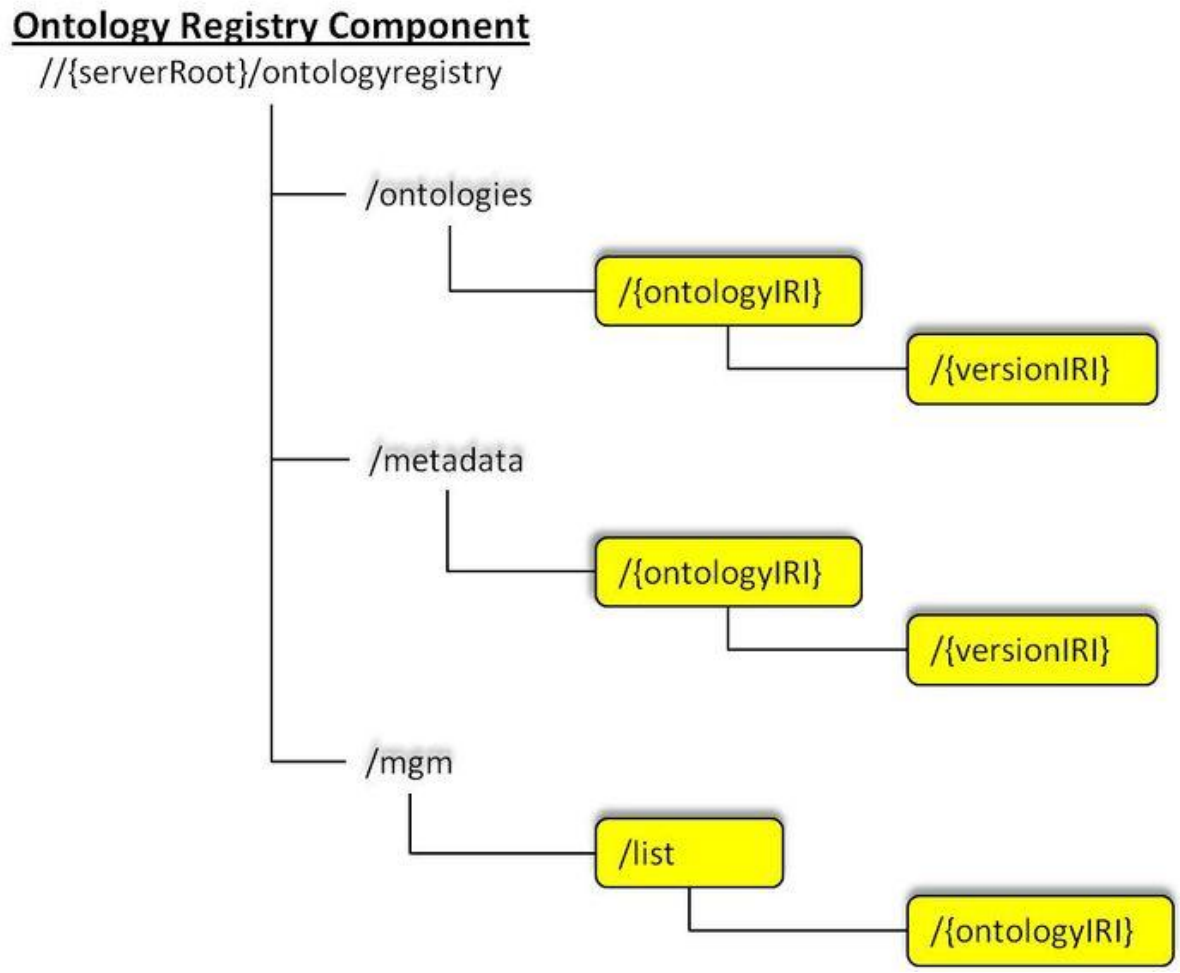
All FI-WARE RESTful API specifications will follow the same list of conventions and will support certain common aspects. Please check Common aspects in FI-WARE Open Restful API Specifications. For a description of some terms used along this document, see Semantic Application Support GE Architecture.

23.1.5 Aditional Resources

For more details about the Semantic Web Application Support GE that this API is based upon, please refer to Semantic Web Application Support documentation. Related documents, including an Architectural Description, are available at the same site.

23.2 General Ontology Registry API Information

23.2.1 Resources Summary



23.2.2 Representation Format

The Ontology Registry API supports XML based representation formats.

23.2.3 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.

23.2.4 Resource Identification

This section must explain which would be the resource identification used by the API in order to identify unambiguously the resource. For HTTP transport, this is made using the mechanisms described by HTTP protocol specification as defined by IETF RFC-2616.

23.2.5 Links and References

No additional links or references are provided in this version.

23.2.6 Limits

Limits section and operations will be provided in further FI-WARE releases.

23.2.7 Versions

Versions section and operations will be provided in further FI-WARE releases.

23.2.8 Extensions

Extensions section and operations will be provided (if needed) in further FI-WARE releases.

23.2.9 Faults

Faults section and operations will be provided (if needed) in further FI-WARE releases.

23.3 API Operations

The following section provides the detail for each RESTful operation giving the expected input and output for each URI.

23.3.1 Ontology Operations

23.3.1.1 *GetOntologyVersion*

Verb	URI	Description
GET	/ontologies/{ontologyIRI}/{versionIRI}	Retrieves the ontology field identified by a given ontology IRI and version IRI

Response codes:

- HTTP/1.1 200 - If the ontology is succesfully retrieved from the registry
- HTTP/1.1 404 - If there is no ontology in the registry identified by given ontology IRI and version IRI
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
GET                 /ontology-registry-service/webresources/ontology-
registry/ontologies/merm.owl/7 HTTP/1.1

Host: localhost:8080

Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

Response example:

```
HTTP/1.1 200 OK
Content-Type: application/xml

<?xml version="1.0"?>

 <!DOCTYPE rdf:RDF [
    <!ENTITY sioc "http://rdfs.org/sioc/ns#" >
    <!ENTITY dcterms "http://purl.org/dc/terms/" >
    <!ENTITY foaf "http://xmlns.com/foaf/0.1/" >
    <!ENTITY sawsdl "http://www.w3.org/ns/sawsdl#" >
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY swrl "http://www.w3.org/2003/11/swrl#" >
    <!ENTITY owl2 "http://www.w3.org/2006/12/owl2#" >
    <!ENTITY dc "http://purl.org/dc/elements/1.1/" >
    <!ENTITY posm "http://www.wsmo.org/ns/posm/0.1#" >
    <!ENTITY swrlb "http://www.w3.org/2003/11/swrlb#" >
    <!ENTITY swrlx "http://www.w3.org/2003/11/swrlx#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY so "http://knoesis.wright.edu/ssw/ont/sensor-
observation.owl#" >
 ]>

 <rdf:RDF xmlns="http://www.tatoo-fp7.eu/tatooweb/merm#"
     xml:base="http://www.tatoo-fp7.eu/tatooweb/merm"
     xmlns:dc="http://purl.org/dc/elements/1.1/"
     xmlns:foaf="http://xmlns.com/foaf/0.1/"
     xmlns:so="http://knoesis.wright.edu/ssw/ont/sensor-
observation.owl#"
     xmlns:swrlx="http://www.w3.org/2003/11/swrlx#"
     xmlns:sawsdl="http://www.w3.org/ns/sawsdl#"
     xmlns:owl2="http://www.w3.org/2006/12/owl2#"
     xmlns:dcterms="http://purl.org/dc/terms/"
     xmlns:sioc="http://rdfs.org/sioc/ns#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:swrl="http://www.w3.org/2003/11/swrl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:posm="http://www.wsmo.org/ns/posm/0.1#">

    <owl:Ontology rdf:about="http://www.tatoo-fp7.eu/tatooweb/merm">
```

```
        <owl:imports rdf:resource="http://www.tatoo-
fp7.eu/tatooweb/foaf_pruned"/>
        <owl:imports rdf:resource="http://www.tatoo-
fp7.eu/tatooweb/V1_0/sioc_pruned"/>
    </owl:Ontology>

    <!--

////////////////////////////////////////////////////////////////////////
//////////////////
    //
    // Annotation properties
    //

////////////////////////////////////////////////////////////////////////
//////////////////
     -->

    <owl:DatatypeProperty rdf:about="http://www.tatoo-
fp7.eu/tatooweb/merm#hasEvaluationMetric">
        <rdfs:label xml:lang="en">has Evaluation Metric</rdfs:label>
        <rdfs:range rdf:resource="&xsd;String"/>
    </owl:DatatypeProperty>
    <owl:ObjectProperty rdf:about="http://www.tatoo-
fp7.eu/tatooweb/merm#dateEvaluated">
        <rdfs:label xml:lang="en">date evaluated</rdfs:label>
        <rdfs:domain rdf:resource="http://www.tatoo-
fp7.eu/tatooweb/merm#Evaluation"/>
        <rdfs:subPropertyOf rdf:resource="http://www.tatoo-
fp7.eu/tatooweb/merm#dateProvided"/>
    </owl:ObjectProperty>

    <owl:ObjectProperty rdf:about="&dc;publisher">
        <rdfs:label xml:lang="en">publisher</rdfs:label>
        <rdfs:comment>The person who publishes the resource in real
world</rdfs:comment>
        <rdfs:domain rdf:resource="http://www.tatoo-
fp7.eu/tatooweb/merm#Resource"/>
        <rdfs:range rdf:resource="&foaf;Agent"/>
    </owl:ObjectProperty>
 ...
```

23.3.1.2 *GetOntology*

Verb	URI	Description
GET	/ontologies/{ontologyIRI}	Retrieves the last version of the ontology identified by a given ontology IRI

Response codes:

- HTTP/1.1 200 - If the ontology is succesfully retrieved from the registry

- HTTP/1.1 404 - If there is no ontology in the registry identified by given ontology IRI
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
GET                 /ontology-registry-service/webresources/ontology-
registry/ontologies/merm.owl HTTP/1.1

Host: localhost:8080

Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

Response example:

```
HTTP/1.1 200 OK
Content-Type: application/xml

<?xml version="1.0"?>

 <!DOCTYPE rdf:RDF [
    <!ENTITY sioc "http://rdfs.org/sioc/ns#" >
    <!ENTITY dcterms "http://purl.org/dc/terms/" >
    <!ENTITY foaf "http://xmlns.com/foaf/0.1/" >
    <!ENTITY sawsdl "http://www.w3.org/ns/sawsdl#" >
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY swrl "http://www.w3.org/2003/11/swrl#" >
    <!ENTITY owl2 "http://www.w3.org/2006/12/owl2#" >
    <!ENTITY dc "http://purl.org/dc/elements/1.1/" >
    <!ENTITY posm "http://www.wsmo.org/ns/posm/0.1#" >
    <!ENTITY swrlb "http://www.w3.org/2003/11/swrlb#" >
    <!ENTITY swrlx "http://www.w3.org/2003/11/swrlx#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY so "http://knoesis.wright.edu/ssw/ont/sensor-
observation.owl#" >
 ]>

 <rdf:RDF xmlns="http://www.tatoo-fp7.eu/tatooweb/merm#"
     xml:base="http://www.tatoo-fp7.eu/tatooweb/merm"
     xmlns:dc="http://purl.org/dc/elements/1.1/"
     xmlns:foaf="http://xmlns.com/foaf/0.1/"
     xmlns:so="http://knoesis.wright.edu/ssw/ont/sensor-
observation.owl#"
     xmlns:swrlx="http://www.w3.org/2003/11/swrlx#"
     xmlns:sawsdl="http://www.w3.org/ns/sawsdl#"
     xmlns:owl2="http://www.w3.org/2006/12/owl2#"
     xmlns:dcterms="http://purl.org/dc/terms/"
     xmlns:sioc="http://rdfs.org/sioc/ns#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:swrl="http://www.w3.org/2003/11/swrl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
```

```
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:posm="http://www.wsmo.org/ns/posm/0.1#">

    <owl:Ontology rdf:about="http://www.tatoo-fp7.eu/tatooweb/merm">
        <owl:imports rdf:resource="http://www.tatoo-
fp7.eu/tatooweb/foaf_pruned"/>
        <owl:imports rdf:resource="http://www.tatoo-
fp7.eu/tatooweb/V1_0/sioc_pruned"/>
    </owl:Ontology>

    <!--

////////////////////////////////////////////////////////////////////////
////////////////////
    //
    // Annotation properties
    //

////////////////////////////////////////////////////////////////////////
////////////////////
     -->

    <owl:DatatypeProperty rdf:about="http://www.tatoo-
fp7.eu/tatooweb/merm#hasEvaluationMetric">
        <rdfs:label xml:lang="en">has Evaluation Metric</rdfs:label>
        <rdfs:range rdf:resource="&xsd;String"/>
    </owl:DatatypeProperty>
    <owl:ObjectProperty rdf:about="http://www.tatoo-
fp7.eu/tatooweb/merm#dateEvaluated">
        <rdfs:label xml:lang="en">date evaluated</rdfs:label>
        <rdfs:domain rdf:resource="http://www.tatoo-
fp7.eu/tatooweb/merm#Evaluation"/>
        <rdfs:subPropertyOf rdf:resource="http://www.tatoo-
fp7.eu/tatooweb/merm#dateProvided"/>
    </owl:ObjectProperty>

    <owl:ObjectProperty rdf:about="&dc;publisher">
        <rdfs:label xml:lang="en">publisher</rdfs:label>
        <rdfs:comment>The person who publishes the resource in real
world</rdfs:comment>
        <rdfs:domain rdf:resource="http://www.tatoo-
fp7.eu/tatooweb/merm#Resource"/>
        <rdfs:range rdf:resource="&foaf;Agent"/>
    </owl:ObjectProperty>
 ...
```

23.3.1.3 *DeleteOntology*

Verb	URI	Description
DELETE	/ontologies/{ontologyIRI}	Removes from the registry the last version of the ontology identified by a given ontology IRI

Response codes:

- HTTP/1.1 200 - If the ontology is succesfully removed from the registry
- HTTP/1.1 404 - If there is no ontology in the registry identified by given ontology IRI
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
DELETE /ontology-registry-service/webresources/ontology-registry/ontologies/owl_time_pruned.owl HTTP/1.1
Accept: application/xml
Host: localhost:8080
```

Response example:

```
HTTP/1.1 200 OK
```

23.3.1.4 *DeleteOntologyVersion*

Verb	URI	Description
DELETE	/ontologies/{ontologyIRI}/{versionIRI}	Removes from the registry the ontology identified by a given ontology IRI and version IRI

Response codes:

- HTTP/1.1 200 - If the ontology is succesfully removed from the registry
- HTTP/1.1 404 - If there is no ontology in the registry identified by given ontology IRI and version IRI
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
DELETE /ontology-registry-service/webresources/ontology-registry/ontologies/owl_time_pruned.owl HTTP/1.1
Accept: application/xml
Host: localhost:8080
```

Response example:

```
HTTP/1.1 200 OK
```

23.3.1.5 *UploadOntology*

Verb	URI	Description
POST	/ontologies/{ontologyIRI}	Upload an ontology file to the repository. Uploaded file is labeled as last ontology version

Response codes:

- HTTP/1.1 200 - If the ontology is succesfully stored at the registry
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
POST /ontology-registry-service/webresources/ontology-
registry/ontologies/sioc_pruned.owl?create=true HTTP/1.1
Content-Type: multipart/form-data;
boundary=Boundary_30_4446747_1334759773635
Accept: application/xml
MIME-Version: 1.0
Host: localhost:8080
--Boundary_30_4446747_1334759773635
Content-Type: application/octet-stream
Content-Disposition: form-data; filename="sioc_pruned.owl";
modification-date="Mon, 28 Nov 2011 14:14:52 GMT"; size=8268;
name="sioc_pruned.owl"
<?xml version="1.0"?>

 <!DOCTYPE rdf:RDF [
    <!ENTITY sioc "http://rdfs.org/sioc/ns#" >
    <!ENTITY terms "http://purl.org/dc/terms/" >
    <!ENTITY foaf "http://xmlns.com/foaf/0.1/" >
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY swrl "http://www.w3.org/2003/11/swrl#" >
    <!ENTITY owl2 "http://www.w3.org/2006/12/owl2#" >
    <!ENTITY swrlb "http://www.w3.org/2003/11/swrlb#" >
    <!ENTITY swrlx "http://www.w3.org/2003/11/swrlx#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
 ]>

 <rdf:RDF xmlns="http://www.tatoo-fp7.eu/tatooweb/ns_module1#"
     xml:base="http://www.tatoo-fp7.eu/tatooweb/ns_module1"
     xmlns:foaf="http://xmlns.com/foaf/0.1/"
     xmlns:swrlx="http://www.w3.org/2003/11/swrlx#"
     xmlns:terms="http://purl.org/dc/terms/"
     xmlns:owl2="http://www.w3.org/2006/12/owl2#"
     xmlns:sioc="http://rdfs.org/sioc/ns#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:swrl="http://www.w3.org/2003/11/swrl#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
```

```
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <owl:Ontology rdf:about="http://www.tatoo-
fp7.eu/tatooweb/sioc_pruned"/>

    <!--

////////////////////////////////////////////////////////////////////
//////////////////
    //
    // Annotation properties
    //

////////////////////////////////////////////////////////////////////
//////////////////
     -->
 ...
```

Response example:

```
HTTP/1.1 200 OK
```

23.3.1.6 *UploadOntologyVersion*

Verb	URI	Description
POST	/ontologies/{ontologyIRI}/{versionIRI}	Upload an ontology file to the repository. Uploaded file is labeled with given ontology IRI and version IRI

Response codes:

- HTTP/1.1 200 - If the ontology is succesfully stored at the registry
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
POST /ontology-registry-service/webresources/ontology-
registry/ontologies/sioc_pruned.owl/1.0?create=true HTTP/1.1
Content-Type: multipart/form-data;
boundary=Boundary_30_4446747_1334759773636
Accept: application/xml
MIME-Version: 1.0
Host: localhost:8080
--Boundary_30_4446747_1334759773636
Content-Type: application/octet-stream
Content-Disposition: form-data; filename="sioc_pruned.owl";
modification-date="Mon, 28 Nov 2011 14:14:52 GMT"; size=8268;
name="sioc_pruned.owl"
<?xml version="1.0"?>

 <!DOCTYPE rdf:RDF [
    <!ENTITY sioc "http://rdfs.org/sioc/ns#" >
    <!ENTITY terms "http://purl.org/dc/terms/" >
    <!ENTITY foaf "http://xmlns.com/foaf/0.1/" >
```

```
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY swrl "http://www.w3.org/2003/11/swrl#" >
    <!ENTITY owl2 "http://www.w3.org/2006/12/owl2#" >
    <!ENTITY swrlb "http://www.w3.org/2003/11/swrlb#" >
    <!ENTITY swrlx "http://www.w3.org/2003/11/swrlx#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
 ]>

 <rdf:RDF xmlns="http://www.tatoo-fp7.eu/tatooweb/ns_module1#"
     xml:base="http://www.tatoo-fp7.eu/tatooweb/ns_module1"
     xmlns:foaf="http://xmlns.com/foaf/0.1/"
     xmlns:swrlx="http://www.w3.org/2003/11/swrlx#"
     xmlns:terms="http://purl.org/dc/terms/"
     xmlns:owl2="http://www.w3.org/2006/12/owl2#"
     xmlns:sioc="http://rdfs.org/sioc/ns#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:swrl="http://www.w3.org/2003/11/swrl#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <owl:Ontology rdf:about="http://www.tatoo-
fp7.eu/tatooweb/sioc_pruned"/>

    <!--

///////////////////////////////////////////////////////////////////////
//////////////////
    //
    // Annotation properties
    //

///////////////////////////////////////////////////////////////////////
//////////////////
     -->
 ...
```

Response example:

```
HTTP/1.1 200 OK
```

23.3.2 Management Operations

23.3.2.1 *GetOntologyList*

Verb	URI	Description
GET	/mgm/list	Retrieves a list of the ontologies and their versions contained within the registry

Response codes:

- HTTP/1.1 200 - If the ontology list is succesfully generated and retrieved
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
GET /ontology-registry-service/webresources/ontology-registry/mgm/list HTTP/1.1
Host: localhost:8080
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

Response example:

```
HTTP/1.1 200 OK
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
   <ontologies>
      <ontology name="AITAlignments.owl">
         <version name="1"/>
      </ontology>
      <ontology name="AIT_ClimateTwins_Domain.owl">
         <version name="2"/>
      </ontology>
      <ontology name="ICD_neoplasms_pruned.owl">
         <version name="6"/>
         <version name="15"/>
         <version name="24"/>
      </ontology>
...
```

23.3.2.2 *GetOntologyVersions*

Verb	URI	Description
GET	/mgm/list/{ontologyIRI}	Retrieves a list of versions of ontology identified by given ontology IRI contained within the registry

Response codes:

- HTTP/1.1 200 - If the ontology is succesfully stored at the registry
- HTTP/1.1 404 - If there is no ontology identified by given ontology IRI.
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
GET /ontology-registry-service/webresources/ontology-registry/mgm/list/merm.owl HTTP/1.1
Host: localhost:8080
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

Response example:

```
HTTP/1.1 200 OK
Content-Type: application/xml
<?xml version="1.0" encoding="UTF-8"?>
<ontology name="merm.owl">
   <version name="7"/>
   <version name="16"/>
   <version name="25"/>
</ontology>
```

23.3.3 Metadata Operations

23.3.3.1 *GetOntologyVersionMetadata*

Verb	URI	Description
GET	/metadata/{ontologyIRI}/{versionIRI}	Retrieves the metadata related with the ontology identified by a given ontology IRI and version IRI

Response codes:

- HTTP/1.1 200 - If the metadata is succesfully retrieved from the registry
- HTTP/1.1 404 - If there is no ontology in the registry identified by given ontology IRI and version IRI
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
GET /ontology-registry-service/webresources/ontology-
registry/metadata/bridge.owl/3 HTTP/1.1
Host: localhost:8080
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

Response example:

```
HTTP/1.1 200 OK
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
       xmlns="http://omv.ontoware.org/2005/05/ontology#"
       xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
       xmlns:owl="http://www.w3.org/2002/07/owl#"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:swrl="http://www.w3.org/2003/11/swrl#"
       xmlns:swrlx="http://www.w3.org/2003/11/swrlx#"
       xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
       xmlns:owl2="http://www.w3.org/2006/12/owl2#"
       xmlns:bridge="http://localhost:8080/ontology-registry-
service/webresources/ontology-registry/ontologies/bridge.owl/3#"
       xmlns:geonames_pruned="http://localhost:8080/ontology-
registry-service/webresources/ontology-
registry/ontologies/geonames_pruned.owl/5#">

<rdf:Description rdf:about="http://eu.atosresearch.ontology-
registry/instance">
       <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#Ontology"/>
</rdf:Description>

<rdf:Description rdf:about="http://localhost:8080/ontology-registry-
service/webresources/ontology-
registry/ontologies/bridge.owl/3#metadata">
       <rdf:type
rdf:resource="http://omv.ontoware.org/2005/05/ontology#Ontology"/>
       <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#NamedIndividual"/>
       <usedOntologyEngineeringTool
rdf:resource="http://omv.ontoware.org/2005/05/ontology#NeOn-
Toolkit"/>
       <hasCreator
rdf:resource="http://omv.ontoware.org/2005/05/ontology#individual133
3110758440"/>
       <hasOntologyLanguage
rdf:resource="http://omv.ontoware.org/2005/05/ontology#OWL"/>
</rdf:Description>

</rdf:RDF>
```

23.3.3.2 *GetOntologyMetadata*

Verb	URI	Description
GET	/metadata/{ontologyIRI}	Retrieves the metadata related to the last version of the ontology identified by a given ontology IRI

Response codes:

- HTTP/1.1 200 - If the metadata is succesfully retrieved from the registry
- HTTP/1.1 404 - If there is no ontology in the registry identified by given ontology IRI
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
GET /ontology-registry-service/webresources/ontology-
registry/metadata/bridge.owl HTTP/1.1

Host: localhost:8080

Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

Response example:

```
HTTP/1.1 200 OK
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
       xmlns="http://omv.ontoware.org/2005/05/ontology#"
       xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
       xmlns:owl="http://www.w3.org/2002/07/owl#"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:swrl="http://www.w3.org/2003/11/swrl#"
       xmlns:swrlx="http://www.w3.org/2003/11/swrlx#"
       xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
       xmlns:owl2="http://www.w3.org/2006/12/owl2#"
       xmlns:bridge="http://localhost:8080/ontology-registry-
service/webresources/ontology-registry/ontologies/bridge.owl/3#"
       xmlns:geonames_pruned="http://localhost:8080/ontology-
registry-service/webresources/ontology-
registry/ontologies/geonames_pruned.owl/5#">

<rdf:Description rdf:about="http://eu.atosresearch.ontology-
registry/instance">
       <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#Ontology"/>
</rdf:Description>

<rdf:Description rdf:about="http://localhost:8080/ontology-registry-
service/webresources/ontology-
registry/ontologies/bridge.owl/3#metadata">
```

```
       <rdf:type
rdf:resource="http://omv.ontoware.org/2005/05/ontology#Ontology"/>
       <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#NamedIndividual"/>
       <usedOntologyEngineeringTool
rdf:resource="http://omv.ontoware.org/2005/05/ontology#NeOn-
Toolkit"/>
       <hasCreator
rdf:resource="http://omv.ontoware.org/2005/05/ontology#individual133
3110758440"/>
       <hasOntologyLanguage
rdf:resource="http://omv.ontoware.org/2005/05/ontology#OWL"/>
</rdf:Description>

</rdf:RDF>
```

23.3.3.3 *DeleteOntologyMetadata*

Verb	URI	Description
DELETE	/metadata/{ontologyIRI}	Removes from the registry the metadata related to the last version of the ontology identified by a given ontology IRI

Response codes:

- HTTP/1.1 200 - If the metadata is succesfully removed from the registry
- HTTP/1.1 404 - If there is no ontology in the registry identified by given ontology IRI
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
DELETE              /ontology-registry-service/webresources/ontology-
registry/metadata/bridge.owl HTTP/1.1
Accept: application/xml
Host: localhost:8080
```

Response example:

```
HTTP/1.1 200 OK
```

23.3.3.4 *DeleteOntologyVersionMetadata*

Verb	URI	Description
DELETE	/metadata/{ontologyIRI}/{versionIRI}	Removes from the registry the metadata related to the ontology identified by a given ontology IRI and version IRI

Response codes:

- HTTP/1.1 200 - If the metadata is succesfully removed from the registry
- HTTP/1.1 404 - If there is no ontology in the registry identified by given ontology IRI and version IRI
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
DELETE /ontology-registry-service/webresources/ontology-registry/metadata/bridge.owl HTTP/1.1
Accept: application/xml
Host: localhost:8080
```

Response example:

```
HTTP/1.1 200 OK
```

23.3.3.5 *UploadOntologyMetadata*

Verb	URI	Description
POST	/metadata/{ontologyIRI}	Upload an metadata file to the repository. This file should be a RDF/XML serialization of a valid instance of OMV Ontology class.

Response codes:

- HTTP/1.1 200 - If the metadata is succesfully stored at the registry
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
POST /ontology-registry-service/webresources/ontology-registry/metadata/bridge.owl?create=true HTTP/1.1
Content-Type: multipart/form-data;
boundary=Boundary_30_4446747_1334759773635
Accept: application/xml
MIME-Version: 1.0
Host: localhost:8080
--Boundary_30_4446747_1334759773635
Content-Type: application/octet-stream
Content-Disposition: form-data; filename="bridge_metadata.owl";
modification-date="Mon, 28 Nov 2011 14:14:52 GMT"; size=8268;
name="bridge_metadata.owl"
```

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
       xmlns="http://omv.ontoware.org/2005/05/ontology#"
       xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
       xmlns:owl="http://www.w3.org/2002/07/owl#"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:swrl="http://www.w3.org/2003/11/swrl#"
       xmlns:swrlx="http://www.w3.org/2003/11/swrlx#"
       xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
       xmlns:owl2="http://www.w3.org/2006/12/owl2#"
       xmlns:bridge="http://localhost:8080/ontology-registry-
service/webresources/ontology-registry/ontologies/bridge.owl/3#"
       xmlns:geonames_pruned="http://localhost:8080/ontology-
registry-service/webresources/ontology-
registry/ontologies/geonames_pruned.owl/5#">

<rdf:Description rdf:about="http://eu.atosresearch.ontology-
registry/instance">
       <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#Ontology"/>
</rdf:Description>

<rdf:Description rdf:about="http://localhost:8080/ontology-registry-
service/webresources/ontology-
registry/ontologies/bridge.owl/3#metadata">
       <rdf:type
rdf:resource="http://omv.ontoware.org/2005/05/ontology#Ontology"/>
       <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#NamedIndividual"/>
       <usedOntologyEngineeringTool
rdf:resource="http://omv.ontoware.org/2005/05/ontology#NeOn-
Toolkit"/>
       <hasCreator
rdf:resource="http://omv.ontoware.org/2005/05/ontology#individual133
3110758440"/>
       <hasOntologyLanguage
rdf:resource="http://omv.ontoware.org/2005/05/ontology#OWL"/>
</rdf:Description>

</rdf:RDF>
```

Response example:

```
HTTP/1.1 200 OK
```

23.3.3.6 *UploadOntologyVersionMetadata*

Verb	URI	Description
POST	/metadata/{ontologyIRI}/{versionIRI}	Upload an metadata file to the repository. The file should be an RDF/XML serialization containing an instance of OMV Ontology class. Uploaded file will be related to the ontology identified by given ontology IRI and version IRI

Response codes:

- HTTP/1.1 200 - If the metadata is succesfully stored at the registry
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
POST /ontology-registry-service/webresources/ontology-
registry/metadata/bridge.owl/3?create=true HTTP/1.1
Content-Type: multipart/form-data;
boundary=Boundary_30_4446747_1334759773636
Accept: application/xml
MIME-Version: 1.0
Host: localhost:8080
--Boundary_30_4446747_1334759773636
Content-Type: application/octet-stream
Content-Disposition: form-data; filename="brdige_metadata.owl";
modification-date="Mon, 28 Nov 2011 14:14:52 GMT"; size=8268;
name="brdige_metadata.owl"
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
       xmlns="http://omv.ontoware.org/2005/05/ontology#"
       xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
       xmlns:owl="http://www.w3.org/2002/07/owl#"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:swrl="http://www.w3.org/2003/11/swrl#"
       xmlns:swrlx="http://www.w3.org/2003/11/swrlx#"
       xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
       xmlns:owl2="http://www.w3.org/2006/12/owl2#"
       xmlns:bridge="http://localhost:8080/ontology-registry-
service/webresources/ontology-registry/ontologies/bridge.owl/3#"
       xmlns:geonames_pruned="http://localhost:8080/ontology-
registry-service/webresources/ontology-
registry/ontologies/geonames_pruned.owl/5#">

<rdf:Description rdf:about="http://eu.atosresearch.ontology-
registry/instance">
       <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#Ontology"/>
</rdf:Description>
```

```
<rdf:Description rdf:about="http://localhost:8080/ontology-registry-
service/webresources/ontology-
registry/ontologies/bridge.owl/3#metadata">
       <rdf:type
rdf:resource="http://omv.ontoware.org/2005/05/ontology#Ontology"/>
       <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#NamedIndividual"/>
       <usedOntologyEngineeringTool
rdf:resource="http://omv.ontoware.org/2005/05/ontology#NeOn-
Toolkit"/>
       <hasCreator
rdf:resource="http://omv.ontoware.org/2005/05/ontology#individual133
3110758440"/>
       <hasOntologyLanguage
rdf:resource="http://omv.ontoware.org/2005/05/ontology#OWL"/>
</rdf:Description>

</rdf:RDF>
```

Response example:

```
HTTP/1.1 200 OK
```

24 FIWARE ArchitectureDescription Data SemanticSupport OMV Open Specification (DRAFT)

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

In order to provide advanced ontology management functionalities, ontologies should be annotated with extended metadata. In order to do so, the selection of a suitable ontology metadata format is needed. In this case, the Ontology Metadata Vocabulary (OMV) has been selected. Some of its key features are:

- OWL-2 ontology developed following NeOn Methodology by consortium members.
- Designed to meet NeOn Methodology reusability use case requirements.
- Extensible, reusable, accessible and interoperable.

OMV describes some metadata regarding ontologies that should be provided by users while loading ontologies into the GE. This metadata include information about ontology developers, ontology language, ontologies imported by the ontology, etc. OMV specification will be included in this section in further releases. In the meantime, a detailed OMV description can be found at OMV Documentation

25 FI-WARE_Open_Specifications_Legal_Notice

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

25.1.1.1 *General Information*

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

25.1.1.2 *Use Of Specification - Terms, Conditions & Notices*

The material in this specification details a FI-WARE Generic Enabler Specification (hereinafter "Specification") in accordance with the terms, conditions and notices set forth below. 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.

25.1.1.3 *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 license, royalty free (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 are an implementation of this Specification.

25.1.1.4 *Patent Information*

The FI-WARE Project Partners shall not be responsible for identifying patents for which a license may be required by 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.

25.1.1.5 *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 owners.

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 owners.

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

25.1.1.6 *Disclaimer Of Warranty*

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE FI-WARE PARTNERS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, 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.

IN NO EVENT SHALL THE FI-WARE PARTNERS BE LIABLE FOR ERRORS CONTAINED HEREIN 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 MATERIAL, 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 license granted to you to use this Specification.

25.1.1.7 *Trademarks*

You shall not use any trademark, marks or trade names (collectively, "Marks") of the FI-WARE Partners or the FI-WARE project without prior written consent.

25.1.1.8 *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.

26 Open Specifications Interim Legal Notice

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

26.1.1.1 *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.

26.1.1.2 *Use Of Specification - Terms, Conditions & Notices*

The material in this specification details a FI-WARE Generic Enabler Specification (hereinafter "Specification") in accordance with the terms, conditions and notices set forth below. 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.

26.1.1.3 *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 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 are an implementation of this Specification, and to use, copy, and distribute this Specification as provided under applicable law.

26.1.1.4 *Patent License*

"Specification Essential Patents" shall mean patents and patent applications, which are necessarily infringed by an implementation of the Specification and which are owned by any of the FI-WARE Project Partners. "Necessarily infringed" shall mean that no commercially reasonable alternative exists to avoid infringement.

Each of the FI-WARE Project Partners, jointly or solely, hereby agrees to grant you, on royalty-free and otherwise 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, and import software implementations utilizing the Specification.

The FI-WARE Project Partners shall not be responsible for identifying patents for which a license may be required by 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.

26.1.1.5 *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 owners.

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

26.1.1.6 *Disclaimer Of Warranty*

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE FI-WARE PARTNERS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, 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.

IN NO EVENT SHALL THE FI-WARE PARTNERS BE LIABLE FOR ERRORS CONTAINED HEREIN 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 MATERIAL, 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 license granted to you to use this Specification.

26.1.1.7 *Trademarks*

You shall not use any trademark, marks or trade names (collectively, "Marks") of the FI-WARE Project Partners or the FI-WARE project without prior written consent.

26.1.1.8 *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.