

Private Public Partnership Project (PPP)

Large-scale Integrated Project (IP)



D.6.1.3: FI-WARE GE Open Specification

Project acronym: FI-WARE

Project full title: Future Internet Core Platform

Contract No.: 285248

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

Project Document Number: ICT-2011-FI-285248-WP6-D.6.1.3

Project Document Date: 2014-08-19

Deliverable Type and Security: Public

Author: FI-WARE Consortium

Contributors: FI-WARE Consortium

1.1 Executive Summary

This document describes the Generic Enablers in the Data/Context Management Services chapter, their basic functionality and their interaction. These Generic Enablers form the core business framework of the FI-WARE platform by supporting the business functionality for commercializing services.

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

This document includes a first block of GEs Open Specifications to be delivered by Month 36. A subsequent version of this deliverable will be published to include all the GEs.

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 Generic Enablers (GEs) able to gather, publish, exchange, process and analyze massive data in a fast and efficient way.

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

The Data/Context Management FI-WARE chapter aims at providing outperforming and platform-like GEs that will ease development and the provisioning 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 GEs coming from the Applications and Services Delivery Framework Chapters, application providers will be able to build innovative business models such as those of the companies mentioned above and beyond.

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

- Generate, 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, leading to changes in context also treatable as events).
- Processing large amounts of context information in an aggregated way, using BigData Map&Reduce techniques, in order to generate new knowledge.
- 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 as location information, presence, user or terminal profile, etc., in a standard 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 enables the usage of "Data Element" structures to represent "Context Elements" and "Events".

1.4.1 Architecture Overview

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

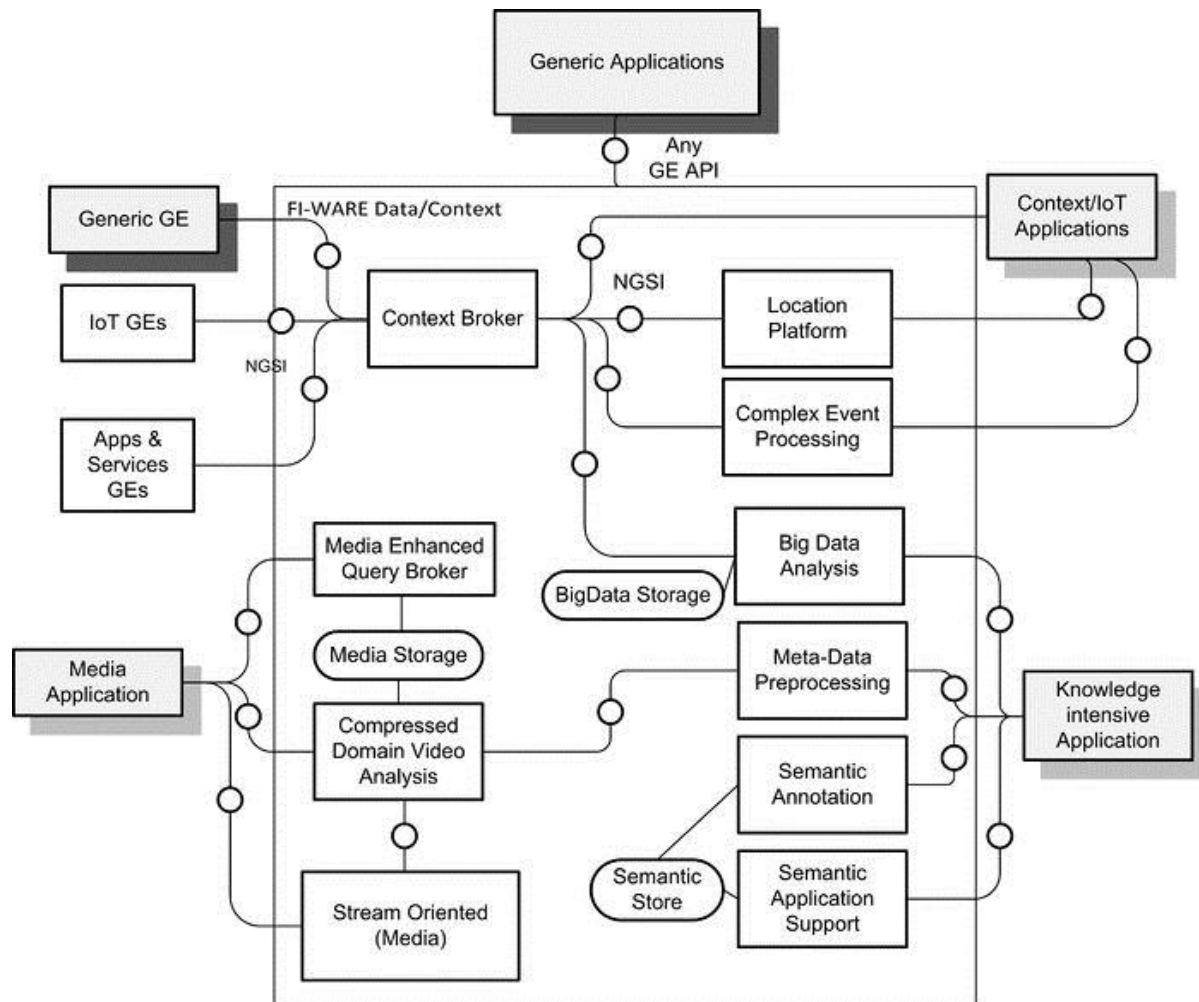


Figure 1: Data Architecture Overview

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

<http://wiki.fi-ware.org>

[Data/Context Management Architecture](#)

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.org/>

The following resources were used to generate this document:

D.6.1.3 FI-WARE GE Open Specifications front page

[FIWARE.OpenSpecification.Data.BigData](#)

[BigData Analysis Open RESTful API Specification](#)

[FIWARE.OpenSpecification.Data.ContextBroker](#)

[FI-WARE_NGSI-9_Open_RESTful_API_Specification](#)

[FI-WARE_NGSI-10_Open_RESTful_API_Specification](#)

[ContextML API](#)

[CQL API](#)

[FIWARE.OpenSpecification.Data.CEP](#)

[Complex Event Processing Open RESTful API Specification](#)

[FIWARE.OpenSpecification.Data.SemanticAnnotation](#)

[Semantic Annotation Open RESTful API Specification](#)

[FIWARE.OpenSpecification.Data.SemanticSupport](#)

[Semantic Support Open RESTful API Specification](#)

[FIWARE.ArchitectureDescription.Data.SemanticSupport.OMV_Open_Specification](#)

[FIWARE.OpenSpecification.Data.StreamOriented](#)

[StreamOriented Open API Specification](#)

[FIWARE.OpenSpecification.Data.SemanticContextExt](#)

[Publish/Subscribe Semantic Extension Open RESTful API Specification](#)

[FIWARE.OpenSpecification.Data.UnstructuredDataAnalysis](#)

[Unstructured Data Analysis Open RESTful API Specification](#)

[FIWARE.OpenSpecification.Data.Location](#)

[Location Server Open RESTful API Specification](#)

[FIWARE.OpenSpecification.Data.MetadataPreprocessing](#)

[Metadata Preprocessing Open RESTful API Specification](#)

[FIWARE.OpenSpecification.Data.CompressedDomainVideoAnalysis](#)

[Compressed Domain Video Analysis Open RESTful API Specification](#)

[FIWARE.OpenSpecification.Data.QueryBroker](#)

[Query Broker Open RESTful API Specification](#)

[FI-WARE Open Specification Legal Notice \(implicit patents license\)](#)

[FI-WARE Open Specification Legal Notice \(essential patents license\)](#)

1.6 Typographical Conventions

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

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

1.6.1 Links within this document

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

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

1.6.2 Figures

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

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

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

1.6.3 Sample software code

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

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

1.7 Acknowledgements

The following partners contributed to this deliverable: [TID](#), [IBM](#), [SIEMENS](#), [ATOS](#), [Thales](#), [FT](#), [TI](#), [URJC](#), [NAEVATEC](#).

1.8 Keyword list

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

1.9 Changes History

Release	Major changes description	Date	Editor
v1	First draft	2014-02-04	TID
v2	Draft for final review	2014-04-24	TID
v5	Draft with Wave 2 GEs	2014-05-12	TID
v6	Draft for final review	2014-06-20	TID
v7	Definitive review. Ready for delivery	2014-08-19	TID

1.10 Table of Content

1.1	Executive Summary.....	2
1.2	About This Document	3
1.3	Intended Audience.....	3
1.4	Chapter Context.....	3
1.5	Structure of this Document	5
1.6	Typographical Conventions.....	7
1.7	Acknowledgements.....	8
1.8	Keyword list.....	8
1.9	Changes History	9
1.10	Table of Content	9
2	FIWARE OpenSpecification Data BigData	16
2.1	Preface	16
2.2	Copyright.....	16
2.3	Legal Notice.....	16
2.4	Overview	16
2.5	Basic concepts.....	18
2.6	Architecture	24
2.7	Main Interactions.....	30

2.8	Basic Design Principles	32
2.9	References	33
2.10	Detailed Specifications.....	34
2.11	Re-utilised Technologies/Specifications	35
2.12	Terms and definitions	35
3	BigData Analysis Open RESTful API Specification	37
3.1	Introduction to the Big Data API	37
3.2	General Big Data GE API Information	38
3.3	API Operations	40
4	FIWARE OpenSpecification Data ContextBroker	56
4.1	Preface	56
4.2	Re-utilised Technologies/Specifications	76
4.3	Terms and definitions	77
5	FI-WARE NGSI-9 Open RESTful API Specification.....	79
5.1	Introduction to the FI-WARE NGSI-9 API	79
5.2	General NGSI-9 API information	81
6	FI-WARE NGSI-10 Open RESTful API Specification.....	89
6.1	Introduction to the FI-WARE NGSI 10 API.....	89
6.2	General NGSI 10 API information.....	90
6.3	References	97
7	ContextML API	99
7.1	Using ContextML to interact with the Publish/Subscribe GE	99
7.2	ContextML Basics	99
7.3	ContextML API	102
8	CQL API.....	110
8.1	ContextQL (CQL).....	110
8.2	CQL API.....	113
9	FIWARE OpenSpecification Data CEP.....	120
9.1	Preface	120
9.2	Copyright.....	120
9.3	Legal Notice.....	120

9.4	Overview	120
9.5	Target Usage	124
9.6	Basic Concepts	125
9.7	Basic Design Principles	130
9.8	References	130
9.9	Detailed Specifications.....	130
9.10	Re-utilised Technologies/Specifications	130
9.11	Terms and definitions	131
10	Complex Event Processing Open RESTful API Specification.....	133
10.1	Introduction to the CEP GE REST API	133
10.2	General CEP API Information	134
10.3	API Operations	135
11	FIWARE OpenSpecification Data SemanticAnnotation.....	145
11.1	Preface	145
11.2	Detailed Specifications.....	151
11.3	Re-utilised Technologies/Specifications	151
11.4	Terms and definitions	152
12	Semantic Annotation Open RESTful API Specification.....	154
13	FIWARE OpenSpecification Data SemanticSupport.....	158
13.1	Preface	158
13.2	Re-utilised Technologies/Specifications	177
13.3	Terms and definitions	177
14	Semantic Support Open RESTful API Specification	180
14.1	Introduction to the Ontology Registry API.....	180
14.2	General Ontology Registry API Information	181
14.3	API Operations	182
14.4	General Workspace Management API Information.....	207
14.5	API Operations	208
15	FIWARE ArchitectureDescription Data SemanticSupport OMV_Open_Specification	222
16	FIWARE OpenSpecification Data StreamOriented.....	223
16.1	Preface	223

16.2	Copyright.....	223
16.3	Legal Notice.....	223
16.4	Overview	223
16.5	Basic Concepts	224
16.6	Stream-oriented GE Architecture	226
16.7	Main Interactions.....	232
16.8	Basic Design Principles	238
16.9	Detailed Specifications.....	239
16.10	Re-utilised Technologies/Specifications	239
16.11	Terms and definitions	240
17	StreamOriented Open API Specification	243
17.2	API General Features	244
17.3	API Specification.....	246
18	FIWARE OpenSpecification Data SemanticContextExt	289
18.1	Preface	289
18.2	Copyright.....	289
18.3	Legal Notice.....	289
18.4	Overview	289
18.5	Basic Concepts	290
18.6	Main Interactions.....	291
18.7	Basic Design Principles	291
18.8	Detailed Specifications.....	293
18.9	Re-utilised Technologies/Specifications	293
18.10	Terms and definitions	293
19	Publish/Subscribe Semantic Extension Open RESTful API Specification	295
19.1	Introduction to the Context Broker (CB) Semantic Extension GE REST API.....	295
19.2	API Operations	296
20	FIWARE OpenSpecification Data UnstructuredDataAnalysis.....	299
20.1	Preface	299
20.2	Detailed Specifications.....	305
20.3	Re-utilised Technologies/Specifications	305

20.4	Terms and definitions	305
21	Unstructured Data Analysis Open RESTful API Specification	307
21.1	Introduction to the Unstructured Data Analysis API	307
21.2	General Ontology Registry API Information	308
21.3	API Operations	309
22	FIWARE OpenSpecification Data Location	315
22.1	Preface	315
22.2	References	339
22.3	Detailed Specifications.....	340
22.4	Re-utilised Technologies/Specifications	340
22.5	Terms and definitions	340
23	Location_Server_Open_RESTful_API_Specification	342
23.1	Dedicated API Introduction.....	342
23.2	Introduction to the Restful Network API for Terminal Location.....	342
23.3	General Location Server REST API Information	344
23.4	Data Types.....	346
23.5	API Operations	352
24	FIWARE OpenSpecification Data MetadataPreprocessing	381
24.1	Preface	381
24.2	Copyright.....	381
24.3	Legal Notice.....	381
24.4	Overview	381
24.5	Basic Concepts	383
24.6	Main Interactions.....	385
24.7	Basic Design Principles	387
24.8	References	387
24.9	Detailed Specifications.....	387
24.10	Re-utilised Technologies/Specifications	388
24.11	Terms and definitions	388
25	Metadata Preprocessing Open RESTful API Specification	390
25.1	Introduction to the Metadata Preprocessing GE API.....	390

25.2	General Metadata Preprocessing GE API Information	391
25.3	API Operations	394
26	FIWARE OpenSpecification Data CompressedDomainVideoAnalysis.....	409
26.1	Preface	409
26.2	Copyright.....	409
26.3	Legal Notice.....	409
26.4	Overview	409
26.5	Basic Concepts	410
26.6	Architecture	413
26.7	Main Interactions.....	417
26.8	Basic Design Principles	421
26.9	References	421
26.10	Detailed Specifications.....	422
26.11	Re-utilised Technologies/Specifications	422
26.12	Terms and definitions	423
27	Compressed Domain Video Analysis Open RESTful API Specification	425
27.1	Introduction to the Compressed Domain Video Analysis GE API	425
27.2	General Compressed Domain Video Analysis GE API Information	426
27.3	API Operations	429
28	FIWARE OpenSpecification Data QueryBroker	451
28.1	Preface	451
28.2	Copyright.....	451
28.3	Legal Notice.....	451
28.4	Overview	451
28.5	Basic Concepts	454
28.6	QueryBroker Architecture.....	461
28.7	Main Interactions.....	464
28.8	Design Principles	472
28.9	References	473
28.10	Detail Specifications.....	474
28.11	Re-utilised Technologies/Specifications	475

28.12	Terms and definitions	475
29	Query Broker Open RESTful API Specification	478
29.1	Introduction to the REST-Interface of the QueryBroker.....	478
29.2	General QueryBroker REST API Information.....	480
29.3	API Operations	482
30	FI-WARE Open Specification Legal Notice (implicit patents license).....	494
31	FI-WARE Open Specification Legal Notice (essential patents license).....	497

2 FIWARE OpenSpecification Data BigData

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

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

2.2 Copyright

Copyright © 2013 by [Telefonica I+D](#)

2.3 Legal Notice

Please check the following [Legal Notice](#) to understand the rights to use these specifications.

2.4 Overview

2.4.1 Big Data

Big Data makes reference to the processing of vast amounts of data, being this data either stored in advance (what is called Big Data Batch Processing), either received in real time (Big Data Streaming Processing). In both cases the goal is to get insights, revealing new information that was hidden in the original data. The result of this new data discovery is automatically added to the initial volume.

Latency is not important when processing batches, but the results must be ready in a reasonable time (hours or days). Nevertheless, the processing of real-time data requires the results to be ready

immediately, basically because this kind of data is not stored at all and the insights must be taken on the fly.

2.4.2 Big Data GE

The Big Data Generic Enabler (GE) provides computing services for the batch processing and almost real time processing in addition to a distributed storage service. The goal is to explore the real value of the big volumes of data.

The computing service of the Big Data GE allows creating and configuring a coordinated cluster of machines in a fast way by means a simple command line or REST API call. Additionally, the GE allows for selecting a subset from a wide range of processing and coordination pre-configured technologies (Hadoop, Hive, Oozie, HBase, Sqoop, PIG, etc.).

The storage service of the GE allows feeding the clusters with a huge variety of data without any special adapter or configuration. The technology used within this GE allows for high throughput due to data translations are not performed. To store data in the storage service of the GE is as easy as using a command line or REST API call.

Using any of the services (computing or storage) does not imply to use the other. This way, the data storage is independent from the data lifecycle associated to its processing.

2.4.3 Target audience

The focus of this GE is in the data (big data) storage and analysis, that is, developers will only have to deal with the problem they want to resolve 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. 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.

2.4.4 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, longitude) of the cell tower to determine the approximate location of a given subscriber or terminal. This information is then **stored** in the GE, creating a profile for the subscribers that identifies where they live and work. This information can then be joined with an **analysis** of the movements of mobile phones that can help to determine the time at which traffic jams are likely to happen in each of the roads. These insights can be then further used to

notify people who are traveling what is the best route between two points, depending on the time of the day and day of the week.

2.5 Basic concepts

As already said, two are the basic capabilities of any Big Data environment: computing and storage.

2.5.1 Computing

Computing capabilities (both processing power and available analysis technologies) are essential in every Big Data environment. On the one hand, the data volume is going to be large enough for having an efficient processing strategy (without being real time, the response time must be acceptable). On the other hand, we need efficacy when looking for insights (we need the better insights, not whatever ones). Efficiency is achieved by means of the MapReduce paradigm, while efficacy is given by the analytic tools and the jobs chaining orchestration.

2.5.1.1 *MapReduce*

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 which 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.

From the field of functional programming, it is proved that any problem which 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 can 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 in an efficient way, 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, where 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.

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.

2.5.1.2 **Analytic tools**

Despite of MapReduce is the core of batch processing, its interface/user experience is complex and only suitable for developers knowing very well the different APIs of the chosen implementation (Java, Python, etc). Arises then the need for a new kind of tools abstracting the complexity of the MapReduce APIs and exposing easier interfaces.

(1)SQL-like tools

The more popular analytic tools around MapReduce are those based on SQL. This is a wide extended, high level and powerful database query language, and its adaptation to the Big Data world had to be a success. And that was the case with Hive [[BDA9](#)], Pig [[BDA10](#)], Shark [[BDA22](#)], Impala [[BDA12](#)] and may other analytic tools running on top of Hadoop.

The key of the success of these tools is, as said, to abstract the MapReduce paradigm, which is still working but in a transparent way. So, when the user decides to retrieve certain information, for instance a specific column of the dataset, the SQL query is automatically transformed into some predefined MapReduce jobs; in the proposed example, a Map function will iterate on all the rows in order to select the desired column, and the Reduce function joins all the individual values in a unique and new column.

As can be inferred, the kind of datasets where these analytic tools have sense are those whose information is stored in a structured fashion, i.e. datasets whose content may be easily transformed into SQL tables.

(2)Other

There are many other alternatives to the SQL-like analytic tools. It is the case of the R adaptation for Hadoop [[BDA13](#)]; the machine learning tools included in MAhout [[BDA14](#)], a component of the Hadoop ecosystem; or the extensions allowing for programming in Python the MapReduce jobs [[BDA15](#)].

These and other tools are out of the scope of the Big Data GE and will not be explained nor detailed.

2.5.1.3 *Coordination*

No data analyst will be able to do a complex processing in a single step. Maybe the chosen tools do not provide all the required functionalities; and in the case the tool is suitable for performing all the analysis, this confronts all the good developing practices (modularity, reusing, etc.). But the main reason for not doing so is that the complex analysis are commonly based in a sequence of steps where several tools may be involved: the output of an analysis will be the input of another one, thus, until the first one does not finish, the second one will not be launched. The problem here is to monitor when a job finishes and its putput can be injected to the next job, and this is something cannot be done by simply waiting because it can last hours even days. It seems to be necessary an orchestration mechanism of processes. There are several alternatives for orchestration, mainly in widely used ecosystems such as Hadoop, where Oozie is the main asset [[BDA16](#)].

2.5.2 *Storage*

If the computing capabilities allow for working with the data, the storage ones make possible the data can be available for the computing capabilities. For achieving that, obviously it is necessary to have a big amount of storage space, but it is also critical the data is stored in an efficient way in order the access time does not become traumatic. Finally, the storage capacities do not have sense without data injection mechanisms.

2.5.2.1 *Distributed storage systems*

A distributed storage system is a network of computers where the information is stored in more than one node, in order each one of those nodes owns certain part of the original information. It is very common these distributed systems apply a replication factor as well, i.e. each part of the original data is store twice o more times in different nodes of the network. The goal is to ensure the data recoverabilty when a node fails.

The distributed storage systems are crucial when dealing with Big Data environments because the grant the required storage volume and the scalability. Normally, a centralized process manages the system, providing a unique view of the total storage independently of the number of nodes within the network and the information contained in each node. This unique view offers services for creating, modifying and deleting files. This is possible because certain meta-information is managed: a list of distributed files, containing each file another list of data blocks and the location of each block.

In Big Data environments, the de facto standard is the Hadoop Distributed File System (HDFS) [[BDA8](#)], the file system under the MapReduce engine of Hadoop. However, many other distributed storage systems are arising, compatible or not with Hadoop, such as Cassandra File System (CFS) [[BDA17](#)] or Google File System [[BDA7](#)].

2.5.2.2 **NoSQL storage systems**

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 [[BDA27](#)] [[BDA28](#)], 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 [[BDA29](#)]). 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. These kinds of stores are typically used in data warehousing. Example implementations are Hadoop HBase [[BDA18](#)], Google BigTable [[BDA19](#)] and Apache Cassandra [[BDA17](#)] (which in addition, as already seen, is a whole file system).

- Key Value Store - A schema-less storage system where data is stored in key-value pairs. Data is accessed via a hash table using the unique key. Example implementations are Dynamo [[BDA23](#)] and Redis [[BDA20](#)].
- 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 are CouchDB [[BDA24](#)] and MongoDB [[BDA25](#)].
- 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. An example implementation is neo4j [[BDA26](#)].

2.5.2.3 *Data lifecycle*

The data to be analyzed in Big Data environments may be of very different nature, but as already commented, they can always be classified into two groups: batch and streaming. The first ones are big collections of data that must be stored in advance to the analysis, while the second kind of data is analyzed on the fly, in real or almost real time.

When dealing with one type of data or the other this difference leads to adopt different ways for feeding the Big Data environment. Thus, in some cases will be necessary to allow input ports for dynamic flows of data, while in other cases the copying historics from the source to the distributed storage will be enough. In any case, this phase is called Data Injection phase, and among other things, it may imply the normalization, anonymization, encryption, filtering and compression of the data.

Next is the Data Ingestion Phase. Within this phase it must be ensured the necessary capability to absorb the volume of data is available. Then, and only then, the data copying (if it has sense) can be performed, either in raw or a predefined format. Sometimes, as occurs with the flows or events, it may be required an analytic study in the phase, due to such data is not usually stored.

After the data ingestion arises the Processing Phase, involving a unique analysis or an orchestrated chain of several ones. Certain elasticity must be granted if the computing capabilities increase during the processing task, and the data protection (privacy, integrity, etc.) must be monitored as well.

Finally, the results are consumed in the last phase. This means the output data becomes new input data for new other analysis, visualization tools, query systems, etc. The results may be stored in a different external repository in order to ensure their perdurability and allow for better access times.

2.6 Architecture

2.6.1 Reference architecture: Apache Hadoop

As will be seen in the next section, the Big Data GE architecture expands the basic architecture of Apache Hadoop. Thus it is necessary to briefly explain it before.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

Three are the pillars of Hadoop:

- Hadoop Common, which provides access to the supported file systems (HDFS and many others).
- The engine for parallel processing of vast amounts of data (Hadoop MapReduce), based on a proprietary resource manager and job scheduler (Hadoop YARN).
- Hadoop Distributed File System (HDFS), a distributed file system assuring high throughput when accessing the data.

Architecturally speaking, a distributed file system is on the bottom of the software stack. Please observe several compatible distributed file systems may be used, e.g. Amazon S3, CloudStore or FTP FileSystem, but here only HDFS will be explained. HDFS supports big data files by splitting them and distributing their blocks among all the Datanodes of the cluster. A special node called the Namenode is responsible for storing the indexes for each distributed file and, in general, for file system management (naming, deletions, creations, etc.).

Over the HDFS is the MapReduce layer, mainly a set of Tasktracker nodes in charge of performing map and/or reduce tasks over the data stored in the HDFS, being a special node called the Jobtracker who decides which task and which data is assigned to each Tasktracker. The list of tasks to be collaboratively accomplished is extracted from a Java JAR file, built by the analyst using the Hadoop API [[BDA36](#)] and containing the desired MapReduce job in terms of Java sentences invoking the Hadoop development API. This Java JAR is also replicated several times in the HDFS by the Jobtracker in order it is highly available for the final interpreters of that Java code, the Tasktrackers. Finally, a workstation or JobClient node is used to initially upload the JAR file to the HDFS and invoke the Hadoop execution.

The above architecture may change depending on the performance requirements and/or the dimensions of the available cluster. If the cluster is very limited, it may be necessary to merge two or more nodes in a single one, which penalizes the CPU performance, but improves the transactions at the same time. Therefore, it is usual to find both Tasktraker and Datanode coexisting in the same node, or

having the Jobtracker running in the same machine than the Namenode. If the cluster is very limited, it is also possible to have a node acting both as Jobtracker, Namenode and (limitedly) Tasktracker and Datanode.

Once a job is finished, some resulting insights are generated. The HDFS may store these results together to the input data, but it is not recommended due to bad access response times, and a NoSQL database is suggested for these purposes (see below).

2.6.2 Big Data GE Architecture

The Big Data GE is based on a Master Node executing management software and acting as a frontend for the final users. Usually, these users will be applications that, using the administration API, will request the creation of storage and/or computing clusters.

On the one hand, the creation of a cluster implies the creation of a Head Node, in charge of the storage management (it does not store anything), the computing management (it does not compute anything) and other specific services (analysis tools, data injectors...). On the other hand, one or more Slave Nodes are created as well, being their purpose the execution of analysis tasks and real storage of the data.

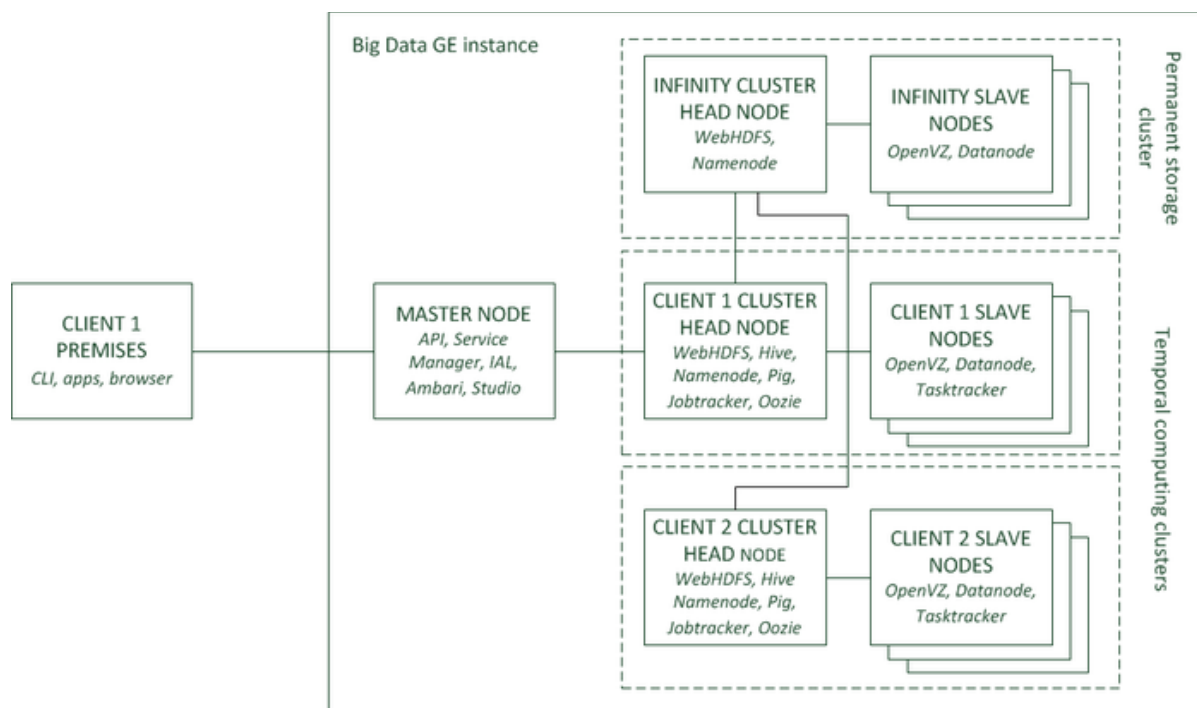


Figure BDA-1 - Big Data GE architecture

As can be seen in the above figure, both the Head Node and the Slave Nodes imitate the Namenode and Jobtracker combination, and the Datanode and Tasktracker combination, respectively. Indeed, those Hadoop services can be used for configuring the Big Data GE Head and Slave nodes, but many other solutions could be used as well:

- Cassandra File System (CFS) [[BDA17](#)], resulting in a configuration where the Hadoop MapReduce engine continues working on top of CFS.
- To install a NoSQL distributed database such as HBase [[BDA18](#)](key-pair schema) on top of HDFS is another option.
- MapReduce can be also substituted/improved by using top level analytic tools such as Cascading [[BDA21](#)], or in-memory analysis systems following the MapReduce paradigm but providing almost real-time results.
- On top of everything, as previously said, a bunch of querying tools is available for the user, e.g. Hive [[BDA9](#)], Impala [[BDA12](#)] or Shark [[BDA22](#)].

The combination of modules from the Hadoop ecosystem that can be installed in the Big Data GE are multiple, and they will depend on the needs the user has. The following figure tries to depict all the software stack that could be installed in the GE:

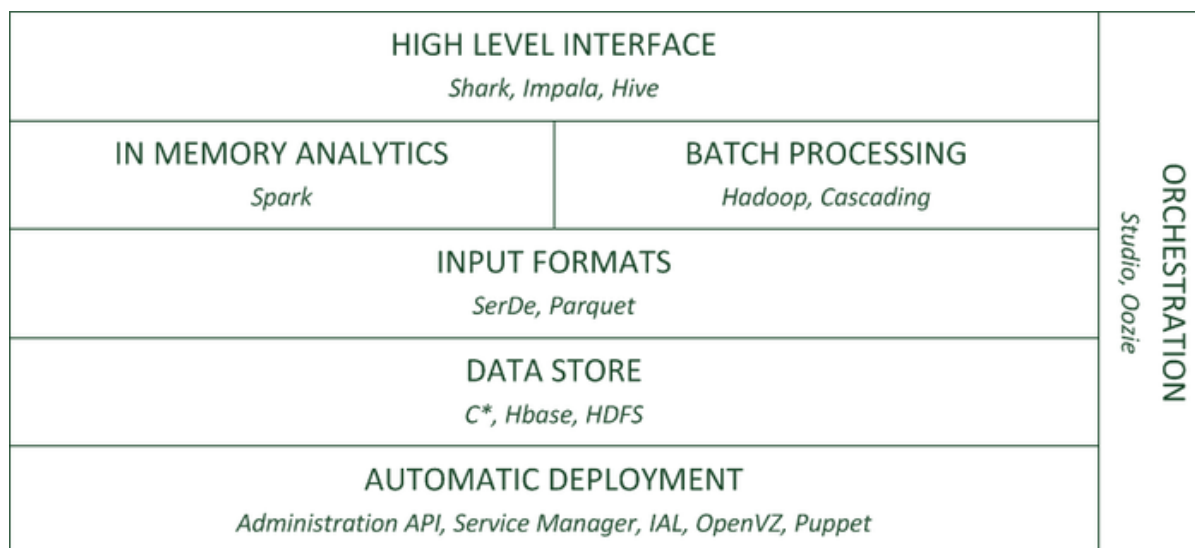


Figure BDA-2 - Big Data GE software stack

Next sections will go into the details about each one of the nodes, their functionality and the components implementing them. The pre-defined cluster for persistent storage will also be shown.

2.6.3 User environment

The user environment is about the application or set of application in charge of interacting with the Master Node regarding the creation of a Big Data cluster, the MapReduce jobs launching, etc. These application may be legacy (e.g. some REST manager), proprietary (using any of the REST libraries) or predefined such as the CLI and the Studio.

2.6.3.1 **CLI**

It is an easy command-line-based application that hides the details of the administration REST API managed by the Master Node. Thus, through intuitive command such as "create a cluster", "upload this file", etc. the user is able to interact with the whole platform.

Since the CLI only interacts with the administration REST API, it is not able to execute analysis of any type. Therefore, once the cluster has been created, it is necessary to use the appropriate interfaces exposed by the analysis tool itself (the "hadoop" command if trying to do MapReduce, SQL-like queries if dealing with Hive, etc.).

2.6.3.2 **Studio**

The Studio is a Graphical User Interface allowing for workflow definitions, specifying in each step of the workflow which kind of analysis must be done (MapReduce job, SQL query, Python scripts, proprietary code...) and how the outputs of the different analysis steps are connected to the inputs of new analysis steps. Once the workflow is defined, it is executed in a cluster specifically created for it.

Typically, the orchestration of jobs is done through Oozie [[BDA16](#)], and this component also runs under the Studio. The graphical workflows are translated into Oozie workflows, and once an Oozie server has been installed in the cluster, these configurations are loaded by using the Oozie APIs.

Nevertheless, a component like this one can also perform other interesting functionalities, such as:

- Host an historical log of workflow executions.
- Host a collection of reusable software.
- Work as an audit platform, due the behaviour of any workflow remains the same all along the time.

2.6.4 **Master node**

The master Node acts as the endpoint of the Big Data platform. It is the node the user will go in order to create storage clusters and/or computing clusters.

2.6.4.1 **Administration API**

The administration API exposes all the operations it is possible to perform in the platform, these ones:

- Cluster creation, parameterized by the cluster identifier, the size of the cluster and the Hadoop ecosystem services that must be installed.
- Cluster deletion, given its identifier.
- Get the cluster details, given its identifier.

- List all the available services that can be installed in a cluster.
- Get the details regarding the Head Node of a cluster.
- Get your user profile details.
- Modify your user profile details.

2.6.4.2 ***Service Manager***

It interprets all the operation of the administration API related to the creation and deletion of clusters. Depending on the operation, the Infrastructure Manager allocates/frees storage and/or computing capabilities, and the software is deployed/undeployed as well.

2.6.4.3 ***Infrastructure Abstraction Layer***

This component independizes the requests done by the Service Manager to the different Infrastructure Managers (the platform may be able to deploy clusters on top of many infrastructure types).

2.6.4.4 ***Ambari***

Ambari [[BDA31](#)] is used to make Hadoop management (provisioning, managing, and monitoring clusters) simpler.

2.6.4.5 ***Studio server***

This is the server side of the Studio described above.

2.6.5 Head node

The Head Node hosts all the services offered by the cluster, both the services exposed to the users and the services exposed for internal purposes.

2.6.5.1 ***Storage (management)***

Regarding storage, the Head Node exposes an internal service based on the Namenode from Hadoop. The Datanodes, which run in the slave nodes (see next sections), will connect with this service in order to receive instructions about I/O. On top of the Namenode service may run, if configured, the HBase service. HBase, conceptually, is a Big Table containing a great volume on key-value pairs, but internally is based on the Hadoop distributed file system (HDFS).

An alternative to HDFS is Cassandra. Nevertheless, Cassandra still needs the Namenode service from Hadoop.

2.6.5.2 **Data injection**

(1)WebHDFS

WebHDFS [[BDA34](#)] is the native REST API from Hadoop, and it is exposed for HDFS management purposes. It works by receiving I/O requests which are managed by the HDFS Namenode, and this service returns a redirection pointing to the specific Datanode where the data can be finally read or write.

(2)HttpFS

It exists an alternative to WebHDFS, HttpFS [[BDA35](#)], which implements the same API but whose behaviour is a bit different.

When using WebHDFS, both the node running the Namenode (Head Node) and each node running the Datanode (Slave Nodes) must have a public IP address (or their FQDN must be resolved into a public IP address). This is because both the Namenode and the Datanodes (pointed by the redirections) must be reachable. This issue implies the usage of a huge number of public IP addresses that is not necessary: HttpFS acts as a gateway, and instead redirecting to the Slave Nodes, redirects to the Head Node itself (which in the end internally performs the desired operation with the Datanode).

2.6.5.3 **Processing (management)**

The tracking of the batch analysis is done through the Jobtracker service from Hadoop. The slave nodes, running the Tasktracker service from Hadoop as well, will connect to this Head Node in order to receive instructions about which Map tasks and/or Reduce tasks (the same node can run both types of tasks) must be executed.

Cascading, a framework for the development, execution and synchronization of MapReduce jobs in Java, can be also installed in the Head Node. The libraries from Cascading highly simplify the Hadoop API. The processing is performed in almost real time thanks to Spark (similar to Hadoop, but memory-based). Anyway, Spark is able to access the HDFS data for processing purposes.

2.6.5.4 **Analytic tools**

The Big Data GE provides SQL-like tools for analysis purposes. Examples are Hive, Impala, Shark. The envisioned used tools are Hive and Pig.

2.6.5.5 **Coordination**

The orchestration in the Big Data GE is done through Oozie. The Head Node runs this service, which is accessible by means of a REST API.

2.6.6 Slave nodes

2.6.6.1 *Storage*

The slave nodes run the Datanode service from Hadoop, needed to build the HDFS of the GE.

2.6.6.2 *Processing*

The slave nodes run the Tasktracker service from Hadoop, needed in the MapReduce processing.

2.6.7 Cluster for persistent storage (Infinity)

In addition to the processing clusters, which are created and destroyed on demand by the users, the platform has a default cluster used for persistent storage. It is available for all the users of the GE, even those not desiring to create processing clusters, and the data injection tools are the already described.

The goal of this cluster is, as said, to offer a persistent storage service, due to all the processing clusters die in the end. The user is responsible for deciding which data is stored in the persistent storage, and viceversa, which persistently stored data is going to be used by the processing cluster. Anyway, the data stored in this special cluster are protected from other users.

This persistent storage does not provide any processing capabilities.

2.7 Main Interactions

2.7.1 Providing input data to Infinity

Infinity is the permanent storage cluster of the GE, based on HDFS. As already explained, this is because the computing clusters have a clear lifecycle: they are created, used for certain computations and finally they are destroyed. Thus all the data to be analyzed must be initially uploaded to Infinity and used from that location.

The way the data is uploaded is done through:

- WebHDFS [[BDA30](#)], the native REST API from Hadoop for HDFS filesystem management.
- Cosmos CLI, a piece of software intended to wrap a WebHDFS client and exposing a "command line" style interface to the user.

In previous releases of Cosmos there was the possibility to upload data by using a HDFS-based SFTP server, but this is a deprecated option.

2.7.2 Retrieving output data from Infinity

The same than in the above case applies when retrieving data from Infinity. Both WebHDFS and the CLI can be used for such purposes.

Again, the usage of the HDFS-based SFTP server is deprecated.

2.7.3 Creating a computing cluster

The creation of computing clusters will be achieved by using:

- The administration API of the GE, which will provide in a REST API style a specific operation for this purpose.
- The CLI, which will wrap a client using the above REST operation and exposing a "command line" style interface to the user.

The creation of a computing client will imply:

- The creation of a Head node containing a private Namenode, Jobtracker and private services for Oozie, Hive and Pig, in addition to a WebHDFS server for the private cluster.
- The creation of a certain number of Slave nodes, containing each one of them private Datanodes and Tasktrackers.

Each private computing cluster will have access to the user-related space in Infinity's HDFS.

2.7.4 Uploading and running MapReduce jobs

Once a private cluster has been created, it can be used to analyze the data stored by the user in Infinity. In order to do that, it will be necessary to develop a MapReduce application in a client machine, following the guidelines described at the [User and Programmer Guide](#) of this GE.

Once the MapReduce application has been developed, it can be uploaded to the private cluster. This step is achieved by copying the application to the Head node, and there it must be copied to the private HDFS by using standard Hadoop copying commands. MapReduce applications are run by commanding Hadoop from a shell opened in the Head node of the private cluster.

In order to do the uploading and running of the applications it is necessary the user logs into the Head node. The running part can be avoided if Oozie is used. Oozie is an analysis tool designed for scheduling MapReduce applications, Hive and Pig tasks, and shell scripts.

2.7.5 Querying for data (Hive or Pig)

Data is queried from the client side by using Hive Query Language [[BDA32](#)] or Pig Language [[BDA33](#)]. These languages are talked by specific clients connecting to the servers running in the private cluster

(within the Head node). Hive or Pig are SQL-like languages which implement pre-defined MapReduce tasks in order to select, count, limit, order, etc. the data. These MapReduce tasks run in the private cluster as any other MapReduce job does, and the data is accessed from Infinity (permanent storage cluster) as any other MapReduce job does; for this purpose, Infinity must allow the creation of Hive and Pig tables.

2.8 Basic Design Principles

The basic desing principles of the Big Data GE are:

- To hide the complexy behind the process of creating Big Data environments, where some software packages must be appropriately configured in order they work in a coordinated fashion.
- To offer a wide set of processing and querying technologies for being installed in the environment. The GE exposes a catalogue, which can grow up in an easy and opened way.
- To focus the efforts on extracting insights and value added information. To achieve that, there are components allowing for complex SQL-like queries design, reusing binaries and to compose processing sequences thanks to an easy and intuitive web-based interface.

These basic principles make the Big Data GE shows the following differentiating features when compared with similar solutions:

- Independency of the resource provision technology: The GE is independent of the chosen infrastructure management system, i.e. the environments may be created in the cloud or directly on bare metal.
- The Studio allows for components reusing and analysis compositions: the capability for reusing and combine different pieces of code, scripts or queries the different users of the GE create makes it becomes an opened environment, allowing a greater potential innovation due to the quick sharing of components and quick experimentation.
- The GE has the capability to deploy and configure heterogeneous technologies: it is possible to create environments where different technologies, focused on solving complementary parts of the same problem, can coexist. This way, it is possible to combine the batch processing with the real time processing, the storage and the SQL-like querying. I.e. the Big Data GE allows creating data processing architectures.
- Data lifecycle end-to-end view: The GE covers all the phases of the data lifecycle, such as data injection, alarm programming, batch processing, real time processing, persistent storage, insights consumption through an API or the possibility to access them by means of SQL queries.

2.9 References

- BDA1 [Google's MapReduce](#)
- BDA2 [MapReduce US patent by Google](#)
- BDA3 [Apache Hadoop](#)
- BDA4 [Sector](#)
- BDA5 [Sphere](#)
- BDA6 [Dryad](#)
- BDA7 [Google File System](#)
- BDA8 [Apache HDFS](#)
- BDA9 [Apache Hive](#)
- BDA10 [Apache Pig](#)
- BDA11 [Apache Spark](#)
- BDA12 [Impala](#)
- BDA13 [Enabling R on Hadoop](#)
- BDA14 [Apache Mahout](#)
- BDA15 [Dumbo](#)
- BDA16 [Apache Oozie](#)
- BDA17 [Apache Cassandra](#)
- BDA18 [Apache HBase](#)
- BDA19 [Google Big Table](#)
- BDA20 [Redis](#)
- BDA21 [Cascading](#)

BDA22 [Shark](#)

BDA23 [Amazon Dynamo](#)

BDA24 [Apache CouchDB](#)

BDA25 [MongoDB](#)

BDA26 [Neo4j](#)

BDA27 [CAP Theorem](#)

BDA28 [Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services](#)

BDA29 [Eventually Consistent - Dr. Werner Vogels](#)

BDA30 [WebHDFS](#)

BDA31 [Apache Ambari](#)

BDA32 [HiveQL](#)

BDA33 [Pig Latin](#)

BDA34 [WebHDFS](#)

BDA35 [HttpFS](#)

BDA36 [Hadoop API](#)

2.10 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.

2.10.1 Open API Specifications

- [BigData Analysis Open RESTful API Specification](#)

2.11 Re-utilised Technologies/Specifications

Cosmos is mainly based on Hadoop ecosystem, not in vain it is intended to be a platform about Hadoop-based cluster automatic deployments. Thus, the re-utilised technologies in a **Cosmos computing cluster** are:

- [Hadoop](#) MapReduce engine
- Hadoop Distributed File System ([HDFS](#))
- [Hive](#) and [Pig](#) as QL-like querying tools
- [Sqoop](#) as MySQL connector, moving data from/to Cosmos
- [Oozie](#) as remote MapReduce job launcher and task scheduler

The **Cosmos storage cluster** (Infinity) also relies on [HDFS](#).

On the other hand, the **automatic deployment engine**, in charge of deploying the computing clusters, is based on [Ambari](#).

2.12 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 **avalue**. 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 be 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 two 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

3.1 Introduction to the Big Data API

This document is intended to describe the Open RESTful API Specification of the Big Data GE. Nevertheless, please notice not all the APIs planned for the GE are REST.

The Big Data GE is mainly based on the Hadoop ecosystem, where several tools, APIs and other interfaces have largely been specified and used along the time by analysts and data scientists. These so-called *Hadoop Open API Specification* is not always supported by a REST style, and thus, by extension, not all the Big Data GE APIs are REST.

Other interfaces, especially those related to proprietary modules added to the Hadoop ecosystem, are, however, full RESTful APIs.

3.1.1 Big Data API Core

The Big Data GE API is a set of sub-APIs, each one addressing one of the different aspects of a Big Data platform:

- **Admin** **API:**
RESTful, resource-oriented API accessed via HTTP that uses JSON-based representations for information interchange. This API is used for administration purposes, allowing the management of users, storage or computing clusters.
- **Hadoop** **commands:**
Command-line style interface used both to administrate HDFS and launching MapReduce jobs.
- **Hadoop** **API:**
Java-based programming API used both to administrate HDFS and launching MapReduce jobs.
- **HDFS** **RESTful** **APIs:**
RESTful, resource-oriented API accessed via HTTP that uses JSON-based representations for information interchange. This API is used for HDFS administration. Includes: WebHDFS, HttpFS and Infinity Protocol.
- **Querying** **systems:**
SQL-like languages such as HiveQL executed locally in a CLI interpreter, or remotely through a server.
- **Data** **injectors:**
A mix of technologies used to inject data in the storage mechanisms of the GE.
- **Oozie** **API:**
Java-based programming API used to orchestrate the execution of MapReduce jobs, Hive tasks, scripts and user-defined applications.

- **Oozie** **Webservices** **API:**
RESTful-like, resource-oriented API accessed via HTTP that uses JSON-based representations for information interchange. This API is used for Oozie-based job and task orchestration.

3.1.2 Intended Audience

This specification is intended for both software developers and reimplementers of this API. For the former, this document provides a full specification of how to interact with the GE in order to perform Big Data operations. For the latter, this specification provides a full specification of how to write their own implementations of the APIs.

In order to use this specifications, the reader should firstly have a general understanding of the appropriate [Big Data GE architecture](#) supporting the API.

3.1.3 API Change History

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

Revision Date	Changes Summary
Apr 11, 2014	<ul style="list-style-type: none">• First version

3.1.4 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 [Basic Concepts](#) section in the [Big Data GE architecture](#).

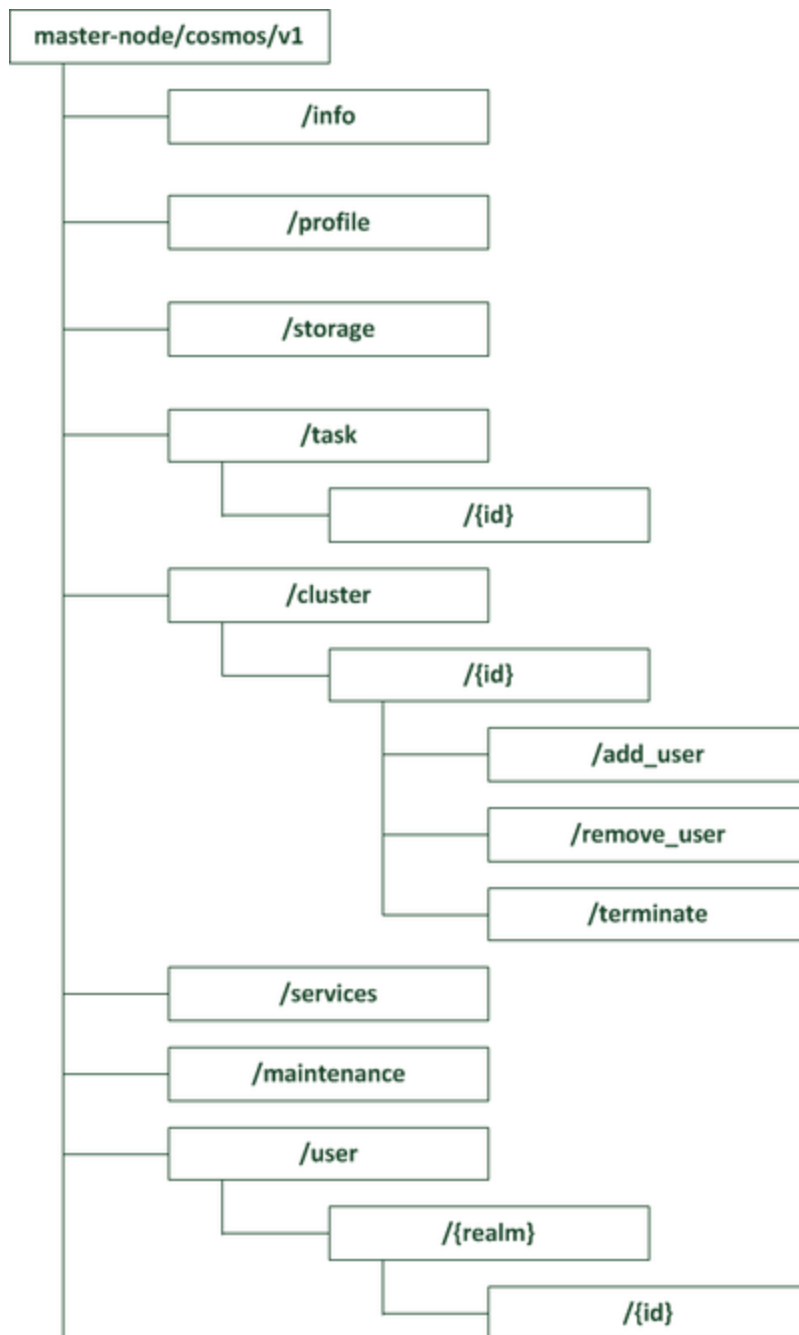
3.1.5 Additional Resources

N.A.

3.2 General Big Data GE API Information

3.2.1 Resources Summary

The following applies to the Admin API of the Big Data GE. For other legacy RESTful APIs please refer to the appropriate link. For non-RESTful APIs, nothing has to be provided.



3.2.2 Representation Format

The Big Data GE API, in its RESTful parts, supports JSON-based representations for information interchange.

3.2.3 Resource Identification

- Task identifiers follow the [TaskID specification](#) in the Hadoop API.

- Cluster identifiers are strings without a fixed format.
- Realm identifiers are strings without a fixed format.
- User identifiers are string without a fixed format.

3.2.4 Links and References

N.A.

3.2.5 Limits

N.A.

3.2.6 Versions

There is a unique version of this API, thus nothing has to be said about deprecated operations.

3.2.7 Extensions

N.A.

3.2.8 Faults

3.2.8.1 *Synchronous Faults*

N.A.

3.2.8.2 *Asynchronous Faults*

N.A.

3.3 API Operations

3.3.1 Admin API

This administration API, which is mostly wrapped by the CLI, is aimed to manage the platform in terms of user and cluster creation/modification/deletion and task and storage management. It is served by the Master node of the platform, thus it is common to all the users.

3.3.1.1 */info*

Verb	URI	Description	Request payload	Response payload
GET	/cosmos/v1/info	Get general information about the user whose credentials are used	-	<pre> { "resources": { "availableForUser": "int", "available": "int", "groupConsumption": "int", "availableForGroup": "int", "individualConsumption": "int" }, "individualQuota": "Object", "handle": "string", "clusters": { "accessible": [{ "id": "string" }] } } </pre>

				<pre>], "owned": [{ "id": "string" }], "profileId": "long", "group": { "guaranteedQuota": "Object", "name": "string" } }</pre>
--	--	--	--	--

3.3.1.2 */profile*

Verb	URI	Description	Request payload	Response payload
GET	/cosmos/v1/profile	Get the user profile details	-	{

				<pre>"handle": "string", "email": "string", "keys": [{ "name": "string", "signature": "string" }]</pre>
PUT	/cosmos/v1/profile	Update the user profile details	<pre>{ "handle": "string", "email": "string", "keys": [{ "name": "string", "signature": "string" }] }</pre>	<pre>{ "handle": "string", "email": "string", "keys": [{ "name": "string", "signature": "string" }] }</pre>

]]
			}	}

3.3.1.3 */storage*

Verb	URI	Description	Request payload	Response payload
GET	/cosmos/v1/storage	Persistent storage connection details	-	<pre>{ "location": "URI", "user": "string" }</pre>

3.3.1.4 */task*

Verb	URI	Description	Request payload	Response payload
GET	/cosmos/v1/task/{id}	Get task details	-	<pre>{ "id": "int", "status": "TaskStatus", "resource": "string" }</pre>

				}

3.3.1.5 */cluster*

Verb	URI	Description	Request payload	Response payload
GET	/cosmos/v1/cluster	List clusters	-	<pre>{ "clusters": [{ "clusterReference": { "assignment": { "creationDate": "Date", "ownerId": "long", "clusterId": { "id": "string" } } }, "description": "ClusterDescription" }] }</pre>

				<pre>}, "href": "string" }] }</pre>
POST	/cosmos/v1/cluster	Create a new cluster	<pre>{ "optionalServices": ["string"], "name": { "underlying": "string" }, "size": "int" }</pre>	-
POST	/cosmos/v1/cluster/{id}/add_user	Add users to existing cluster	<pre>{</pre>	-

			<div>"user": "string"</div> <div>}</div>	
GET	/cosmos/v1/cluster/{id}	Get cluster machines		<div>{</div> <div> "id": "string",</div> <div> "users": [<div> {</div><div> "publicKey": "string",</div><div> "enabled": "boolean",</div><div> "sudoer": "boolean",</div><div> "username": "string",</div><div> "hdfsEnabled":</div><div> "boolean",</div><div> "sshEnabled": "boolean"</div><div> }</div><div>],</div><div> "services": [<div> "string"</div><div>],</div><div>}</div></div></div>

				<pre>"stateDescription": "string", "name": { "underlying": "string" }, "slaves": [{ "hostname": "string", "ipAddress": "string" }], "state": "string", "href": "string", "master": { "hostname": "string", "ipAddress": "string" }, "size": "int" }</pre>
--	--	--	--	--

POST	/cosmos/v1/cluster/{id}/remove_user	Remove a user from an existing cluster	<pre>{ "user": "string" }</pre>	-
POST	/cosmos/v1/cluster/{id}/terminate	Terminate an existing cluster	<pre>{ "message": "string" }</pre>	-

3.3.1.6 */services*

Verb	URI	Description	Request payload	Response payload
GET	/cosmos/v1/services	List services that can be deployed in the private cluster	-	The response is a bare array of strings (e.g. ["service1", "service2"])

3.3.1.7 */maintenance*

Verb	URI	Description	Request payload	Response payload
GET	/cosmos/v1/maintenance	Check if some maintenance is being done on the platform	-	boolean

PUT	/cosmos/v1/maintenance	Change the maintenance status	boolean	boolean
-----	------------------------	-------------------------------	---------	---------

3.3.1.8 /user

Verb	URI	Description	Request payload	Response payload
POST	/admin/v1/user	Create a new user account	{	
			"sshPublicKey": "string", "authRealm": "string", "handle": "string", "email": "string", "authId": "string"	{ "handle": "string", "apiSecret": "string", "apiKey": "string" }
			}	
DELETE	/admin/v1/user/{realm}/{id}	Delete an existent user account		{ "message": "string" }

3.3.2 Hadoop commands

Hadoop can be completely governed by commanding it through a Shell. The complete list of Hadoop commands can be found in the [official documentation](#).

It is worth mentioning the file system subcommand (`hadoop fs`). This command is used for HDFS administration (file or directory creation, deletion, modification, etc), and offers a complete suite of options that can be found in the [official documentation](#) as well.

3.3.3 Hadoop API

The Hadoop API is a Java-based programming API addressing several Hadoop topics, but it is mainly used for:

- Programming custom HDFS clients.
- Programming custom MapReduce applications.

You can explore this API in the [official Javadoc](#).

3.3.4 HDFS RESTful APIs

These API specifications are given by the Hadoop ecosystem, since WebHDFS/HttpFS have been already defined by Apache and the Infinity Protocol is an augmented version of WebHDFS.

3.3.4.1 **WebHDFS**

While the Hadoop commands are useful for users with authorized access to the machines within the cluster, or applications running within it, sometimes it is necessary to access the HDFS resources by an external entity. To achieve this, Hadoop provides a HTTP Rest API supporting a complete interface for HDFS. This interface allows for creating, renaming, reading, etc. files and folders, in addition to many other common operations on file systems (permissions, owning, etc.). It is available in the 50070/TCP port of the *NameNode*.

The whole details for this API can be found in the official documentation [\[1\]](#).

Please observe some actions, such as CREATE or APPEND, are two-step operations, sending a first request to the *NameNode* of the cluster, and receiving a response containing a redirection URL to the final *DataNode* where the data must be stored.

Please notice as well there is no operation for uploading and running jobs.

3.3.4.2 *HttpFS*

HttpFS is another implementation of the WebHDFS API (14000/TCP port of the server, which usually runs on the *NameNode*). While the WebHDFS implementation implies a total knowledge and accessibility to the whole cluster, HttpFS hides these details by behaving as a gateway between the Http client and WebHDFS. Two-step operations continue working in the same way but now the redirection URL points to the same HttpFS server; the first and second operations are distinguished by adding a new parameter, 'data=true', in the redirection.

HttpFS is not natively available for CDH3 (it is distributed from CDH4), but there is a backporting for CDH3 at GitHub [\[2\]](#).

3.3.4.3 *Infinity Protocol*

The Infinity Server is an Ambari service that runs on all nodes in the Infinity cluster. Deployment of this service installs the corresponding executables and blocks all traffic (except traffic from any Infinity server) to the following ports:

- Namenode metadata service: 8020 and 9000
- Datanode data transfer: 50010
- Datanode metadata operations: 50020
- WebHDFS ports: 50070 (namenode), 50075 (datanode)

The Infinity Server listens on a port of your choice and provides an augmented WebHDFS REST API that uses HTTPS to avoid data sniffing and an authentication mechanism. The authentication mechanism can be one of these two:

- A query parameter called secret to authenticate the user through its cluster secret. The cluster secret is a long ASCII string which authenticates the user in the context of a given cluster:

<http://infinity:123/webhdfs/v1/user?user.name=turing&op=LISTSTATUS&secret=bac69680bafb11e19fd7c2b027b06d18>

- The API key and API secret of the user through an HTTP basic auth header:

Authorization: Basic QWxhZGRpbjpvYVUHNlc2FtZQ==

3.3.5 Querying systems

These API specifications are given by the Hadoop ecosystem, since Hive and Pig have been already defined by Apache

3.3.5.1 *Hive*

The clusters created with the Big Data platform provide the Hive querying system. As stated in the Hive home page [3], the Apache Hive data warehouse software facilitates querying and managing large datasets residing in distributed storage. Hive provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL. At the same time this language also allows traditional map/reduce programmers to plug in their custom mappers and reducers when it is inconvenient or inefficient to express this logic in HiveQL.

The [Apache Hive wiki](#) provides full information about how to use this tool. Among other interesting topics you could find:

- [Hive Query Language](#) reference.
- How to use the [command line interface](#),
- How to use the [Hive Web Interface](#), a web-based GUI which is an alternative to the command line interface.
- How to build ODBC-based [hive clients](#).

3.3.5.2 *Pig*

Pig is another querying tool similar to Hive. From its home page [4] we found that Apache Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. The salient property of Pig programs is that their structure is amenable to substantial parallelization, which in turns enables them to handle very large data sets.

The [getting started](#) guide will lead you on how to run Pig, how to write Pig scripts, etc.

3.3.6 Data injectors

3.3.6.1 *Sqoop*

From its user guide [5], Sqoop is a tool designed to transfer data between Hadoop and relational databases. You can use Sqoop to import data from a relational database management system (RDBMS) such as MySQL or Oracle into the Hadoop Distributed File System (HDFS), transform the data in Hadoop MapReduce, and then export the data back into an RDBMS. Sqoop automates most of this process, relying on the database to describe the schema for the data to be imported. Sqoop uses MapReduce to import and export the data, which provides parallel operation as well as fault tolerance.

Full details can be found, as said, in the [Sqoop user guide](#).

3.3.6.2 *Cygnus*

Cygnus is the connector allowing for Publish Subscribe Context Broker GE context data persistence in the Big Data GE. This is a component that is designed to receive notifications about certain data context for which Cygnus has previously subscribed to. Thus, the interface for this connector is just a simple Http server listening for REST-based notifications coming from the Publish Subscribe Context Broker GE.

3.3.7 Orchestration

(1)Oozie

Oozie is a workflow scheduler system to manage Apache Hadoop jobs. Oozie is integrated with the rest of the Hadoop stack supporting several types of Hadoop jobs out of the box (such as Java map-reduce, Streaming map-reduce, Pig, Hive, Sqoop and Distcp) as well as system specific jobs (such as Java programs and shell scripts) [6].

Writing Oozie applications is as simple as including in a *package* the MapReduce, Hive/Pig scripts, etc. going to be run and defining a Workflow. Oozie Workflow jobs are Directed Acyclical Graphs (DAGs) of actions. Oozie Coordinator jobs are recurrent Oozie Workflow jobs triggered by time (frequency) and data availability.

Full documentation about Oozie can be found [here](#), including:

- The [command line interface](#).
- The [Workflow functional specification](#).
- The [Web Services API](#) it provides.
- How to create [custom Oozie clients](#) written in Java.

4 FIWARE OpenSpecification Data ContextBroker

Name	FIWARE.OpenSpecification.Data.ContextBroker		
Chapter	Data/Context Management,		
Catalogue-Link Implementation	to Context Awareness Platform / Orion Context Broker	Owner	Telecom Italia,Telefónica I+D,Boris Moltchanov (TI),Fermin Galan (TID)

4.1 Preface

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

4.1.1 Copyright

- Copyright © 2012 by [Telecom Italia](#), [Telefonica I+D](#)

4.1.2 Legal Notice

Please check the following [FI-WARE Open Specification Legal Notice \(implicit patents license\)](#) to understand the rights to use these specifications.

4.1.3 Overview

4.1.3.1 *Introduction to the Context Broker GE*

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. Changes in context information are therefore considered as events that can be handled by applications or FI-WARE GEs.

The Context Broker GE supports two ways of communications: push and pull towards both the Context Producer and the Context Consumer. It does mean that a Context Producer with a minimal or very simple logic may continuously push the context information into the Context Broker, when the information is available or due to the internal logic of the Context Producer. The Context Broker on its side can request the context information from Context Producers if they provide the ability to be queried (Context Producers able to act as servers are also referred as Context Providers). In a similar way, Context Consumers can pull the context information from the Context Broker (on-request mode), while the Context Broker can push the information to Context Consumer interested in it (subscription mode).

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 data without knowing which, where and when Context Consumers will consume published data; therefore they do not 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 event itself but not in who generated it. As a result, the Context Broker GE is an excellent bridge enabling external applications to manage events related to the Internet of the Things (IoT) in a simpler way hiding the complexity of gathering measures from IoT resources (sensors) that might be distributed or involving access through multiple low-level communication protocols.

4.1.3.2 ***Target usage***

The Context Broker is a GE of the FI-WARE platform that exposes the (standard) interfaces for retrieval of the context information, events and other data from the Context or Data/Event Producers to the Context or Data/Event Consumers. The consumer doesn't need to know where the data are located and what is the native protocol for their retrieval. It will just communicate to the Context Broker GE through a well-defined interface specifying the data it needed in a defined way: on request or on subscription basis. The Context Broker GE will provide the data back to the consumer when queried, in case of "on-request", or when available, in case of "on-subscription" communication mode.

4.1.3.3 ***Example Scenarios***

The number of potential context sources permanently connected through 3G links, e.g. mobile user terminals, embedded sensors, microphones and cameras, is expected to increase significantly in the coming years. 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].

A few examples of how usage of the Context Broker GE may improve the user experience and enrich a service are given below.

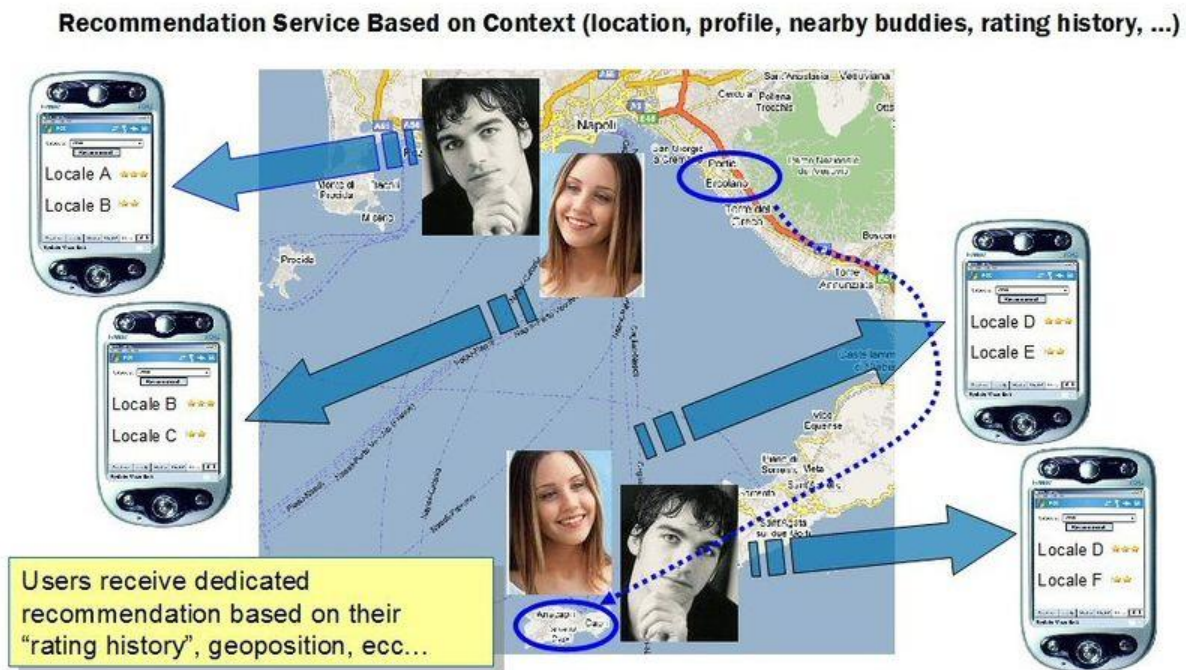
Next figure shows a context-aware advertising service (described in [3]) sending an invitation and a coupon to a customer in proximity to a boutique. Also the goods are chosen for that customer based on her/his preferences and previous acquisition. Therefore advertisement messages traffic is significantly reduced, targeting only potential clients, and the clients' user experience is not suffering from a "broadcast" of 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 some content (e.g. advertisement message and coupon) under certain conditions (the customer is close to the boutique and matching the preferences).



Example of context-aware advertising service

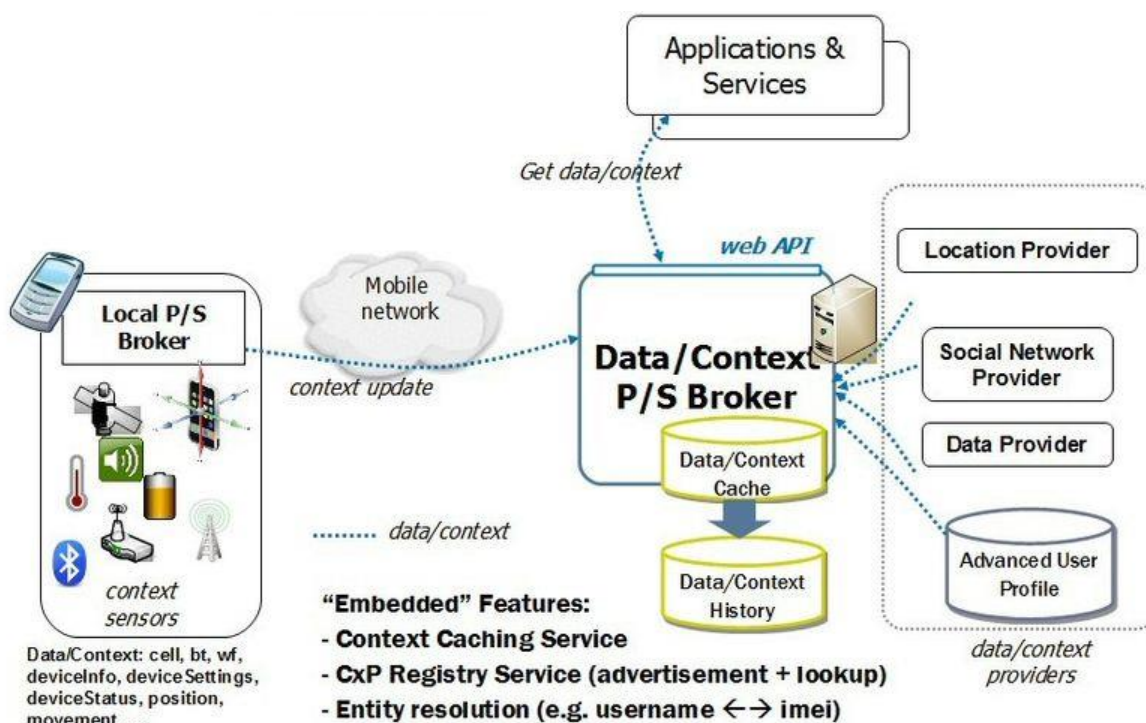
Another example might be context-aware content exchange shown in the figure below, where a customer sees only the content published by social friends (friends in a social network the customer is member of) only when this content is related to her location as well as when that content was originally "placed" in that location. Using interfaces provided by the Context Broker, the application used by the customer can subscribe to be informed based on recommendations related to her/his current location,

her/his preferences and coming from friends of the Social Networks the customer is member of. Recommendation would be handled as content information provided by recommender systems or individuals.



Example of context-aware content exchange

The following figure shows a logical architecture of the Context Broker GE with its main components and the basic information it handles to enrich traditional Mobile Advertisement and Content Share services.



Logical architecture of the Context Broker GE

4.1.4 Basic Concepts

All the communications between the various components of the Context Broker high-level architecture occur via two different interfaces/protocols, which are described in the following sections:

- ContextML/CQL built on top of HTTP in RESTlike way and allowing to publish or to retrieve context in a very simple, easy and efficient way, especially for mobile devices environments;
- NGSI RESTful interface inspired and based on the OMA NGSI specification [4]. This is a standard interface allowing to handle any type of data, which may include meta-data.

4.1.4.1 Context Elements

Aligned with the standard OMA NGSI specification, Context Information in FI-WARE is represented through generic data structures referred to as Context Elements. A Context Element refers to information that is produced, 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 to as context 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.

A Context Element typically provides information relevant to a particular entity, being it a physical thing or part of 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. That's why they typically contain an EntityId and an EntityType uniquely identifying the entity. Finally, there may be meta-data (also referred to as semantic data) linked to attributes in a context element. However, the existence of meta-data linked to a context element attribute is optional.

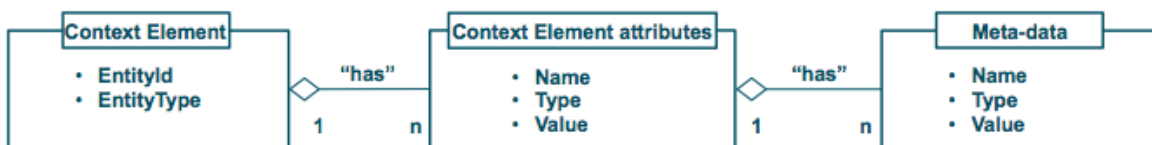
In summary, context information in OMA NGSI is represented through data structures called context elements, which have associated:

- 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)

As an example, we may consider the context element linked to updates on:

- attributes “speed”, “geolocation”, “current established route” of a “car”, or
- attributes “message geolocation”, “message contents” of a “user”

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



A cornerstone concept in FI-WARE is that context elements are not bound to a 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 database like MongoDB.

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

4.1.4.2 **Basic Actors in the Context Broker GE Model**

(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 actors. Primarily the CB has to control context flow among all attached actors; in order to do that, the CB has to know every Context Provider (CP) in the architecture; this feature is done through an announcement process detailed in the next sections.

Typically, the CB provides a Context Provider Lookup Service, a Context Cache and a Context History Service.

(2)Context Provider

A *Context Provider (CP)* is an actor that provides context information on demand, in synchronous mode; that means that the Context Broker or even the Context Consumer can invoke the CP in order to acquire context information. A CP provides context data only further to a specific invocation. Moreover, a CP can produce new context information inferred from the computation of input parameters; hence it is many times responsible for reasoning on high-level context information 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.

(3)Context Source

A *Context Source (CS)* spontaneously updates context information, about one or more context attributes or scopes. A CS sends context information according to its internal logic and does not expose the same interfaces as the 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, from the CS to the CB.

(4)Context Consumer

A *Context Consumer (CC)* is an entity (e.g. a context based application) that exploits 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 by subscribing to context information updates that match certain conditions (e.g., are related to certain set of entities). The CC registers a call-back operation with the subscription for the purpose, so the CB notifies the CC about relevant updates on the context by invoking this call-back function.

(5)Entity

Every exchange of context data here 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, things or group of things, etc.), which context data refer to. It is composed of two parts: a type and an identifier. Every Context Provider supports one or more entity types and this information is published to the Context Broker during an announcement process described later.

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

- human users - identified by *username*;
- mobile devices - identified by *imei*;
- mobile users - identified by *mobile* (GSM phone number);
- SIP accounts - identified by *SIP uri*;
- groups of other entities - identified by *groupid*;

The entity identifier specifies a particular item in a set of entities belonging to the same type. Every human user of the context management platform could be identified by multiple means and not just based on the username. That means that a process that provides identity resolution could be 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 entities that are mobile users identified by the type *mobile*. When the CB receives a *getContext* request about location of a human user, therefore identified by *username*, 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, identifiers of entities it knows how to process. 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 connected 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* related to the user; afterwards, the CB could invoke the location provider using the obtained entity and could send response with location data to the requester.

(6)Context scopes

Context information attributes managed by the Context Broker can be defined as part of a “scope”, which is a set of closely related context attributes. Every context attribute has a name and belongs to only one scope. Using a scope in operation exported or invoked by a Context Broker 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 data within a scope is always atomic so data associated to attributes 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 parameters 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 is changed, and vice versa is obviously not correct.

4.1.4.3 ***Advanced Features and Functionalities***

(1)Context caching

Context information 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 context information to the Context Broker, it can be retrieved from the cache, unless entries in the cache have expired (see next Chapter 3.4). This way, the Context Broker does not need to invoke the same Context Provider again and context delivery speeds up.

(2)Context validity

Any scope used during exchange of context information is tagged with a timestamp and expiration 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 a registered Context Provider.

(3)Context history

Every context information exchanged between the Context Broker and Context Providers or Context Sources is logged in the context history. The context history is different from the context cache, which stores only currently valid information (i.e., current values of attributes associated to context entities). The context history makes past context information about an entity also 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.

4.1.4.4 ***FI-WARE NGSI Specification***

Most of this GE's API operations regarding Events/Context 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 FI-WARE NGSI API description included in the present document will vary to be aligned with the final FI-WARE NGSI API specifications. FI-WARE NGSI specifications differ from the OMA NGSI specifications mainly in the binding, as far as OMA doesn't have any binding by definition. However FI-WARE NGSI improves some of the OMA NGSI aspects, which could be improved in the OMA as well, and finally, probably not all the mandatory/optional definitions will be

respected in the FI-WARE binding. Therefore the FI-WARE NGSI is mainly the technological binding for the OMA specifications with very little omits and differences.

4.1.5 Main Interactions

4.1.5.1 ***Using 'FI-WARE NGSI API' to interact with the Context Broker GE***

(1)Notions about OMA NGSI Specs

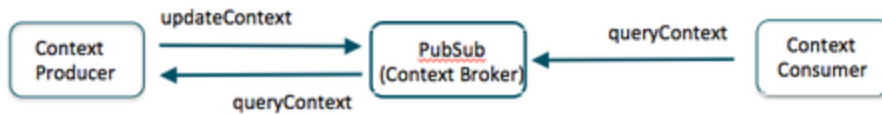
OMA NGSI (Next Generation Service Interface) Operations are grouped into two major interfaces:

- NGSI-10
 - updateContext
 - queryContext
 - subscribeContext / unsubscribeContext / updateContextSubscription
 - notifyContext
- NGSI-9
 - registerContext
 - discoverContextAvailability
 - subscribeContextAvailability / unsubscribeContextAvailability / updateContextAvailabilitySubscription
 - notifyContextAvailability

The FI-WARE specification is an evolution/modification proposal of the OMA NGSI specification which aims to maximize the role of NGSI for massive data collection from the IoT world, where a myriad of IoT resources are providing context elements occurrence/updates involving low level protocols. In other words FI-WARE NGSI is mainly the binding over OMA NGSI, however some small differences, out of scope of this documents, have been implemented in FI-WARE NGSI with respect to OMA specification.

(2)Basic Interactions and related Entities

The following diagram depicts the basic interactions of the Context Broker GE with its natural counterparts, that is the Context Producers and the Context Consumers.

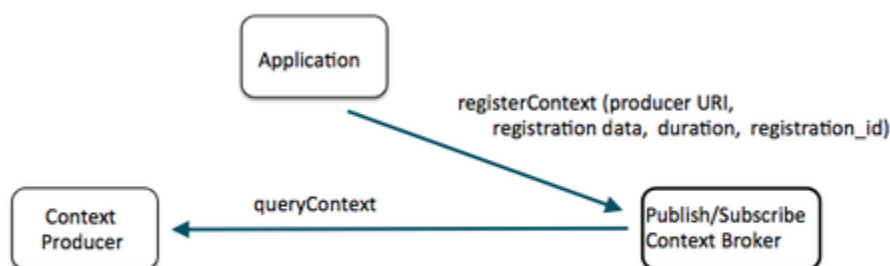


- Context Producers publish data/context elements by invoking the `updateContext` operation on a Context Broker.
- Some Context Producers may also implement a `queryContext` method. Context Brokers may invoke it at any given time to query on values of a designated set of attributes linked to a given 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 or Event Notification standards.

(3) Interactions related to query-able Context Producers

Context Producers publish data/context elements by invoking the `updateContext` operation on a Context Broker. Some Context Producers may also 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

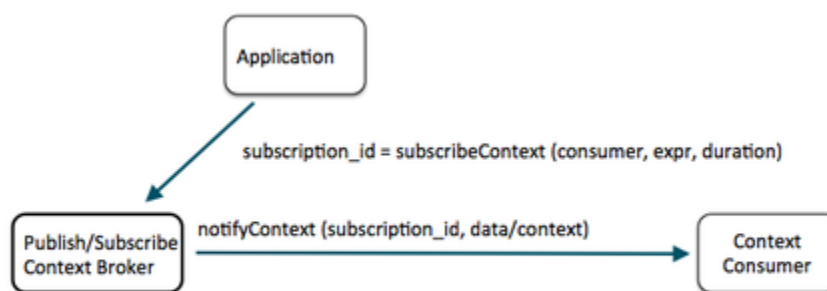
- Context Consumers can retrieve data/context elements by invoking the `queryContext` operation on a Context Broker
- Context data is kept persistent by Context Brokers and ready to be queried while not exceeding a given expiration time. This is a distinguishing feature of the OMA Context Management model as compared to some Event Brokering standards.



(4) Interactions to force Context Consumers to subscribe 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. A duration may be assigned the subscriptions.

- 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 Context Consumer itself



(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 (Context Providers) but also to register existence of entities in the system and availability of attributes.

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



(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. They do so by means of using the subscribeContextAvailability operation a Context Broker may export. A duration may be assigned to the subscriptions.

- 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



4.1.5.2 Using ContextML to interact with the Context Broker 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>? \[<OTHER_PARAMETERS>\]](http://<SERVER>/<MODULE>/<INTERFACE>/<OPERATION>/<RESOURCE>? [<OTHER_PARAMETERS>])

The returned data are formatted according to the Context Management Language (ContextML) proposed for this architecture.

(1) ContextML Basics

'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 enabling Context Providers to register themselves to the Context Broker. It also has commands and enabling 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
- **'ctxAdvs'**: 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

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 figure below shows ContextML context data provided from a Context Provider 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>

```

civilAddress Scope Example

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`

(2)ContextML API

A description of available methods and examples can be found in [ContextML API](#).

(3)ContextQL (CQL)

ContextQL or CQL [8] is an XML-based language allowing subscriptions to the Context Broker by scope conditions and rules consisting of more than 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) request. 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 ContextQL matching or conditional rules.

Context Query

A context query allows to send to the Context Broker a complex request consisting of many rules with conditions and matching parameters over the data available to the broker in real-time (including 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 two 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

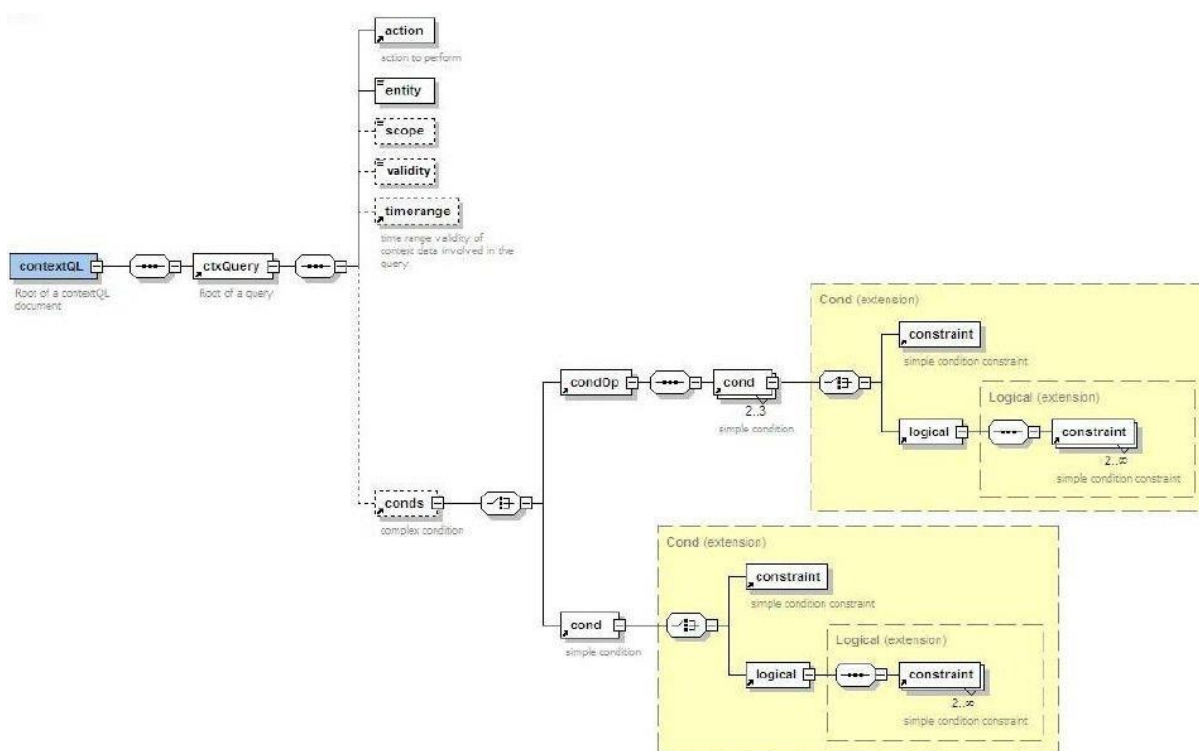
The following actions can be represented in CQL:

- **SELECT** – allow to request to broker the context information regarding certain entity and certain scope matching certain conditions. A wildcard 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 a unique *subId*, which 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** – conditions 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 SUBSCRIBE actions



XSD schema of a ContextML query

The *conds* tag may contain one or more conditions for any condition type. If there are more than

conditions elements they shall be linked *condOp*. The following table indicated the conditions combinations of different types that can be handled by the broker.

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

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

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* – identifies 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 exists only in the conditions ONVALUE. Currently defined attributes are of arithmetic and string-based types, which are listed in the below table

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

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.

(4)CQL API

A description of Context Query Language with some examples can be found in [CQL API](#)

4.1.6 Basic Design Principles

4.1.6.1 *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 supported:

- **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.).

4.1.7 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.

4.1.8 Detailed Specifications

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

4.1.8.1 *Open API Specifications*

- [FI-WARE NGSI Open RESTful API Specification](#)
- RESTlike over HTTP [ContextML API](#) Open Specification
- RESTlike over HTTP Context Query Language [CQL API](#) Open Specification

4.2 Re-utilised Technologies/Specifications

The following technologies are used by the Context Broker:

- OMA NGSI
- RESTful web services
- HTTP/1.1
- JSON and XML data serialization formats

4.3 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 **avalue**. 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 FI-WARE NGSI-9 Open RESTful API Specification

5.1 Introduction to the FI-WARE NGSI-9 API

5.1.1 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 (also called 'agents' here) 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)

5.1.2 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 the large number of FI-WARE Generic Enable that expose NGSI interfaces. 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 official OMA NGSI-9/NGSI-10 specification [2] and FI-WARE NGSI binding documents [5] for details on the resource structure and message formats.

5.1.3 Change history

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

Revision Date	Changes Summary

July 14, 2012	<ul style="list-style-type: none">• 1st stable version
May 12, 2014	<ul style="list-style-type: none">• Slightly updated the descriptions to the current state of FI-WARE
May 26, 2014	<ul style="list-style-type: none">• Addressed minor peer review comments

5.1.4 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 from the list of [publicly available documents](#).

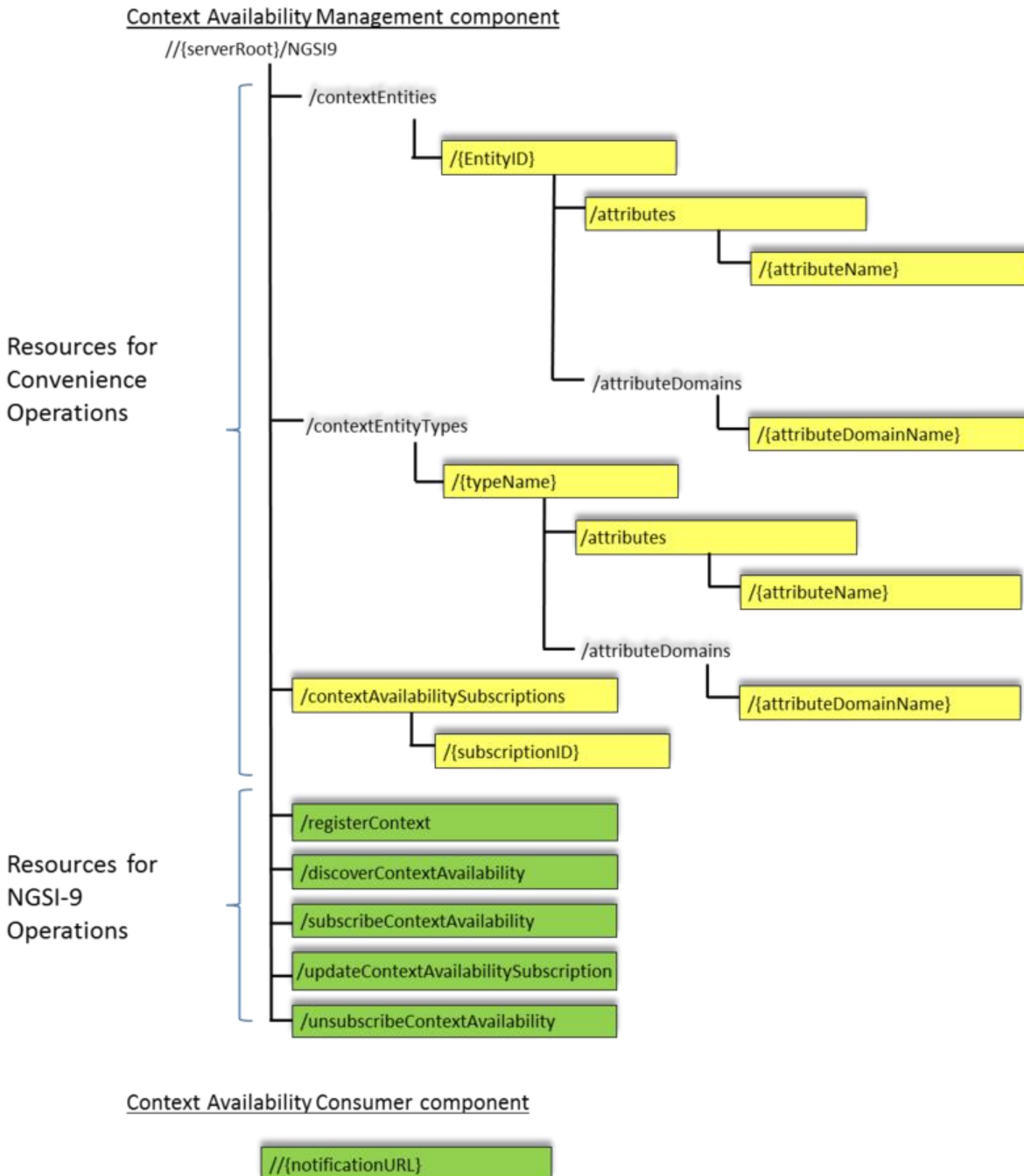
OMA NGSI-10 and OMA NGSI-9 share the same [NGSI-9/NGSI-10 information model](#). Be sure to have read it before continuing on this page.

5.1.5 Legal Notice

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

5.2 General NGSI-9 API information

5.2.1 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 HTTP operations (GET PUT, POST, and DELETE).

The convenience functions 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 that are part of the binding.

5.2.2 Representation Format

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

5.2.3 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.

5.2.4 API Operations on Context Management Component

5.2.4.1 *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	/discoverContextAvailability	Generic discovery of context information

resource		providers. The expected request body is an instance of <code>discoverContextAvailabilityRequest</code> ; the response body is an instance of <code>discoverContextAvailabilityResponse</code> .
Availability subscription resource	<code>/subscribeContextAvailability</code>	Generic subscription to context availability information. The expected request body is an instance of <code>subscribeContextAvailabilityRequest</code> ; the response body is an instance of <code>subscribeContextAvailabilityResponse</code> .
Availability subscription update resource	<code>/updateContextAvailabilitySubscription</code>	Generic update of context availability subscriptions. The expected request body is an instance of <code>updateContextAvailabilitySubscriptionRequest</code> ; the response body is an instance of <code>updateContextAvailabilitySubscriptionResponse</code> .
Availability subscription deletion resource	<code>/unsubscribeContextAvailability</code>	Generic deletion of context availability subscriptions. The expected request body is an instance of <code>unsubscribeContextAvailabilityRequest</code> ; the response body is an instance of <code>unsubscribeContextAvailabilityResponse</code> .

5.2.4.2 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	/contextEntities/{EntityID}/attributes/{attributeName}	Retrieve information	-	Register a provider of	-

context entity		on providers of the attribute value		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 entities of the type	-	Register a provider of information about context entities of the type	-
Attribute container of entity type	/contextEntityTypes/{typeName}/attributes	Retrieve information on providers of any information	-	Register a provider of information about context	-

		about context entities of the type		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	/contextAvailabilitySubscriptions	-	-	Create a new	-

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

5.2.5 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.

- [1] https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/NGSI_9/10_information_model
- [2] http://technical.openmobilealliance.org/Technical/release_program/docs/NGSI/V1_0-20120529-A/OMA-TS-NGSI_Context_Management-V1_0-20120529-A.pdf
- [3] https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI-9_Open_RESTful_API_Specification_%28PRELIMINARY%29
- [4] https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI-10_Open_RESTful_API_Specification_%28PRELIMINARY%29
- [5] https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI_Open_RESTful_API_Specification
- [6] https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/NGSI_association
- [7] https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_Architecture
- [8] https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_Open_Specifications_Legal_Notice
- [9] <http://edu.fi-ware.org/course/view.php?id=33>

6 FI-WARE NGSI-10 Open RESTful API Specification

6.1 Introduction to the FI-WARE NGSI 10 API

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

6.1.1 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)

6.1.2 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 the numerous FI-WARE Generic Enablers that expose NGSI interfaces. 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 [2] and binding documents [5] for details on the resource structure and message formats.

6.1.3 Change history

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

Revision Date	Changes Summary

May 14, 2012	<ul style="list-style-type: none">• 1st stable version
May 12, 2014	<ul style="list-style-type: none">• Slight changes to adopt to changes in FI-WARE.
May 26, 2014	<ul style="list-style-type: none">• Addressed minor peer review comments

6.1.4 Additional Resources

The formal specification of OMA NGSI 10 can be [downloaded](#) from the website of the [Open Mobile Alliance](#).

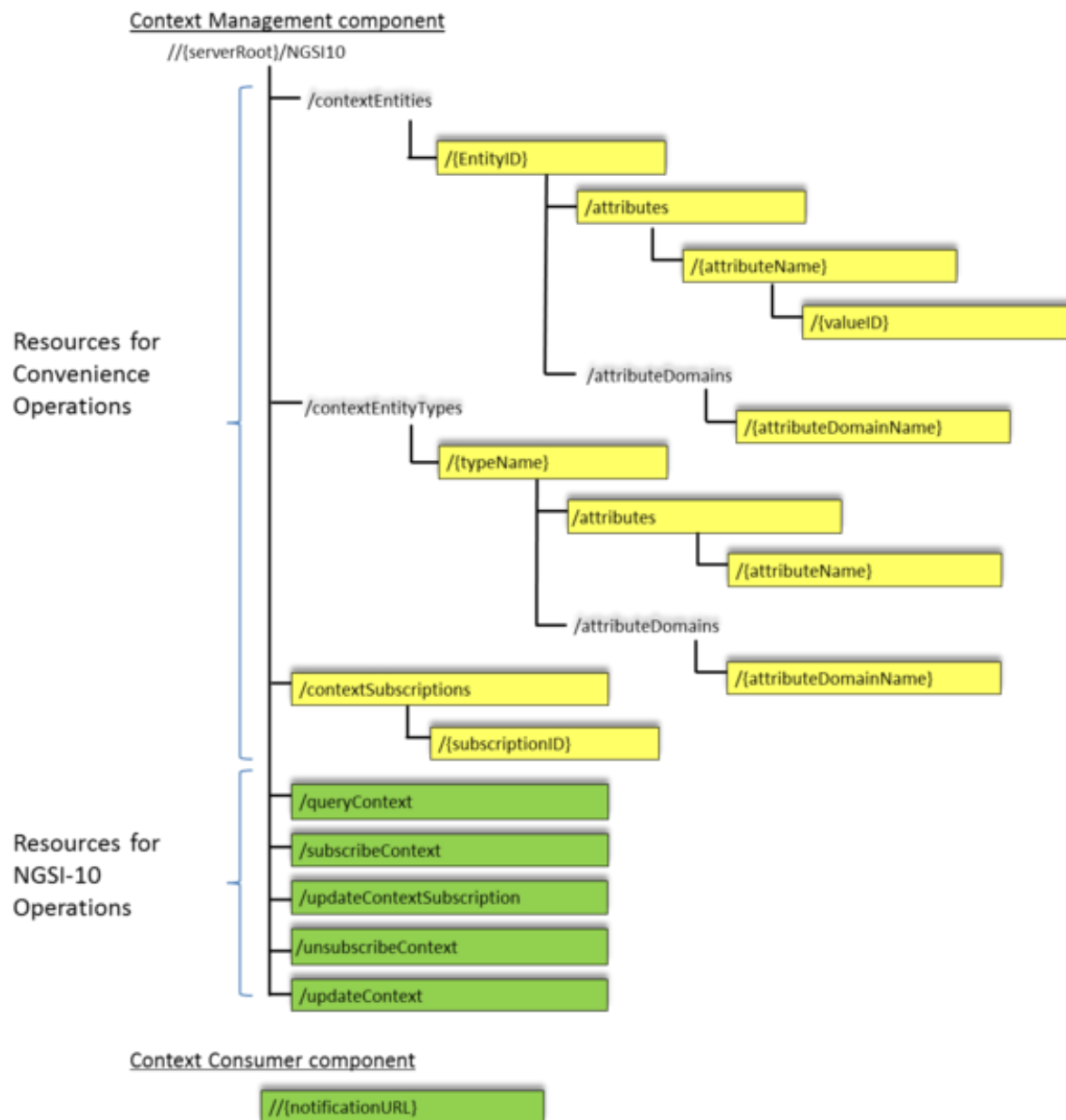
The FI-WARE RESTful binding of OMA NGSI-10 described on this page has been defined by the FI-WARE project. It can be accessed from the list of [publicly available documents](#).

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.

6.2 General NGSI 10 API information

6.2.1 Resources Summary

The following scheme shows the resources tree in NGSI-9/10 schema of REST resources.



Schema of REST resources in NGSI-9/10

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 HTTP 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.

6.2.2 Representation Format

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

6.2.3 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.

6.2.4 API Operations on Context Management Component

6.2.4.1 *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	/queryContext	Generic queries for context information. The expected request body is an instance of queryContextRequest; the response body is an instance of queryContextResponse.
Subscribe context resource	/subscribeContext	Generic subscriptions for context information. The expected request body is an instance of subscribeContextRequest; the response body is an instance of subscribeContextResponse.
Update context	/updateContextSubscription	Generic update of context subscriptions. The expected request body is an instance of

subscription resource		updateContextSubscriptionRequest; 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.

6.2.4.2 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 context entity	/contextEntities/{EntityID}/attributes/{attributeName}/{attributeID}	Retrieve specific attribute value	Replace attribute value	-	Delete attribute value
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

6.2.5 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.

6.3 References

- [1] https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/NGSI_9/10_information_model
- [2] http://technical.openmobilealliance.org/Technical/release_program/docs/NGSI/V1_0-20120529-A/OMA-TS-NGSI_Context_Management-V1_0-20120529-A.pdf
- [3] https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI-9_Open_RESTful_API_Specification_%28PRELIMINARY%29
- [4] https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI-10_Open_RESTful_API_Specification_%28PRELIMINARY%29

- [5] https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI_Open_RESTful_API_Specification
- [6] https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/NGSI_association
- [7] https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_Architecture
- [8] https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_Open_Specifications_Legal_Notice
- [9] <http://edu.fi-ware.org/course/view.php?id=33>

7 ContextML API

7.1 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.

7.2 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

7.2.1 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>

```

7.2.2 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*

7.3 ContextML API

In the following paragraphs a description of the available method is given.

7.3.1 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>

```

7.3.2 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=[scopeName]`

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">mobile2.tilab.com/LP/loc/getLocation</par>
        </parS>
      </parA>
    </ctxPrvEl>
  </ctxPrvEls>
</contextML>
```

7.3.3 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>

```

7.3.4 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">7.6752519444444445</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"/>

  </ctxResp>

</contextML>
```

```

    <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>

```

7.3.5 Get context

This method allows the retrieval of context elements from the platform, which 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">7.6752519444444445</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>
```

8 CQL API

8.1 ContextQL (CQL)

ContextQL is an XML-based language allowing to subscribe to the Publish/Subscribe Context Broker GE (referred as "CB" in the following) by scope conditions and rules consisting of more than 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 CB 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.

8.1.1 Context Query

A context query allows to send to the CB a complex request consisting of many rules with conditions and matching parameters over the data available to the broker in real-time (including 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 two 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 wildcard 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. Validity is expressed in seconds and shall be set in all subscriptions.

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** – conditions 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

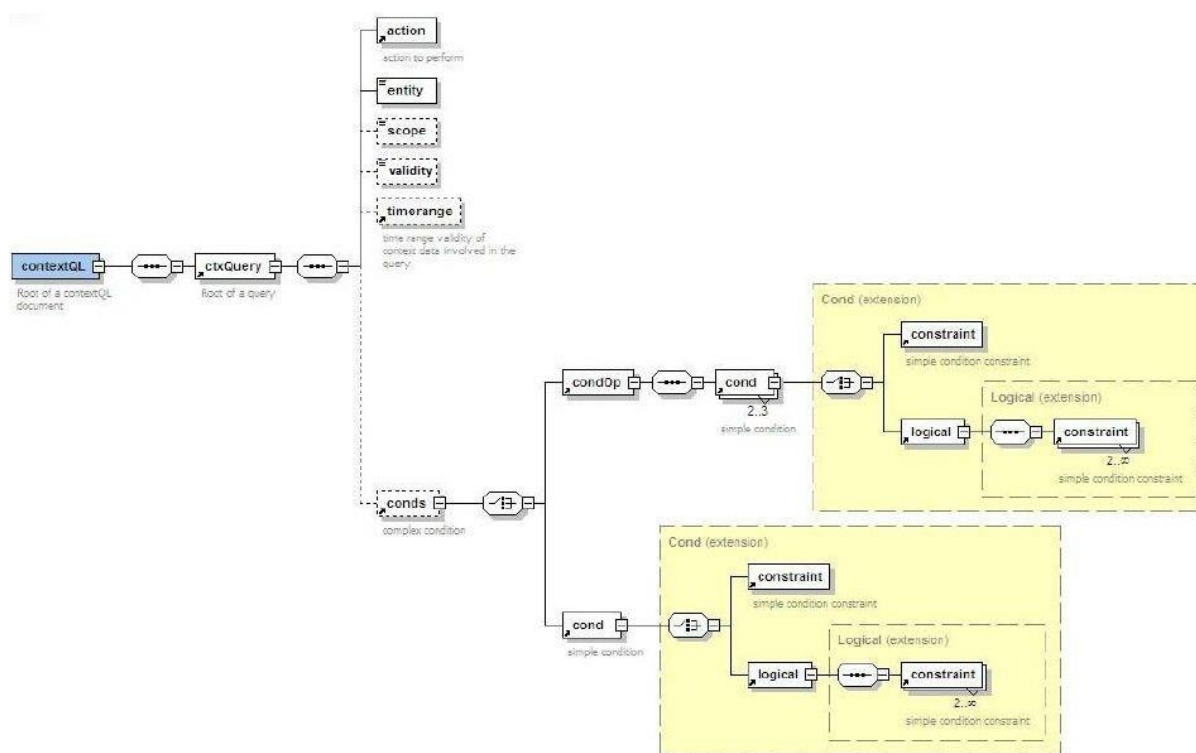


Figure: XSD schema of a ContextML query

The *conds* tag may contain one or more conditions for any condition type. If there are more than 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* – identifies 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.

8.2 CQL API

In the following paragraphs a description of the available methods is given. The aim of the examples is to show general capabilities of CQL language, therefore some features are not supported in current implementation of the Context Broker.

8.2.1 Examples of Context Queries

8.2.1.1 **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>
  <ctxQuery>
    <action type="SELECT" />
    <entity>username|sergio</entity>
    <scope>civilAddress</scope>
    <conds>
      <cond type="ONVALUE">
        <constraint          param="activity.status"          op="CONT"
value="with_friend"/>
      </cond>
    </conds>
  </ctxQuery>
</contextQL>
```

The answer is a ContextML message, as for a standard ContextML getContext request.

8.2.1.2 **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>
  <ctxQuery>
    <action type="SUBSCRIBE" />
    <entity>username|sergio</entity>
    <scope>position</scope>
    <validity>36000</validity>
    <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/1.7"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ContextML/1.7
http://contextml.tilab.com/ContextML-1.7.xsd">
  <ctxResp>
    <contextProvider id="CB" v="1.4.5"/>
    <timestamp>2014-03-06T12:43:35+02:00</timestamp>
```

```

<method>getContextQL</method>

<resp status="OK" code="200"/>

<validity>10000</validity>

<subId>86</subId>

</ctxResp>
</contextML>

```

8.2.1.3 **SUBSCRIBE ONCLOCK**

The SUBSCRIBE ONCLOCK allows to subscribe to context data notification which are sent at a specific time interval.

```

<contextQL>
  <ctxQuery>
    <action type="SUBSCRIBE" />
    <entity>imei|358361005978493</entity>
    <scope>deviceStatus</scope>
    <validity>3600</validity>
    <conds>
      <cond type="ONCLOCK">
        <constraint value="300"/>
      </cond>
    </conds>
  </ctxQuery>
</contextQL>

```

8.2.1.4 **SUBSCRIBE ONCLOCK/ONCHANGE**

Also subscription on mixed conditions could be specified in CQL (not supported in current release):

```

<contextQL>
  <ctxQuery>

```

```

<action type="SUBSCRIBE" />
<entity>imei|358361005978493</entity>
<scope>cell</scope>
<validity>3600</validity>
<conds>
  <condOp op="OR">
    <cond type="ONCLOCK">
      <constraint value="300"/>
    </cond>
    <cond type="ONCHANGE">
      <constraint param="cell.cgi"/>
    </cond>
  </condOp>
</conds>
</ctxQuery>
</contextQL>

```

8.2.1.5 ***SELECT COUNT***

The SELECT COUNT returns the number of entities which have context data in cache verifying the required condition (timerange parameter is not supported in current release):

```

<contextQL>
  <ctxQuery>
    <action type="SELECT" option="COUNT" />
    <entity>username|*</entity>
    <timerange      from="2007-10-24T12:00:00+02:00"      to="2007-10-
25T12:00:00+02:00" />
    <conds>
      <cond type="ONVALUE">

```

```

        <constraint param="civilAddress.city" op="EQ" value="Torino"/>
    </cond>
</conds>
</ctxQuery>
</contextQL>

```

The response is as follows:

```

<?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
http://contextml.tilab.com/ContextML-1.7.xsd">

    <ctxResp>

        <contextProvider id="CB" v="1.4.5"/>

        <timestamp>2014-03-06T16:13:34+02:00</timestamp>

        <method>getContextQL</method>

        <resp status="OK" code="200"/>

        <dataPart>

            <count>4</count>

        </dataPart>

    </ctxResp>

</contextML>

```

8.2.1.6 ***SELECT LIST***

The SELECT LIST returns the list of entities which have context data in cache verifying the required condition:

```

<contextQL>

    <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:

```

<?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
http://contextml.tilab.com/ContextML-1.7.xsd">

  <ctxResp>

    <contextProvider id="CB" v="1.4.5"/>

    <timestamp>2014-03-06T16:16:24+02:00</timestamp>

    <method>getContextQL</method>

    <resp status="OK" code="200"/>

    <dataPart>

      <parA n="entities">

        <par n="entity">username|cristina</par>

      </parA>

```

```
</dataPart>  
  
</ctxResp>  
  
</contextML>
```

9 FIWARE OpenSpecification Data CEP

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

9.1 Preface

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

9.2 Copyright

- Copyright © 2012 by [IBM](#)

9.3 Legal Notice

Please check the following [Legal Notice](#) to understand the rights to use these specifications.

9.4 Overview

9.4.1 Introduction to the CEP GE

The Complex Event Processing (CEP) GE is intended 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 non-functional requirements without taking a toll on the application performance, removing one issue from the application developer's and system managers concerns.

9.4.2 Basics of CEP

Entities connected to the CEP GE (application entities or some other GEs like the Context 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. Events generated by such sensors 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 Context Broker GE, having the CEP acting as a context consumer of the Context Broker GE.

Event Producers can provide events in two modes:

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

Event Consumers are the destination 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 Context Broker GE: which can connect as an event consumer to the CEP and forward 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). An EPN is made up of processing nodes 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 dealing 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 scalability, performance and optimization requirements may be addressed by design.

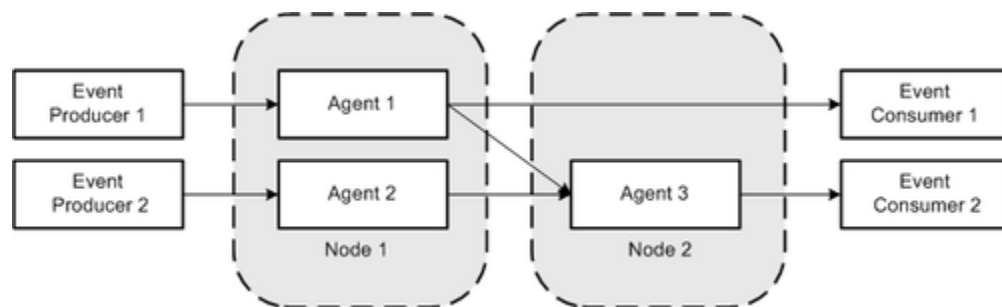


Illustration of an Event Processing Network made of event producers, agents and event consumers

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.

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 of 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 supported in each part of the rule language.

9.4.2.1 *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 (optional) additional 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 bought 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, as described in [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.

9.4.2.2 *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

9.4.2.3 **Actions**

As part of the rule definition, the application developer of CEP specifies what should be done when a rule is detected. This can include generation of one or more derived events to be sent to the consumers and actions to be performed by the consumers. These actions definitions include the parameters needed for their execution.

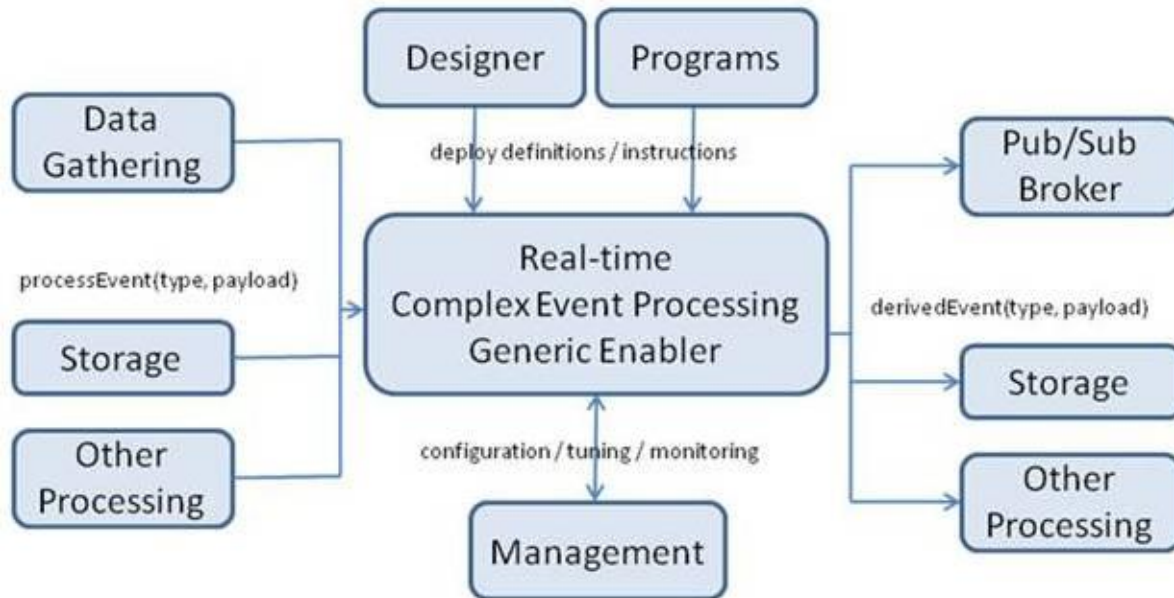
9.5 Target Usage

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 Context Broker 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 that reacting or making use of the event is meaningful for the consumer/subscriber). 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 specification and definitions programmatically, by an application or tools. 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 supports the roles of event system manager and operator, which could be played either by real people or management components. Actors playing these roles are responsible for managing configurations, monitor processing performance, handling problems, and monitoring the system's health. They make use of the pseudo API *configuration/tuning/monitoring* for this purpose.



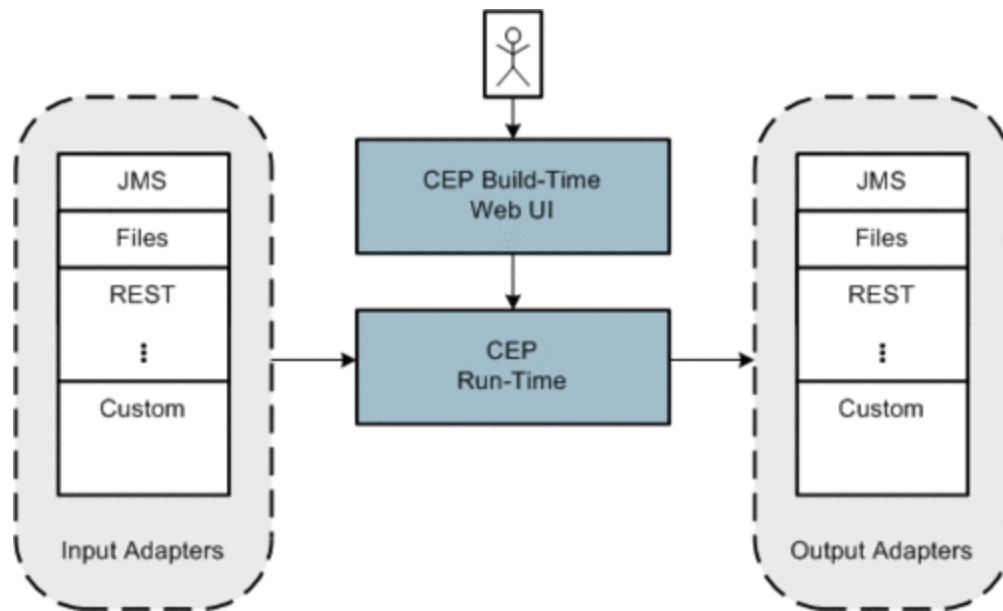
Interactions with and APIs of the Real-time CEP Generic Enabler

9.6 Basic Concepts

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

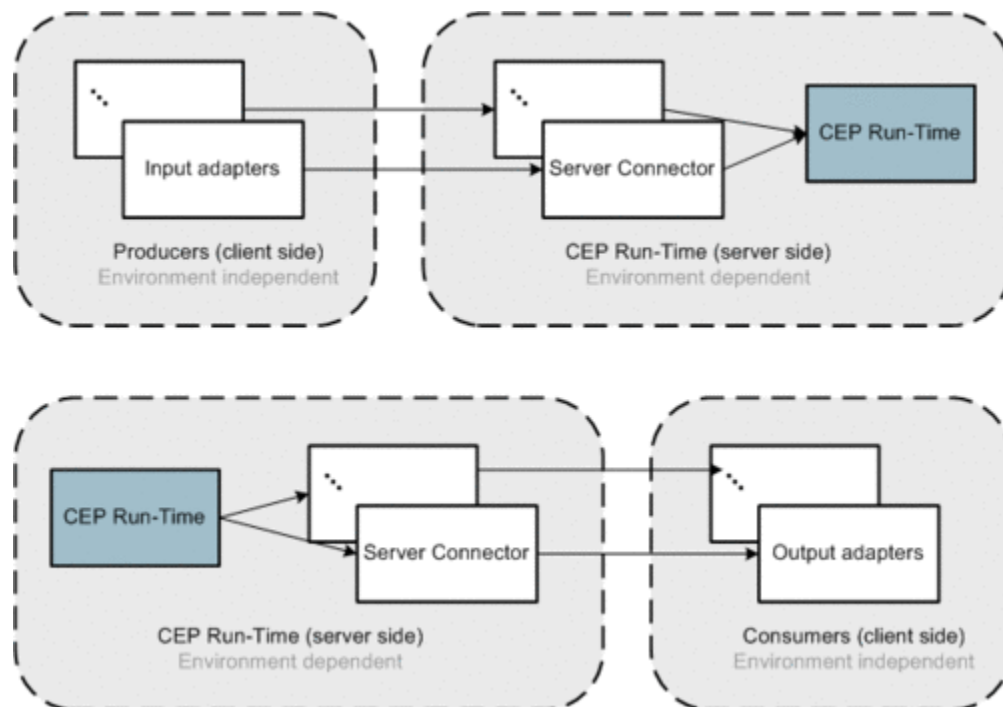
1. Input Adapters and REST service for getting incoming events
2. Output Adapters for sending derived events
3. CEP Application definition
4. Administrative REST services

The application definitions, including the EPN, can be 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 build time are translated into input and output adapters at 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, an input adapter is defined for each producer, which defines how to pull the data from the source, 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 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, allowing them to be environment-agnostic.

9.6.1 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 producers includes the resource from which the adapter pulls the information (whether this resource is a file in a file system, a JMS queue or a REST service). It also includes format settings which allow the adapter to transform the resource-specific information into a CEP event data object. The formatting depends on the kind of resource we are dealing with – for files it can be a tagged file formatter, for JMS an object transformer. Likewise, the specification of consumers 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 the specifications of the source of the event data and 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 holds the specifications of the target of the events and the format of the 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.
- All the input adapters implement a common interface, which is extendable to cover custom input adapter types and allows adding new producers for custom-type resources.
- All the output adapters implement a common interface, which is extendable to cover custom output adapter types and allows adding 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 pulling the information from designated resource according to its specifications, each time processing the incremental additions in the resource. However, producers operating 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 its desirable events list.

9.6.2 Definition of CEP Application

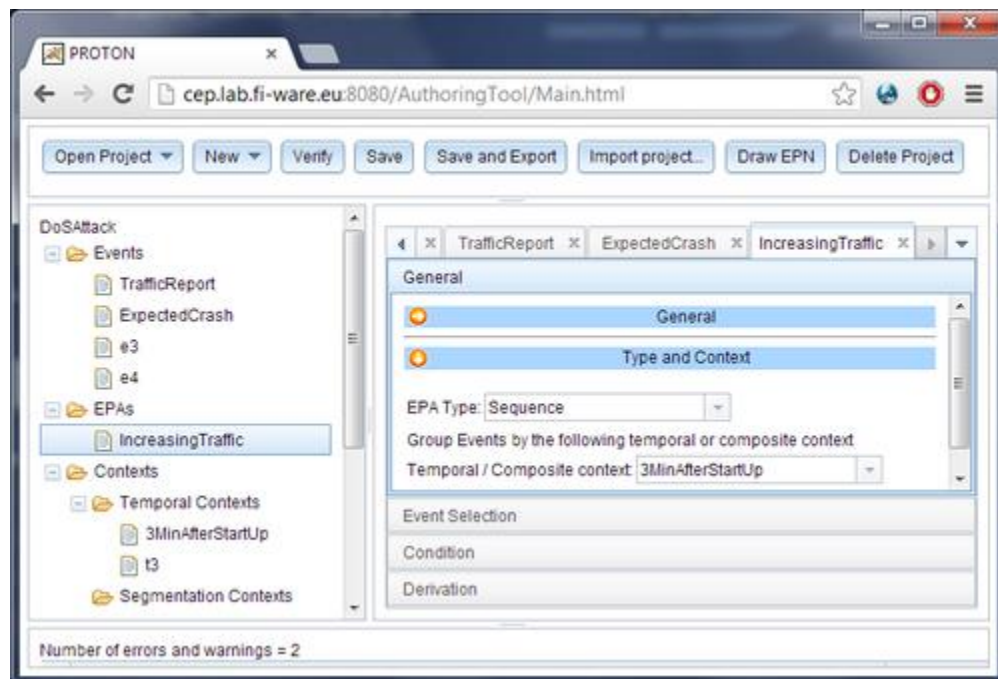
A CEP definition file is created using the CEP build-time web based user interface. Using this UI, the application developer creates the building blocks of the application definitions. This is done by filling up forms without the need to write any code.

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 – responsible of applying rules on incoming events in specific context as to detect situations and generate derived events.

The UI (see a figure below) provides many functions, including defining a CEP application, examining the event processing network of this application, validating it and, exporting the event processing network definition. The export action creates a JSON format representation of the CEP application. This JSON can be exported either to the engine repository or to a local file (to be later fed to the engine repository).

This JSON file that describes a CEP application, is the input for the CEP engine. It can be generated by the UI, and it can be generated programmatically by any other application and fed to the CEP engine. The format of this definition file is described by a [JSON schema for a CEP application](#) while the semantics of the various elements in this schema are described in a [user guide](#)



CEP Web based User interface for application definition

9.6.3 Administrative REST services

There are REST services that allow managing the CEP definition repository that holds the CEP application definitions available to the CEP engine instances at run time. These services allow putting a new definition file to the repository, getting a specific definition from the repository, updating a repository definition file or deleting a definition from the repository. In addition, there are REST services that allow controlling the CEP engine instances at run time. These services allow starting and stopping a CEP engine instance, updating CEP engine instance definitions and reading the state of the CEP engine instance (started/stopped and its definition url).

9.7 Basic Design Principles

- The EPN application definition can be done using a user interface without the need to write any code, with intention for visual programming.
- The CEP application is composed of a network of Event Processing Agents. This allows the agents to run in parallel and to be distributed on several machines.
- Logical 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.

9.8 References

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

9.9 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.

9.9.1 Open API Specifications

- [Complex Event Processing Open RESTful API Specification](#)

9.10 Re-utilised Technologies/Specifications

The CEP authoring tool is running on Apache Tomcat web server

9.11 Terms and definitions

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

- **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 **avalue**. 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.

10 Complex Event Processing Open RESTful API Specification

10.1 Introduction to the CEP GE REST API

As described in the CEP GE open specification document [FIWARE.ArchitectureDescription.Data.CEP](#), CEP has three main interfaces: one for receiving raw events from event producers using a RESTful service, second for sending output events to event consumers using an output REST client adapter, and a third for administrating the CEP engine state, and its definition repository. Following are detailed descriptions and examples of the APIs since the 2nd release of the CEP GE.

Please check the [Legal Notice](#) to understand the rights to use FI-WARE Open Specifications.

10.1.1 The CEP APIs

The CEP supports RESTful, resource-oriented APIs accessed via HTTP for:

1. Receiving events in JSON-based, tag-delimited or XML-NGSI format via a provided service using POST
2. Sending events in JSON-based, tag-delimited or XML-NGSI format via a REST client using POST
3. Administrating CEP via a provided service for:
 1. Managing the definitions repository (adding, replacing and deleting definitions)
 2. Managing engine instances (changing their definitions and start\stop the instances)

10.1.2 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, tag-delimited, or XML/NGSI data serialization formats

10.1.3 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	<ul style="list-style-type: none">• Initial Version
Apr 30, 2013	<ul style="list-style-type: none">• 2nd Release Version
Apr 30, 2014	<ul style="list-style-type: none">• 3rd Release Version

10.1.4 How to Read This Document

The assumption is that a reader is familiarized with REST architecture style. For descriptions of terms used in this document see [FIWARE.ArchitectureDescription.Data.CEP](#).

10.1.5 Additional Resources

For more details on the CEP GE Adapters and Architectural Description of the CEP GE please refer to [FIWARE.ArchitectureDescription.Data.CEP](#).

10.2 General CEP API Information

10.2.1 Resources Summary

- The CEP GE provides a REST service allowing for external systems to push events using the POST method.

http://{server}:{port}/{instance_name}/rest/events

- The CEP GE uses a REST output adapter which is a client that activates a REST service using the POST method.
- The CEP GE provides a REST service for administrating the generic enabler. It allows for managing the definitions repository

http://{server}:{port}/{CEP_Admin}/resources/definitions/{definition_name} and the engine instances http://{server}:{port}/{CEP_Admin}/resources/instances/{instance_name}

10.2.2 Representation Format

For incoming and outgoing events, CEP GE supports JSON-based, tag-delimited or XML-NGSI format.

The receiving event service will accept either formats automatically, when the content-type is provided. Use `application/json` Content-Type for JSON events, `text/plain` Content-Type for tag-delimited events and `application/xml` Content-Type for XML-NGSI events.

The client for notifying on events using the REST output adapter is configured with one of the formats when defining the consumer in the Event Processing Network (using the authoring tool). This will define the Content-Type header of the request to be issued.

10.2.3 Representation Transport

Resource representation is transmitted between client and server by using the 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 the wrapped representation. In addition, both client and server may use as many HTTP headers as they consider necessary.

10.3 API Operations

In this section we describe each operation in-depth for each provided resource.

10.3.1 Receiving Events API

The CEP GE provides a REST service for receiving events. (Pulling events from an externally provided REST service, as was supported in Release 1, still exists as an input adapter and details for using this adapter are within the programmer guide)

Verb	URI example	Description
POST	<code>/instance_name/rest/events</code>	Receive events by the specified engine instance

10.3.1.1 Usage Examples

In tag-delimited format:

```
POST //localhost:8080/ProtonOnWebServer/rest/events
Content-Type: text/plain
```

```
Name=TrafficReport;volume=1000;
```

In JSON format:

```
POST //localhost:8080/ProtonOnWebServer/rest/events
Content-Type: application/json
{"Name":"TrafficReport", "volume":"1000"}
```

In XML/NGSI format:

```
POST //localhost:8080/ProtonOnWebServer/rest/events
Content-Type: application/xml
<notifyContextRequest>
  <subscriptionId>51a60c7a286043f73ce9606c</subscriptionId>
  <originator>localhost</originator>
  <contextResponseList>
    <contextElementResponse>
      <contextElement>
        <entityId type="Node" isPattern="false">
          <id>OUTSMART.NODE_3505</id>
        </entityId>
        <contextAttributeList>
          <contextAttribute>
            <name>TimeInstant</name>
            <type>urn:x-ogc:def:trs:IDAS:1.0:ISO8601</type>
            <contextValue>2013-05-31T18:59:08+0300</contextValue>
          </contextAttribute>
          <contextAttribute>
```



```
<name>presence</name>

<type>urn:x-ogc:def:phenomenon:IDAS:1.0:presence</type>

<contextValue></contextValue>

</contextAttribute>

<contextAttribute>

  <name>batteryCharge</name>

  <type>urn:x-ogc:def:phenomenon:IDAS:1.0:batteryCharge</type>

  <contextValue>2</contextValue>

</contextAttribute>

<contextAttribute>

  <name>illuminance</name>

  <type>urn:x-ogc:def:phenomenon:IDAS:1.0:illuminance</type>

  <contextValue></contextValue>

</contextAttribute>

<contextAttribute>

  <name>Latitud</name>

  <type>urn:x-ogc:def:phenomenon:IDAS:1.0:latitude</type>

  <contextValue></contextValue>

</contextAttribute>

<contextAttribute>

  <name>Longitud</name>

  <type>urn:x-ogc:def:phenomenon:IDAS:1.0:longitude</type>

  <contextValue></contextValue>

</contextAttribute>

</contextAttributeList>

</contextElement>
```

```

    <statusCode>

      <code>200</code>

      <reasonPhrase>OK</reasonPhrase>

    </statusCode>

  </contextElementResponse>

</contextResponseList>

</notifyContextRequest>

```

Note:

Name is a built-in attribute used to represent the event type being reported. Please consult the user guide for event representation and built-in attributes.

The data in the tag format should be given with no blanks.

In the JSON format, all the attributes values are given as strings, the CEP processes each attribute value according to its defined type (in the event definition).

10.3.2 Sending Events API

The CEP GE activates a REST client for sending output events (in a push mode).

Verb	URI example	Description
POST	/application-name/consumer	Send a derived event to a consumer

10.3.2.1 Usage Examples

The following is what the REST output adapter will generate as a request to a REST service called /application-name/consumer and is expected to be able to interpret either the tag-delimited, JSON or XML/NGSI format via the POST method.

Note: **"Name"** is a built-in attribute used to represent the event type being reported. Please consult the user guide for event representation and built-in attributes.

In tag-delimited format:

```

POST //localhost:8080/application-name/consumer

Content-type: text/plain

```

```
Name=TrafficReport;Certainty=0.0;Cost=0.0;EventSource=;OccurrenceTime=
null;Annotation=;Duration=0.0;volume=1000;
```

```
EventId=40f68052-3c7c-4245-ae5a-
6e20def2e618;ExpirationTime=null;Chronon=null;DetectionTime=1349181899
221;
```

In JSON format:

```
POST //localhost:8080/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"}
```

In XML/NGSI format:

```
POST //localhost:8080/application-name/consumer

Content-type: application/xml

<updateContextRequest>
  <contextElementList>
    <contextElement>
      <entityId type="CEPEventReporter" isPattern="false">
        <id>CEPEventReporter_Singleton</id>
      </entityId>
      <contextAttributeList>
        <contextAttribute>
          <name>EventId</name>
          <contextValue>4be0ab1c-ec30-4525-b278-
78222f3ce081</contextValue>
        </contextAttribute>
```

```
<contextAttribute>
  <name>EventType</name>
    <contextValue>LowBatteryAlert</contextValue>
</contextAttribute>
<contextAttribute>
  <name>DetectionTime</name>
    <contextValue>2013-06-
05T08:25:15.804000CEST</contextValue>
</contextAttribute>
<contextAttribute>
  <name>EventSeverity</name>
    <contextValue>Critical</contextValue>
</contextAttribute>
<contextAttribute>
  <name>Cost</name>
    <contextValue>0.0</contextValue>
</contextAttribute>
<contextAttribute>
  <name>Certainty</name>
    <contextValue>1</contextValue>
</contextAttribute>
<contextAttribute>
  <name>Name</name>
    <contextValue>LowBatteryAlert</contextValue>
</contextAttribute>
<contextAttribute>
  <name>OccurrenceTime</name>
```

```
                <contextValue>2013-06-
05T08:25:15.804000CEST</contextValue>

            </contextAttribute>

            <contextAttribute>

                <name>TimeInstant</name>

                <contextValue>2013-06-
05T08:24:45.581000CEST</contextValue>

            </contextAttribute>

            <contextAttribute>

                <name>Duration</name>

                <contextValue>0</contextValue>

            </contextAttribute>

            <contextAttribute>

                <name>AffectedEntityType</name>

                <contextValue>Node</contextValue>

            </contextAttribute>

            <contextAttribute>

                <name>AffectedEntity</name>

                <contextValue>OUTSMART.NODE_3505</contextValue>

            </contextAttribute>

        </contextAttributeList>

    </contextElement>

</contextElementList>

    <updateAction>UPDATE</updateAction>

</updateContextRequest>
```

10.3.3 Managing the Definitions Repository

The CEP GE provides a REST service for managing the definitions repository. The repository is a file directory. Adding or deleting a definition will add or remove a file from the directory respectively. Each definition is identified via a unique name (prefixed by the repository location) and a URI associated with it. The URI is used to retrieve the file by the applications that make use of the definition.

Verb	URI example	Description
GET	<code>//{CEP_Admin}/resources/definitions</code>	Retrieve all the existing definitions in the repository
POST	<code>//{CEP_Admin}/resources/definitions</code>	Add a new definition
GET	<code>//{CEP_Admin}/resources/definitions/{definition_name}</code>	Retrieve the complete definition in JSON format
PUT	<code>//{CEP_Admin}/resources/definitions/{definition_name}</code>	Replace content of an existing definition with new content
DELETE	<code>//{CEP_Admin}/resources/definitions/{definition_name}</code>	Remove the definition from the repository

10.3.3.1 Usage Examples

- Retrieving all definitions

```
GET //localhost:8080/ProtonOnWebServerAdmin/resources/definitions
```

Sample result:

```
[{"name": "D:\\Apps\\DoSAttack.json", "url": "\\ProtonOnWebServerAdmin\\resources\\definitions\\DoSAttack"}]
```

- Creating a new definition (notice the “name” property (containing the name for the definition) added alongside the “epn” property (containing the full definition))

The definition file itself, in JSON format, can be generated by the UI, and can be generated programmatically by any other application. The format of this definition file is described by a [JSON schema for a CEP application](#) while the semantics of the various elements in this schema are described in a [user guide](#). Examples for definitions can be found in the [CEP test plan](#)

```
POST //localhost:8080/ProtonOnWebServerAdmin/resources/definitions
{"name": "MyDefinition", "epn": {...}}
```

Result:

```
/ProtonOnWebServerAdmin/resources/definitions/MyDefinition
```

Performing GET on the returned resource will retrieve the complete definition (epn) in JSON format.

10.3.4 Administrating Instances

There are two administration actions that can be performed on an instance.

1. Changing the definition (epn) for the instance to work with. This will define the types of events the instance will accept for processing and the type of patterns it will be computing.
2. Starting or stopping the instance

Verb	URI example	Description
GET	<code>//{CEP_Admin}/resources/instances/{instance_name}</code>	Retrieve the status of an instance, the definition URI it is configured with and its state (stopped or started)
PUT	<code>//{CEP_Admin}/resources/instances/{instance_name}</code>	Configuring the instance with a definition file or start\stop the instance

10.3.4.1 Usage Examples

Retrieving an instance status

```
GET
//localhost:8080/ProtonOnWebServerAdmin/resources/instances/ProtonOnWebServer
```

Sample result:

```
{"state": "started", "definitions-  
url": "\/ProtonOnWebServerAdmin\/resources\/definitions\/DoSAttack"}
```

Configuring\changing a definition for an instance

```
PUT  
//localhost:8080/ProtonOnWebServerAdmin/resources/instances/ProtonOnWe  
bServer  
  
{"action": "ChangeDefinitions", "definitions-  
url": "\/ProtonOnWebServerAdmin\/resources\/definitions\/DoSAttack"}
```

Starting an instance (replace start with stop to stop an instance)

```
PUT  
//localhost:8080/ProtonOnWebServerAdmin/resources/instances/ProtonOnWe  
bServer  
  
{"action": "ChangeState", "state": "start"}
```


11 FIWARE OpenSpecification Data SemanticAnnotation

Name	FIWARE.OpenSpecification.Data.SemanticAnnotation		
Chapter	Data/Context Management,		
Catalogue-Link Implementation	to <Semantic Annotation>	Owner	Telecom Italia, Mondin Fabio Luciano

11.1 Preface

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

11.1.1 Copyright

- Copyright © 2012 by [Telecom Italia](#)

11.1.2 Legal Notice

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

11.1.3 Overview

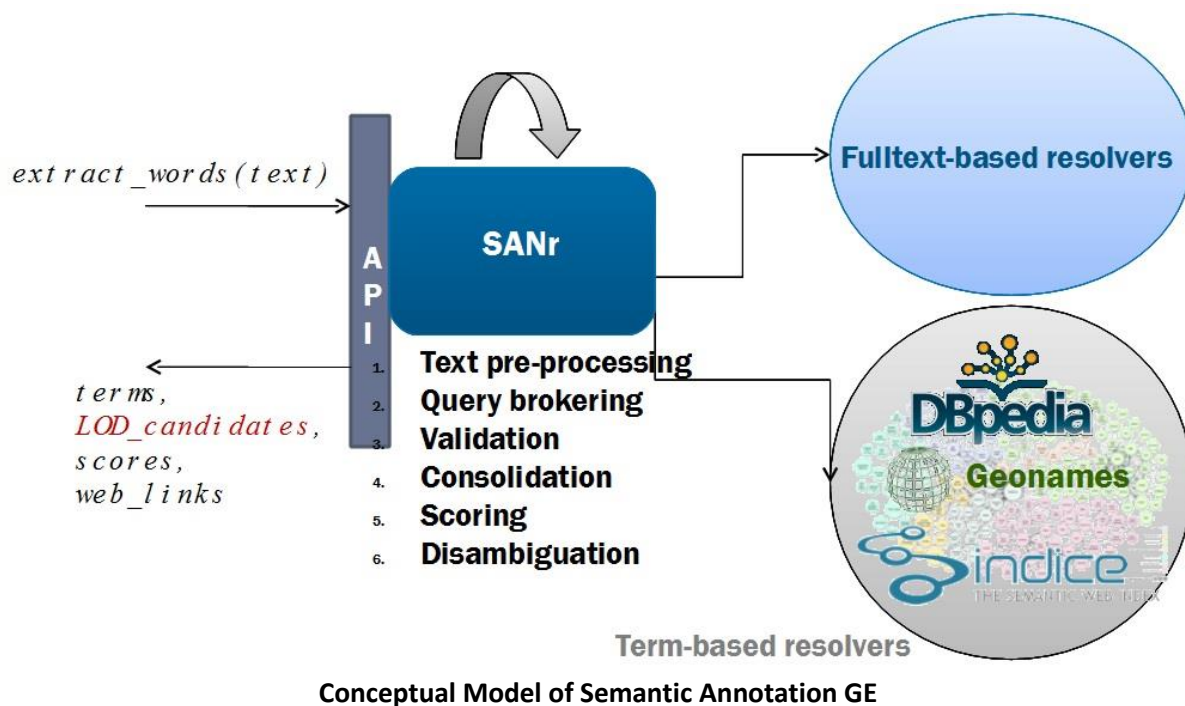
The principle standing behind Semantic Web is to evolve the "link" concept from an unspecified element describing the relationship between two elements into a "named relationship". This should clarify which is(are) the relationship(s) between those elements.

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

The Subject is a URI, identifying uniquely a particular resource to be described, while the predicate (and sometimes the object) can describe objects and their relationships. The Semantic Annotator is basically a tool which tries to identify important entities (places, persons, organizations) and associate them 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 built 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 Semantic Annotation GE is shown in the Figure below.



11.1.3.1 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)

11.1.3.2 *Basic Design Principles*

The Enabler has been designed following a modular approach, as it is shown in the figure above. This way each component in the enabler can be developed or changed, given that it provides the same input/output format.

The Semantic Annotation reasoner (SANr), communicates with a full text based resolver, in order to identify entities in text and with Semantic Data Storages to link these identities with candidates.

This leaves open the road to change data sources in order to have other data sources than Dbpedia [\[1\]](#) or Geonames [\[2\]](#) or to change the process standing behind the candidate's choice for each entity.

11.1.4 Basic Concepts

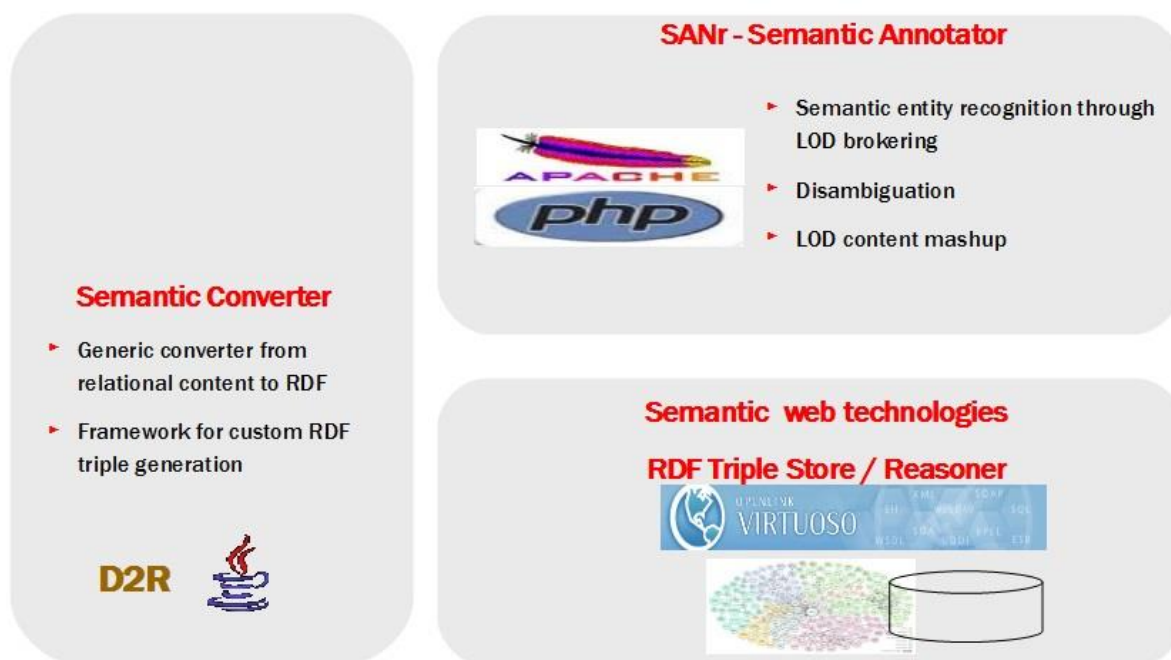
The GE has a web API, supports multilingual texts (Italian, English, Spanish, Portuguese) and includes "candidate" LOD resources and performs disambiguation. As a result the GE creates external links and HTML snippets showing in a user-friendly way LOD information.

The API processes the input text with a language processor in order to identify entities in text which are basically persons, places and organizations. This is performed by crossing grammatical and syntactic information.

Once the entities are identified, the system tries to associate a list of candidates to each entity. Candidates are entries coming from Dbpedia and Geonames which are the most used general purpose semantic databases. Candidate association is performed by comparing each entity with the Dbpedia Labels, the most similar ones are chosen as candidates.

For each candidate, the system computes a score based on a syntactic similarity metric (e.g. if the entity is “foo”, a candidate with label “foo” will have higher score than another one with label “foo bar”). This score is then mixed with another score coming from an algorithm trying to evaluate how each candidate semantically fits in the context. To understand well a candidate structure check the example in “Main Interactions” section.

External Modules (such as Semantic Data Repositories) are parametric, so one can decide to replicate semantic datasets (such as DBPedia) locally, in order to improve performance. A typical usage, with Semantic Annotation used jointly with a local semantic data storage and a Relational-to-Semantic Converter, is shown in the Figure below.



Semantic Annotation Typical Usage

11.1.5 Main Interactions

The enabler basically consists of an API, which can be called by a simple HTTP GET request to this URL, so the interaction is a simple CALL->RESPONSE.

http://semantican.lab.fi-ware.org/ajax/extract_words.php?text=

with a text to analyze as input which has to be passed as "text" parameter as shown in the link above.

This system will:

1. Identify 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 (since it is more versatile than XML) as "candidates". Each candidate has a score. The candidate with the highest score is flagged as "preferred".
5. The query is logged into a Database with an ID.

Here's an example of the return result in JSON format.

```
{
  "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": "http://dbpedia.org/resource/Mario_Monti",
            "tag--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_Presi
dente_del_Consiglio_incaricato_Mario_Monti_(cropped).jpg/200px-
Il_Presidente_del_Consiglio_incaricato_Mario_Monti_(cropped).jpg'
height=160 /><br><div class=info>^ 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'Università Bocconi dal 1994,
Monti ˆ 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>Conc
etti 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.

11.2 Detailed Specifications

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

11.2.1 Open API Specifications

- [Semantic Annotation Open RESTful API Specification](#)

11.3 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 Recognition. [\[3\]](#)
- **Dbpedia** One of the most important general data sources used by the enabler. [\[4\]](#)

- **Geonames** Reference data source for places [\[5\]](#)

11.4 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 **avalue**. 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.

12 Semantic Annotation Open RESTful API Specification

12.1.1 Introduction to Semantic Annotation API

The enabler basically consists of an API, which can be called by a simple HTTP GET request to an URL plus some parameters in order to obtain a specific JSON result, related to the input text. This API processes the input text in order to find entities (Persons, places, organizations) and associate to each entity a list of specific candidates coming from Linked Open Data Datasets (i.e. Dbpedia, Geonames, etc.)

This document specifically explains the interaction and result structure for the Semantic Annotation API. For further details about datasets and general info about the enabler check the Open Specifications.

12.1.2 API Operations

Verb	URI	Description
GET	http://semantican.lab.fi-ware.org/ajax/extract_words.php	Semantically Annotate Text

Response codes:

- JSON Response - If the request is successful (even if no entity is found in text)
- HTTP/1.1 500 - If there are some unidentified errors.

12.1.3 API Parameters

Parameter shall be passed as HTTP 1.1 - GET:

Parameter	Accepted Values	Description
text	urlencoded text	The text to annotate
html_snippets	on off	Tell the system to return or not the HTML Snippets related to entities

12.1.4 API Result

Returns Entities and Related Information in JSON format. Main parameters are:

Return Parameter	Description
queryId	Query Id, in order to cache queries and results (for future applications)

keywords	The keywords found
terms	array, each entry contains info about an entity
candidates	array of candidates for each entity, each entry contains info about a candidate

for each "candidate" entry:

Return Parameter	Description
label	human readable label for the candidate
uri	candidate URI
wrapper	data source containing the candidate
sc	relevance score for the entity
preferred	parameter set to true for the candidate with the highest score

Return Example:

```
{
  "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_Presi
dente_del_Consiglio_incaricato_Mario_Monti_(cropped).jpg/200px-
Il_Presidente_del_Consiglio_incaricato_Mario_Monti_(cropped).jpg'
height=160 /><br><div class=info>Ãˆ senatore a vita dal 9 novembre
2011 e dal successivo 16 novembre assume, per la prima volta,
l'incarico di Presidente del Consiglio dei Ministri della Repubblica
Italiana e allo stesso tempo di Ministro dell'Economia e delle Finanze
dello stesso governo. Presidente dell'UniversitÃ Bocconi dal 1994,
Monti Ã¨ stato c...<ul><li><a
href='http://www.guardian.co.uk/world/mario-monti'
target='_blank'>Link
utile</a></li></ul></div></div></div></fieldset><fieldset><legend>Conc

```

```

etti 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"
  }
]
}

```

13 FIWARE OpenSpecification Data SemanticSupport

Name	FIWARE.OpenSpecification.Data.SemanticSupport		
Chapter	Data/Context Management,		
Catalogue-Link Implementation	to <Semantic Application Support>	Owner	Atos Origin, Jose Maria Fuentes Lopez

13.1 Preface

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

13.1.1 Copyright

- Copyright © 2012 by [Atos Origin](#)

13.1.2 Legal Notice

Please check the following [Legal Notice](#) to understand the rights to use these specifications.

13.1.3 Overview

13.1.3.1 *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, 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 [\[SAS\]](#).

13.1.3.2 *Semantic Application Support GE Description*

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 [Bizer 2009]. 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:

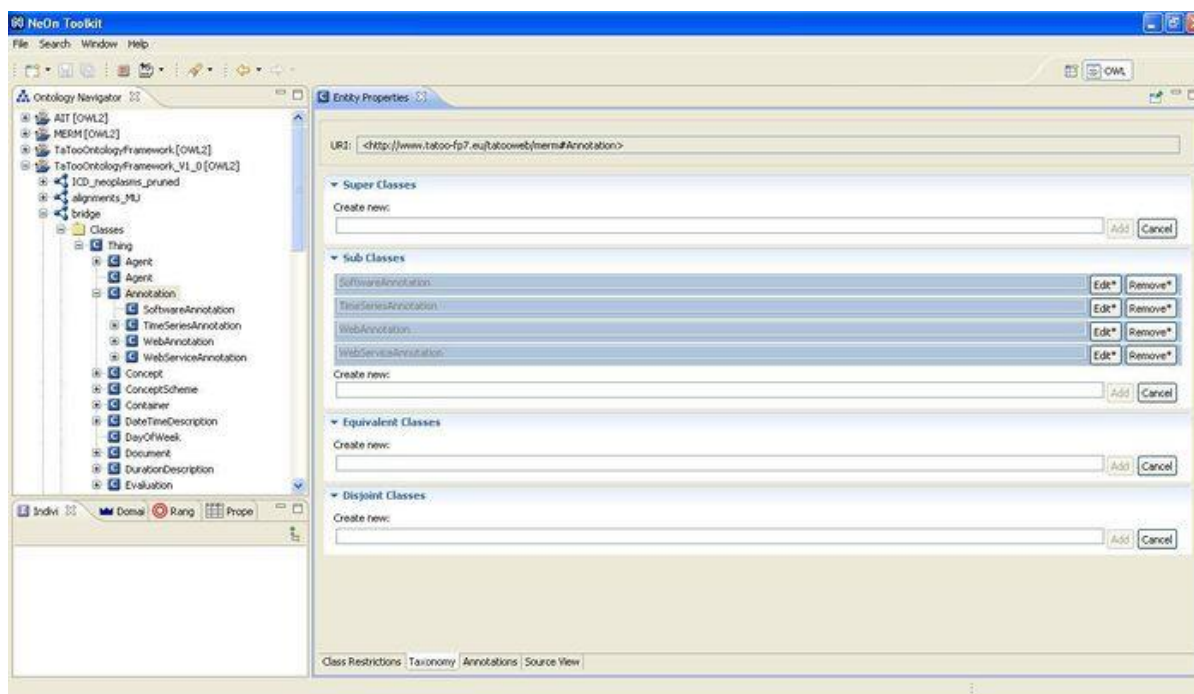
- An infrastructure for metadata publishing, retrieving and subscribing that meets industry requirements like scalability, distribution and security. From now and so on, we will refer to this infrastructure as SAS Infrastructure.
- A set of tools for infrastructure and data management, supporting most adopted methodologies and best practices. From now and so on, we will refer to these tools as SAS Engineering Environment.

13.1.3.3 *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 for an infrastructure that eases the development, storage and use of ontologies and allows 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 future Internet application developers. TaToo is a research project in

the environmental domain with the goal of developing tools to facilitate the discovery of environmental resources. In order to enhance the discovery process, one of the applications stores annotations (metadata) of existing environmental resources by tagging them with ontology terms. Therefore, an ontology framework [Pariante 2011] has been developed including three domain ontologies that describe three different environmental domains plus a bridge ontology that allows cross domain interoperability. Moreover, TaToo ontologies are not built from scratch but by reusing (completely or partially) existing ontologies publicly available on 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. By hence, the use of a methodology, as well as a set of tools to assist in the process of ontology engineering will be required. 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 for the ontology engineering process. The NeOn Toolkit helped TaToo's ontology developers to apply the NeOn methodology to develop ontologies providing several functionality such as ontology edition, ontology modularization, ontology search, etc. Besides, these ontologies are expected to evolve over time, and would therefore 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. These functionalities have been developed in the scope of FI-WARE project. Next figure shows a screenshot of NeOn Toolkit being used in the scope of TaToo.



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 RDF / OWL oriented storages, so there is a lack of knowledge base management capabilities. As an example, once an ontology is loaded into a Sesame workspace, it is not possible to keep tracking management over it. In case the ontology evolves over time, there is no possibility to track the workspace in order to look for the incremental updates over the ontology. This kind of knowledge base management problems are solved by the Semantic Support Application.

To summarize, 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. This environment will support strong ontology development methodology, covering the whole semantic web application lifecycle. This is clearly extensible to many different Semantic Web-based applications.

13.1.4 Basic Concepts

This section introduces the basic concepts related to the Semantic Support Application GE including ontologies, ontology languages and ontology development methodologies.

13.1.4.1 *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 it is 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 fulfil 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 [\[Suarez-Figueroa 2008\]](#) as reference methodology for ontology development.

Both decisions will be discussed in the following sections.

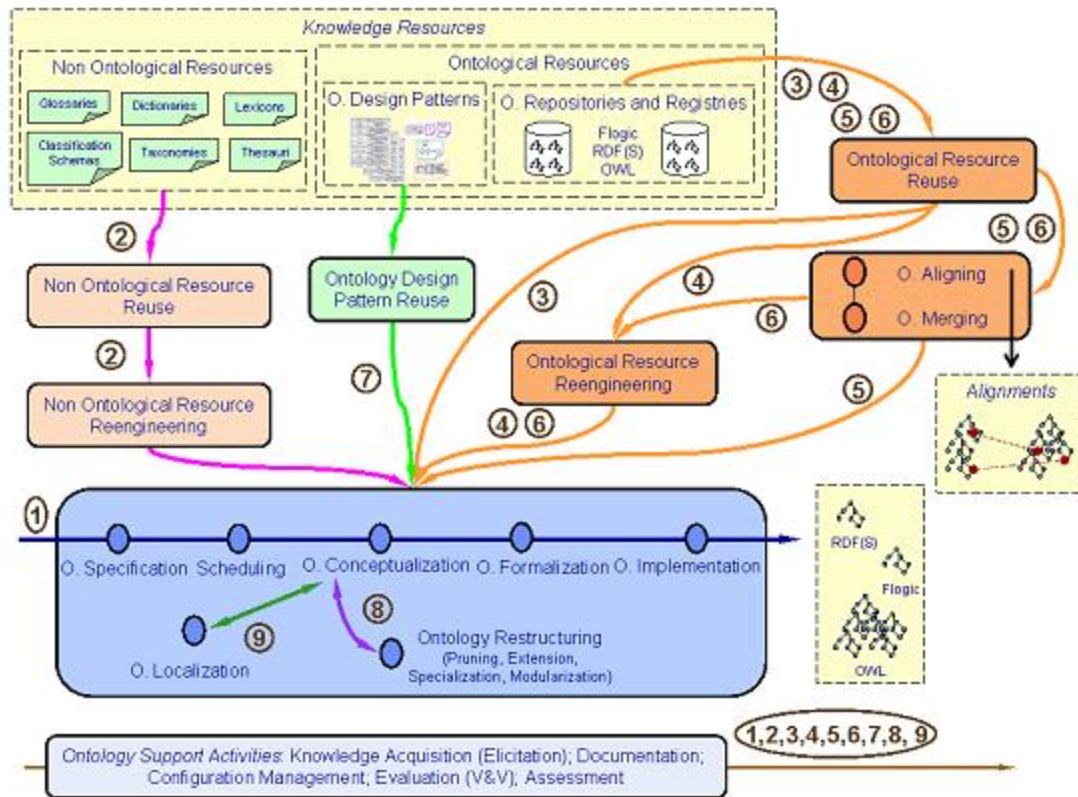
13.1.4.2 **OWL-2**

Since the inception of the ontologies, several ontology languages, with different expressivity, serialization and underlying logic formalisms have risen and fallen (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, providing 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 reasonable 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.

13.1.4.3 **Ontology Engineering**

SAS 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 provides 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.



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.

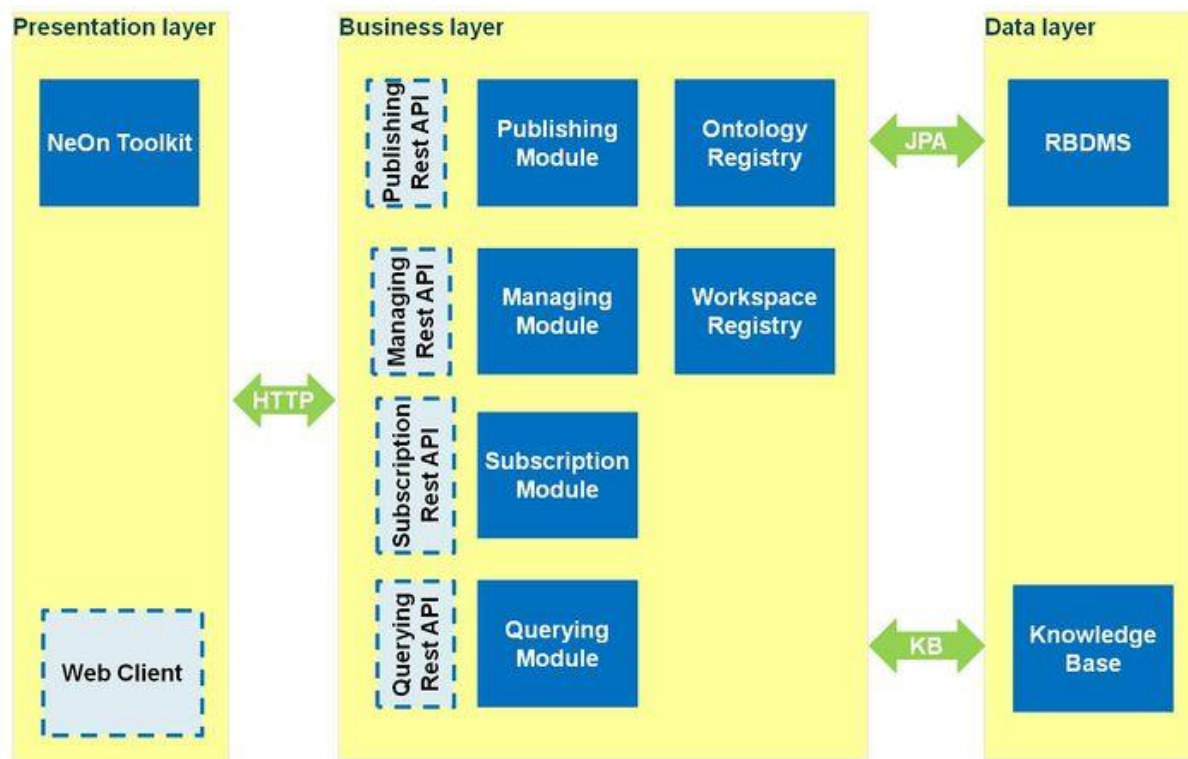
The first scenario represents the typical case where the ontology is built from scratch; the other scenarios involve different levels of reuse of other resources, those scenarios are visually described in

previous figure. For each of all these scenarios, the NeOn methodology establishes detailed guidelines, tools to use, etc. The Semantic Web Application Support GE provides a technological support for those guidelines.

13.1.5 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. Next figure presents the SAS Infrastructure architecture.



SAS 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, which 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. On the one hand to interact with the relational database, business components will use JPA (Java Persistence API) that make business components database system independent. On the other hand, business components interacting with the knowledge base will be knowledge base implementation dependent. In Semantic Application Support reference implementation, the combination of Sesame and OWLIM has been chosen as knowledge base implementation.

Business Layer contains following components:

- Ontology registry that manages ontologies loaded into the system and its related metadata. Operations such as retrieving / uploading ontology, retrieving / uploading metadata, etc would be provided by this component. A description of methods provided 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 as creating / deleting a workspace, listing ontologies loaded into the workspace is provided by this component.
- 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.
- 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.
- 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.
- Querying module that allows users to query their workspace following SPARQL Query Protocol.

In order to provide GE functionality in a platform independent way, several Rest APIs was developed. This allows to clients or presentation layer applications to interact with business components by means of HTTP requests / responses.

13.1.6 Main Interactions

13.1.6.1 *Modules and Interfaces*

This section reports on 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.

13.1.6.2 *Backend Functionality*

Backend functionality describes functionality provided by the GE as service invocation methods for both human or computer agents. As described in Architecture section [\[SAS Architecture\]](#), this functionality is accessible by means of Rest Web Services API. In this second FI-WARE release, a sub set of methods belonging to publishing, managing, semantic workspaces management and semantic workspaces operations 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 a file attachment 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 attachment 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 a file attachment 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`. The output is an xml encoding the requested information that 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>`. The output is an xml encoding the requested information that will be sent as response.

- **Workspaces Management Rest API.**

- **List Workspaces:** Retrieves an XML document which contains a list of all workspaces managed by the server, a GET http request should be sent to `http://<ge url location>/semantic-workspaces-service/rest/workspaces/mgm/list`. The output is an xml encoding the list of workspaces.
- **Workspace Operations Rest API.**
- **Create Workspace:** Creates a new semantic workspace, a POST http request should be sent to `http://<ge url location>/semantic-workspaces-service/rest/workspaces/[WORKSPACE_NAME]`. The output is an xml document encoding the result of the operation.
- **Remove Workspace:** Remove an existing semantic workspace, a DELETE http request should be sent to `http://<ge url location>/semantic-workspaces-service/rest/workspaces/[WORKSPACE_NAME]`. The output is an xml encoding the result of the operation.
- **Duplicate Workspace:** Creates a duplicate of a existing workspace with his metadata (ontologies and triples), a PUT http request should be sent to `http://<ge url location>/semantic-workspaces-service/rest/workspaces/[WORKSPACE_NAME]/duplicate`. The output is an xml document encoding the result of the operation.
- **Execute Query:** Execute a SPARQL query into a existing workspace, a POST http request should be sent to `http://<ge url location>/semantic-workspaces-service/rest/workspaces/[WORKSPACE_NAME]/sparql/`. The output is an xml document encoding the result of the query.
- **Get Workspace:** Retrieves the RDF from a specific workspace, a GET http request should be sent to `http://<ge url location>/semantic-workspaces-service/rest/workspaces/[WORKSPACE_NAME]`. The output is a RDF/XML encoding the data.
- **Get ontologies updates:** Retrieves a list of available updates for the ontologies included in a workspace, a GET http request should be sent to `http://<ge url location>/semantic-workspaces-service/rest/workspaces/[WORKSPACE_NAME]/checkupdates`. The output is a XML encoding the list of updates.
- **Load Ontology:** Load an ontology into a workspace from a specific ontology registry, a POST http request should be sent to `http://<ge url location>/semantic-workspaces-service/rest/workspaces/[WORKSPACE_NAME]/ontology/[ONTOLOGY_NAME]`. The output is a XML encoding the operation result.
- **List Ontologies:** Retrieves a list with the ontologies included in a workspace, a GET http request should be sent to `http://<ge url location>/semantic-workspaces-service/rest/workspaces/[WORKSPACE_NAME]/ontology/list`. The output is a XML encoding the list of ontologies.

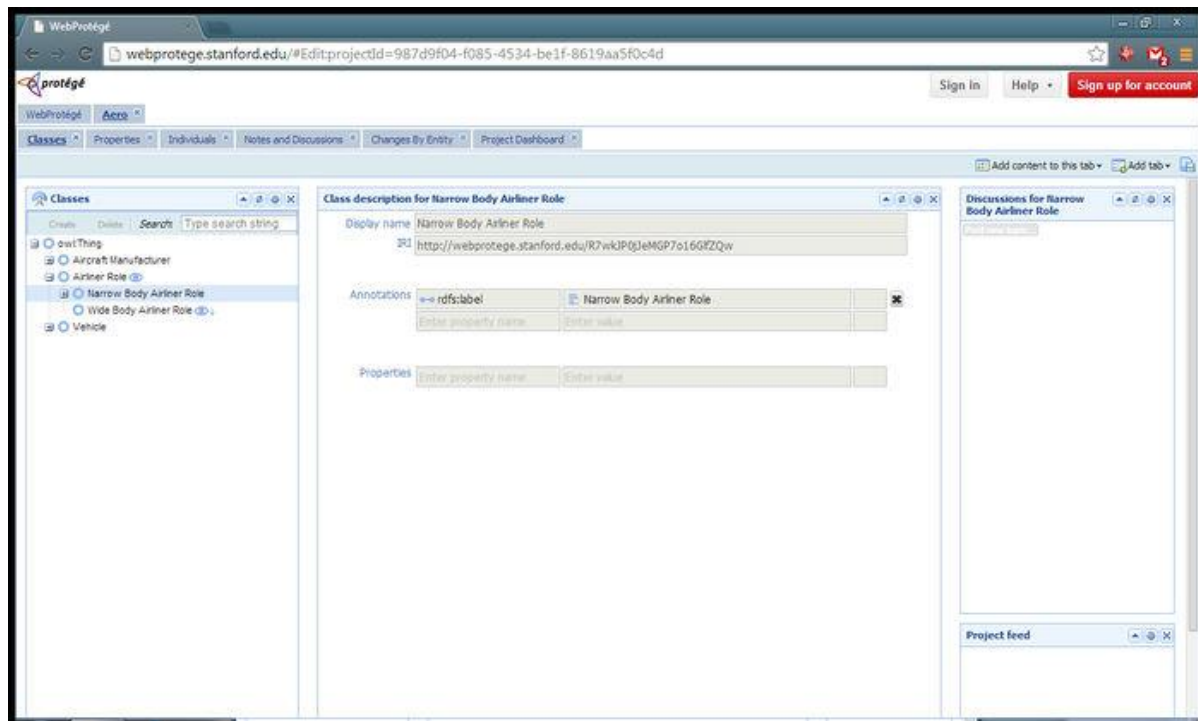
- **Update ontology:** Update an ontology included in a workspace using a specific ontology registry, a GET http request should be sent to `http://<ge url location>/semantic-workspaces-service/rest/workspaces/[WORKSPACE_NAME]/ontology/[ONTOLOGY_NAME]/update`. The output is a XML encoding the result of the operation.
- **Delete Ontology:** Delete an ontology from a workspace , a DELETE http request should be sent to `http://<ge url location>/semantic-workspaces-service/rest/workspaces/[WORKSPACE_NAME]/ontology/[ONTOLOGY_NAME]`. The output is an XML document encoding the operation result.
- **Create Context with RDF:** Create a context with RDF data into an existing workspace , a POST http request should be sent to `http://<ge url location>/semantic-workspaces-service/rest/workspaces/[WORKSPACE_NAME]/context/[CONTEXT_NAME]`. The context will be cleared and then the RDF will be loaded. The output is an xml document encoding the result of the operation.
- **Load RDF into Context:** Load RDF data into a context of an existing workspace , a PUT http request should be sent to `http://<ge url location>/semantic-workspaces-service/rest/workspaces/[WORKSPACE_NAME]/context/[CONTEXT_NAME]`. The context will be cleared and then the RDF will be loaded. The output is an xml document encoding the result of the operation.
- **Delete Context:** Removes a context of a specific workspace, a DELETE http request should be sent to `http://<ge url location>/semantic-workspaces-service/rest/workspaces/[WORKSPACE_NAME]/context/[CONTEXT_NAME]`. The output is a XML encoding the result of the operation.
- **List Contexts:** List the contexts included in a specific workspace, a GET http request should be sent to `http://<ge url location>/semantic-workspaces-service/rest/workspaces/[WORKSPACE_NAME]/context/list`. The output is a XML encoding the list of context.
- **Add Statement:** Add a statement (RDF triple) into a specific workspace, a POST http request should be sent to `http://<ge url location>/semantic-workspaces-service/rest/workspaces/[WORKSPACE_NAME]/context/[CONTEXT_NAME]/statement`. The output is a XML encoding the result of the operation.
- **Remove Statement:** Remove a statement (RDF triple) from a specific workspace, a DELETE http request should be sent to `http://<ge url location>/semantic-workspaces-service/rest/workspaces/[WORKSPACE_NAME]/context/[CONTEXT_NAME]/statement`. The output is a XML encoding the result of the operation.

All methods described can be invoked by means of regular HTTP requests either using a web browser (for those ones who rely on GET requests) or by a APIs such as Jersey.

13.1.6.3 Frontend Functionality

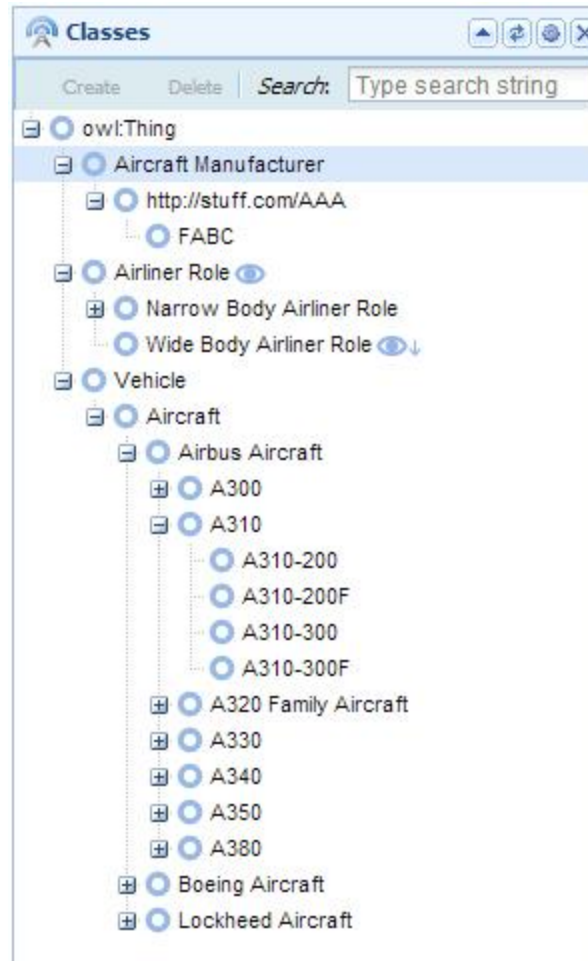
SAS Web User interface is based on the functionality provided by the Web Protégé. The Web Protégé is the web version of Protégé [\[WebProtege\]](#), a well-know ontology editor, which provides comprehensive

support for the ontology engineering life-cycle. Due to its nature, it wouldn't be possible to describe all SAS Engineering Environment provided functionality in a service like manner. Anyway, an overview of the functionality required for the SAS Engineering Environment is now introduced. Some screenshots from Web Protégé will be used in this section to provide a better understanding of the SAS functionalities. Figure below presents an overview of Web GUI.



Web Protégé main window

Most of the functionality provided by SAS Web UI is provided as editors, views and perspectives. Next figure presents the Ontology navigation perspective.



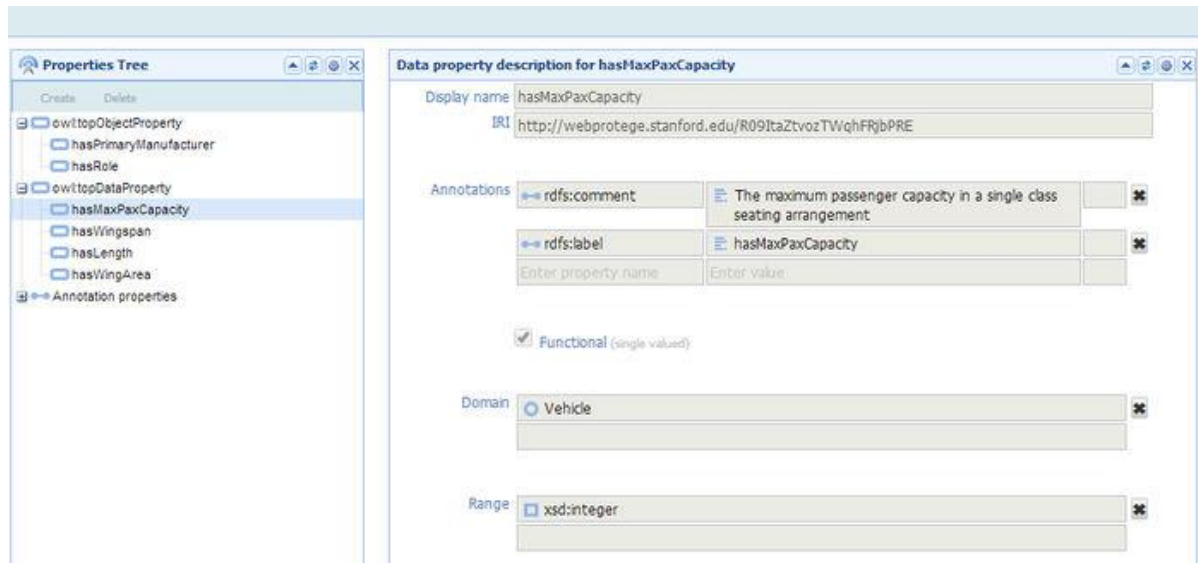
Ontology navigation perspective

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, properties and individuals. Once selected, ontology contents can be edited by means of a proper editor. Next figure presents the class editor.



Class editor

Properties provide functionalities to represent the interactions between classes and individuals. The Properties view allows managing the ontology's properties and its related metadata. Next figure presents the properties view.



Properties View

The SAS Web UI interacts with the Ontology Registry and Semantic Workspaces allowing to retrieve and modify the ontological resources contained in these components. As the SAS Web UI is based on the widely used Protégé editor will be easy to any developer to use this Web UI, this fact facilitate the diffusion of the Semantic Application Support GE in the Semantic developers community.

13.1.7 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.

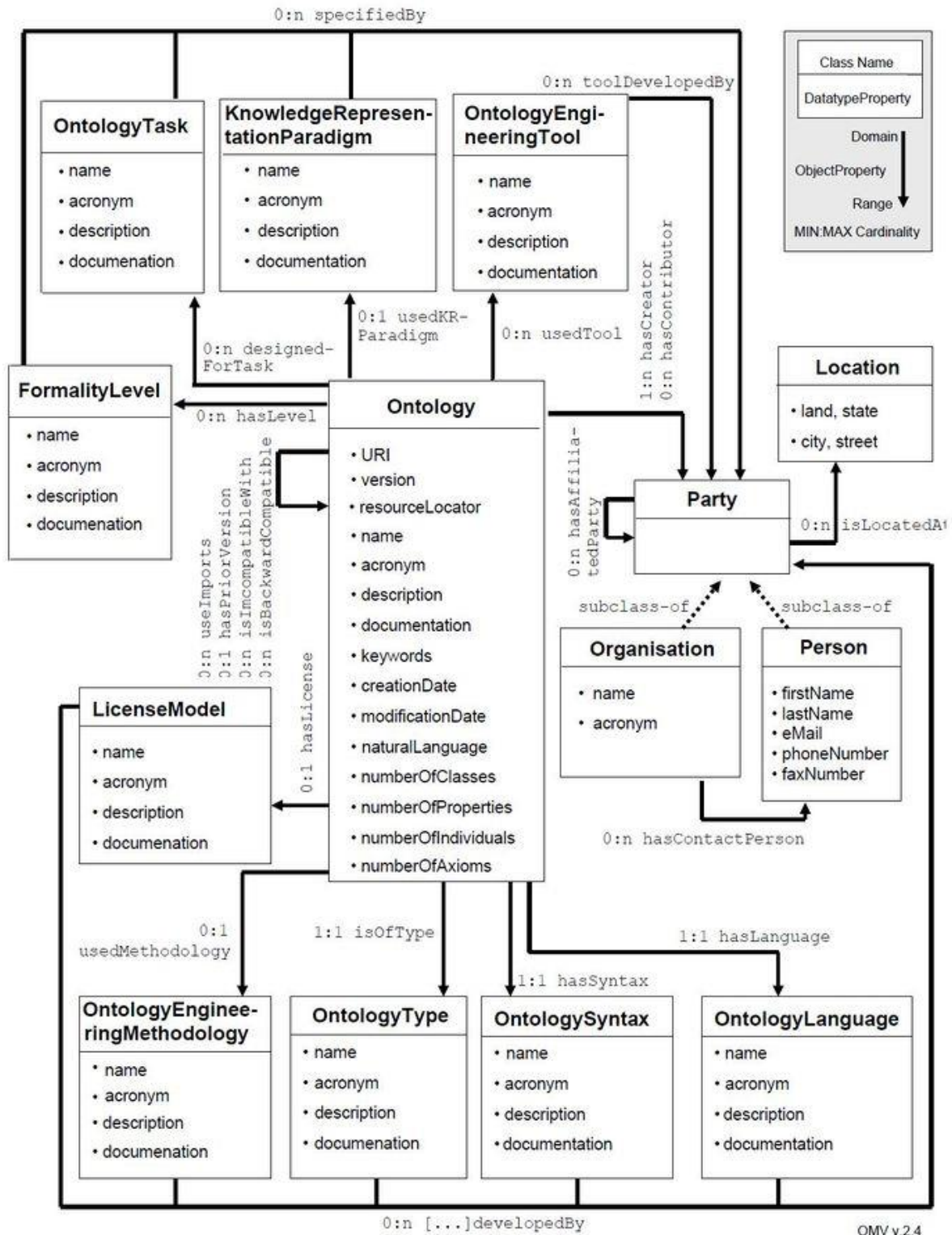
- **Ontology management:** The enabler will provide an ontology registry and the API to control it, including some high-level ontology management functionalities.
- **Knowledge Base management:** The enabler will provide a knowledge base registry and the API to control it, including some high-level knowledge-base 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.

For the Ontology Registry component, some decisions regarding the design have 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 store ontologies.

In order to provide advanced ontology management, 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** An ontology developed in the NeOn project following the NeOn Methodology.
- 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 the figure below.



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 seem to be an easy task but it is not even completely tackled, even in OWL-2 specification. Taking a look into the 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 their ontology IRI plus version IRI, being this last one optional. In case no version IRI is provided, the 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 database. This would avoid potential performance problems while serving ontologies for developers for editing purposes.

Finally, an interface for accessing the ontology registry will be provided. In this case, Semantic Application Support GE will follow [\[SPARQL Query protocol\]](#): If a service supports HTTP bindings, it must support the bindings as described in the 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.

13.1.8 References

[Pariante 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.
-----------------	---

[Bizer 2009]	Bizer, C., Heath, T., & Berners-Lee, T. (2011). Linked Data - The Story So Far. International Journal on Semantic Web and Information Systems, 5(3), 1-22.
[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 Mántaras 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
[SPARQL Query]	http://www.w3.org/TR/rdf-sparql-protocol/

protocol]	
[WebProtege]	http://protegewiki.stanford.edu/wiki/WebProtege

13.1.9 Detailed Specifications

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

13.1.9.1 *Open API Specifications*

- [Semantic Support Open RESTful API Specification](#)

13.1.9.2 *Other Open Specifications*

- [FIWARE.ArchitectureDescription.Data.SemanticSupport.O MV Open Specification](#)

13.2 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

13.3 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 **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 be 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 two 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.

14 Semantic Support Open RESTful API Specification

14.1 Introduction to the Ontology Registry API

Please check the [FI-WARE Open Specification Legal Notice \(implicit patents license\)](#) to understand the rights to use FI-WARE Open Specifications.

14.1.1 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 ontologies and their related metadata, enhancing ontology development lifecycle. This API is part of the set of APIs provided by the Semantic Application Support GE.

14.1.2 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 [Semantic Application Support GE](#).

14.1.3 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

14.1.4 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](#).

14.1.5 Additional Resources

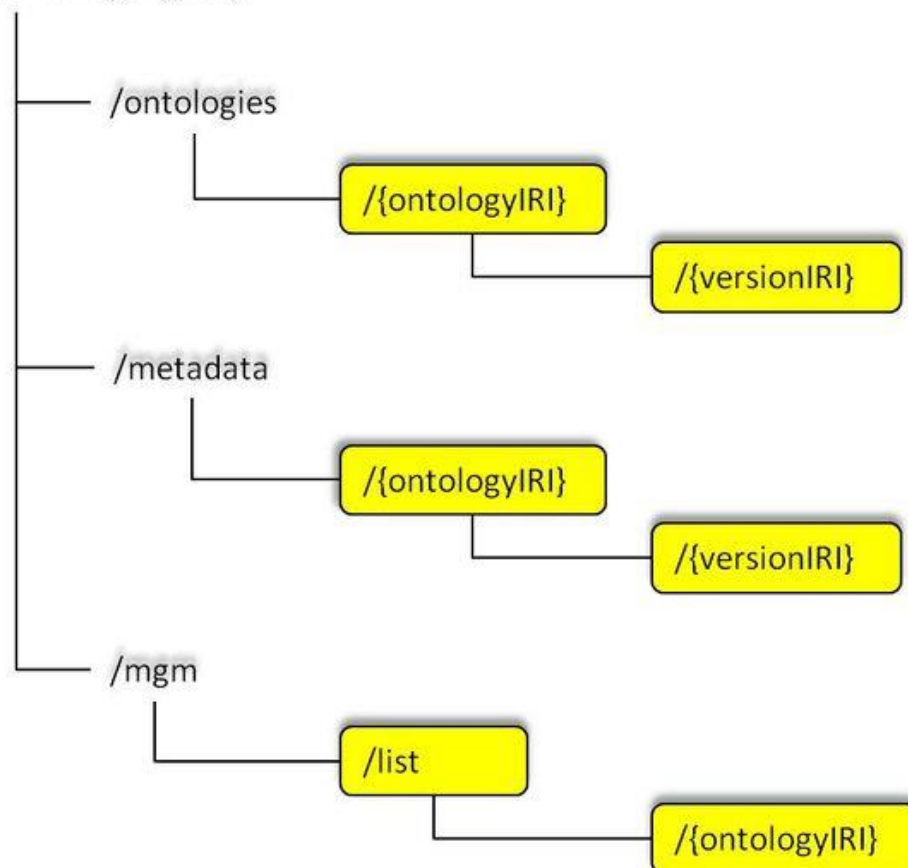
For more details about the Semantic Web Application Support GE that this API is based upon, please refer to [Semantic Web Application Support documentation](#). Related documents, including an Architectural Description, are available at the same site.

14.2 General Ontology Registry API Information

14.2.1 Resources Summary

Ontology Registry Component

//{serverRoot}/ontologyregistry



14.2.2 Representation Format

The Ontology Registry API supports XML based representation formats.

14.2.3 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.

14.2.4 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.

14.2.5 Links and References

No additional links or references are provided in this version.

14.3 API Operations

The following section provides the detail for each RESTful operation giving the expected input and output for each URI.

14.3.1 Ontology Operations

14.3.1.1 *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>

    ...

```

14.3.1.2 *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 successfully 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>

    ...

```

14.3.1.3 *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
```

14.3.1.4 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 successfully 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
```

14.3.1.5 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 successfully 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
```

14.3.1.6 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 successfully 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
```

14.3.2 Management Operations

14.3.2.1 *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>

  ...
```

14.3.2.2 *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 successfully 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>
```

14.3.3 Metadata Operations

14.3.3.1 *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 successfully 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#individual13331
10758440"/>

    <hasOntologyLanguage
rdf:resource="http://omv.ontoware.org/2005/05/ontology#OWL"/>
</rdf:Description>

</rdf:RDF>

```

14.3.3.2 *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#individual13331
10758440"/>

    <hasOntologyLanguage
rdf:resource="http://omv.ontoware.org/2005/05/ontology#OWL"/>

</rdf:Description>

</rdf:RDF>

```

14.3.3.3 *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
```

14.3.3.4 *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 successfully 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
```

14.3.3.5 *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 successfully 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#individual13331
10758440"/>

    <hasOntologyLanguage
rdf:resource="http://omv.ontoware.org/2005/05/ontology#OWL"/>

</rdf:Description>

</rdf:RDF>
```

Response example:

```
HTTP/1.1 200 OK
```

14.3.3.6 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-
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#individual13331
10758440"/>

    <hasOntologyLanguage
rdf:resource="http://omv.ontoware.org/2005/05/ontology#OWL"/>

</rdf:Description>

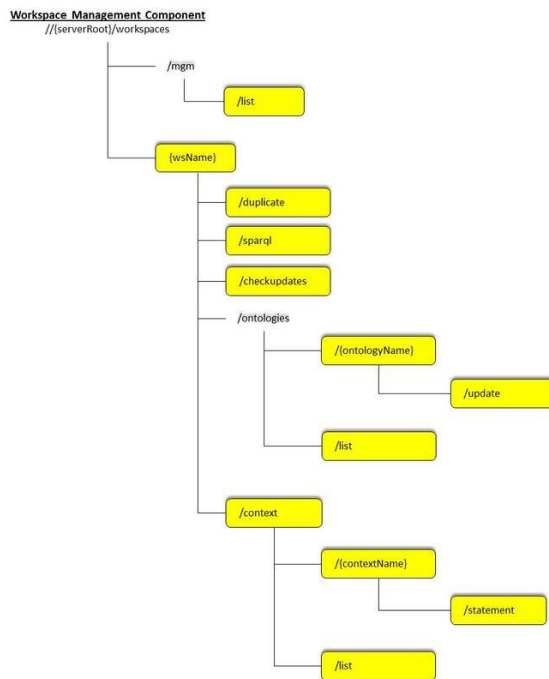
</rdf:RDF>
```

Response example:

```
HTTP/1.1 200 OK
```

14.4 General Workspace Management API Information

14.4.1 Resources Summary



14.4.2 Representation Format

The Workspace Management API supports XML based representation formats.

14.4.3 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.

14.4.4 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.

14.4.5 Links and References

No additional links or references are provided in this version.

14.4.6 Limits

Limits section and operations will be provided in further FI-WARE releases.

14.4.7 Versions

Versions section and operations will be provided in further FI-WARE releases.

14.4.8 Extensions

Extensions section and operations will be provided (if needed) in further FI-WARE releases.

14.4.9 Faults

Faults section and operations will be provided (if needed) in further FI-WARE releases.

14.5 API Operations

The following section provides the detail for each RESTful operation giving the expected input and output for each URI.

14.5.1 Workspace Operations

14.5.1.1 *ListWorkspaces*

Verb	URI	Description
GET	/semantic-workspaces-service/rest/workspaces/mgm/list	List all available workspaces.

Response codes:

- HTTP/1.1 200 - If the list of workspace is succesfully provided

- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
POST /semantic-workspaces-service/rest/workspaces/mgm/list HTTP/1.1
Host: localhost:8080
Accept: text/xml;q=0.9,*/*;q=0.8
```

Response example:

```
HTTP/1.1 200 OK
Content-Type: text/xml
<?xml version="1.0" encoding="UTF-8"?>
<response><workspaces><workspace><name>test</name><description>This is
a
test</description><type>eu.atosresearch.jsrc.sesame2610.Sesame2610Driv
er</type><endpoint>http://semanticas.lab.fi-ware.eu:8080/openrdf-
sesame</endpoint></workspace><workspace><name>Test1</name><description
>This is a
test</description><type>eu.atosresearch.jsrc.sesame2610.Sesame2610Driv
er</type><endpoint>http://semanticas.lab.fi-ware.eu:8080/openrdf-
sesame</endpoint></workspace><workspace><name>TestWorkspace</name><des
cription>Workspace created to unit test the
server</description><type>eu.atosresearch.jsrc.sesame2610.Sesame2610Dr
iver</type><endpoint>http://semanticas.lab.fi-ware.eu:8080/openrdf-
sesame</endpoint></workspace><workspace><name>TestWorkspaceDuplicate</
name><description>Duplicate of the test
workspace</description><type>eu.atosresearch.jsrc.sesame2610.Sesame261
0Driver</type><endpoint>http://semanticas.lab.fi-ware.eu:8080/openrdf-
sesame</endpoint></workspace></workspaces></response>
```

14.5.1.2 Create Workspace

Verb	URI	Description
POST	/semantic-workspaces-service/rest/workspaces/{WORKSPACE_NAME}	Creates a new semantic workspace

Response codes:

- HTTP/1.1 200 - If the workspace is successfully created
- HTTP/1.1 500 - If there are some unidentified errors.

Request example:

```
POST /semantic-workspaces-service/rest/workspaces/test HTTP/1.1
```

```
Host: localhost:8080
```

```
Accept: application/xml;q=0.9,*/*;q=0.8
```

Response example:

```
HTTP/1.1 200 OK
```

```
Content-Type: application/xml
```

14.5.1.3 Remove Workspace

Verb	URI	Description
DELETE	/semantic-workspaces-service/rest/workspaces/{WORKSPACE_NAME}	Remove an existing semantic workspace

Response codes:

- HTTP/1.1 200 - If the workspace was successfully deleted
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
DELETE /semantic-workspaces-service/rest/workspaces/test HTTP/1.1
```

```
Host: localhost:8080
```

```
Accept: text/xml;q=0.9,*/*;q=0.8
```

Response example:

```
HTTP/1.1 200 OK
```

```
Content-Type: text/xml
```

14.5.1.4 Duplicate Workspace

Verb	URI	Description
PUT	/semantic-workspaces-service/rest/workspaces/{WORKSPACE_NAME}/duplicate	Creates a duplicate of a existing workspace with his metadata (ontologies and triples)

Response codes:

- HTTP/1.1 200 - If the workspace was successfully duplicated.
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
PUT /semantic-workspaces-service/rest/workspaces/test/duplicate
HTTP/1.1

Host: localhost:8080

Accept: text/xml;q=0.9, */*;q=0.8
```

Response example:

```
HTTP/1.1 200 OK

Content-Type: text/xml
```

14.5.1.5 Execute Query

Verb	URI	Description
POST	/semantic-workspaces-service/rest/workspaces/{WORKSPACE_NAME}/sparql/	Execute a SPARQL query into a existing workspace

Response codes:

- HTTP/1.1 200 - If the query was successfully executed
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
POST /semantic-workspaces-service/rest/workspaces/test/sparql HTTP/1.1

Host: localhost:8080

Accept: text/xml;q=0.9, */*;q=0.8

FormParam:
query=SELECT%20DISTINCT%20*%20WHERE%20%7B%20%20%20%3Fs%20%3Fp%20%3Fo%20%7D%20%20LIMIT%201
```

Response example:

```
HTTP/1.1 200 OK

Content-Type: application/xml

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

<sparql xmlns='http://www.w3.org/2005/sparql-results#'>

  <head>

    <variable name='s' />
```

```

        <variable name='p' />

        <variable name='o' />

    </head>

    <results>

        <result>

            <binding name='s'>

                <uri>http://www.w3.org/1999/02/22-rdf-syntax-
ns#type</uri>

            </binding>

            <binding name='p'>

                <uri>http://www.w3.org/1999/02/22-rdf-syntax-
ns#type</uri>

            </binding>

            <binding name='o'>

                <uri>http://www.w3.org/1999/02/22-rdf-syntax-
ns#Property</uri>

            </binding>

        </result>

    </results>

</sparql>

```

14.5.1.6 *Get Workspace*

Verb	URI	Description
GET	/semantic-workspaces-service/rest/workspaces/{WORKSPACE_NAME}	Retrieves the RDF from a specific workspace

Response codes:

- HTTP/1.1 200 - If the workspace was successfully retrieved.
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
GET /semantic-workspaces-service/rest/workspaces/test HTTP/1.1
Host: localhost:8080
Accept: text/xml;q=0.9, */*;q=0.8
```

Response example:

```
HTTP/1.1 200 OK
Content-Type: text/xml
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:gn="http://www.geonames.org/ontology#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:wgs84_pos="http://www.w3.org/2003/01/geo/wgs84_pos#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:www="http://www.geonames.org/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ontology="http://www.geonames.org/ontology/"
  xmlns:skos="http://www.w3.org/2004/02/skos/core#"
  xmlns:dcterms="http://purl.org/dc/terms/">
</rdf:RDF>
```

14.5.1.7 Get ontologies updates

Verb	URI	Description
GET	/semantic-workspaces-service/rest/workspaces/{ WORKSPACE_NAME}/checkupdates	Retrieves a list of available updates for the ontologies included in a workspace

Response codes:

- HTTP/1.1 200 - If the list of available updates was successfully retrieved.

- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
GET /semantic-workspaces-service/rest/workspaces/test/checkupdates
HTTP/1.1

Host: localhost:8080

Accept: text/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"?>

<response><outofdate></outofdate></response>
```

14.5.1.8 Load Ontology

Verb	URI	Description
POST	/semantic-workspaces-service/rest/workspaces/{WORKSPACE_NAME}/ontology/{ONTOLOGY_NAME}	Load an ontology into a workspace from a specific ontology registry

Response codes:

- HTTP/1.1 200 - If the ontology was successfully loaded into workspace.
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
POST /semantic-workspaces-
service/rest/workspaces/test/ontology/foaf.owl HTTP/1.1

Host: localhost:8080

Form-Param: version=303

Accept: 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"?>
```

```
<response><loaded>true</loaded></response>
```

14.5.1.9 *List Ontologies*

Verb	URI	Description
GET	/semantic-workspaces-service/rest/workspaces/{WORKSPACE_NAME}/ontology/list	Retrieves a list with the ontologies included in a workspace

Response codes:

- HTTP/1.1 200 - If the ontologies list included in a workspace was successfully retrieved.
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
GET      /semantic-workspaces-service/rest/workspaces/test/ontology/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"?>

<response>

<ontologies>

<ontology><name>foaf.owl</name><version>303</version>

<context>http://semanticas.lab.fi-ware.eu/semantic-
workspaces/foaf.owl</context>

</ontology>

</ontologies>

</response>
```

14.5.1.10 *Update ontology*

Verb	URI	Description
PUT	/semantic-workspaces-service/rest/workspaces/[WORKSPACE_NAME]/ontology/{ONTOLOGY_NAME}/update	Update an ontology included in a workspace using a specific ontology registry

Response codes:

- HTTP/1.1 200 - If the ontology was successfully updated in the workspace.
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
PUT /semantic-workspaces-
service/rest/workspaces/test/ontology/foaf.owl/update HTTP/1.1

Host: localhost:8080

Form-Param: version=404

Accept: 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"?>

<response><updated>true</updated></response>
```

14.5.1.11 *Delete Ontology*

Verb	URI	Description
DELETE	/semantic-workspaces-service/rest/workspaces/{WORKSPACE_NAME}/ontology/{ONTOLOGY_NAME}	Delete an ontology from a workspace

Response codes:

- HTTP/1.1 200 - If the ontology was successfully deleted from workspace.
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
DELETE /semantic-workspaces-
service/rest/workspaces/test/ontology/foaf.owl HTTP/1.1
```



```
Host: localhost:8080
Accept: 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"?>
<response><cleared>true</cleared></response>
```

14.5.1.12 Create Context with RDF

Verb	URI	Description
POST	/semantic-workspaces-service/rest/workspaces/{WORKSPACE_NAME}/context/{CONTEXT_NAME}	Create a context with RDF data into an existing workspace

Response codes:

- HTTP/1.1 200 - If the context with RDF data was successfully created.
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
POST /semantic-workspaces-
service/rest/workspaces/test/context/testContext HTTP/1.1
Host: localhost:8080
Form-Param: rdf=<rdf:RDF
```

```
xmlns:gn="http://www.geonames.org/ontology#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-
schema#" xmlns:wgs84_pos="http://www.w3.org/2003/01/geo/wgs84_pos#"
xmlns:foaf="http://xmlns.com/foaf/0.1/" xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:www="http://www.geonames.org/"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:ontology="http://www.geonames.org/ontology/"
xmlns:skos="http://www.w3.org/2004/02/skos/core#" xmlns:dcterms="http://purl.org/dc/terms/">
<foaf:Person rdf:about="#danbri" xmlns:foaf="http://xmlns.com/foaf/0.1/"> <foaf:name>Dan
Brickley</foaf:name><foaf:homepage rdf:resource="http://danbri.org/" /> <foaf:openid
rdf:resource="http://danbri.org/" /> <foaf:img rdf:resource="/images/me.jpg" /></foaf:Person>
</rdf:RDF>
```

```
Accept: 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"?>

<response><loaded>true</loaded></response>
```

14.5.1.13 Load RDF into Context

Verb	URI	Description
PUT	/semantic-workspaces-service/rest/workspaces/{WORKSPACE_NAME}/context/{CONTEXT_NAME}	Load RDF data into a context of an existing workspace

Response codes:

- HTTP/1.1 200 - If the RDF data was successfully loaded into the context.
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
PUT /semantic-workspaces-
service/rest/workspaces/test/context/testContext HTTP/1.1

Host: localhost:8080

Form-Param: rdf=<rdf:RDF
xmlns:gn="http://www.geonames.org/ontology#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:wgs84_pos="http://www.w3.org/2003/01/geo/wgs84_pos#"
xmlns:foaf="http://xmlns.com/foaf/0.1/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:www="http://www.geonames.org/"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:ontology="http://www.geonames.org/ontology/"
xmlns:skos="http://www.w3.org/2004/02/skos/core#"
xmlns:dcterms="http://purl.org/dc/terms/">
<foaf:Group>
<foaf:name>ILRT staff</foaf:name>
<foaf:member>
<foaf:Person>
<foaf:name>Martin Poulter</foaf:name>
<foaf:homepage
rdf:resource="http://www.ilrt.bris.ac.uk/aboutus/staff/staffprofile/?s
earch=plmlp"/>
<foaf:workplaceHomepage
rdf:resource="http://www.ilrt.bris.ac.uk/">
</foaf:Person>
</foaf:member> </foaf:Group> </rdf:RDF>

Accept: 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"?>

<response><loaded>true</loaded></response>
```

14.5.1.14 Delete Context

Verb	URI	Description
DELETE	/semantic-workspaces-service/rest/workspaces/{WORKSPACE_NAME}/context/{CONTEXT_NAME}	Removes a context of a specific workspace

Response codes:

- HTTP/1.1 200 - If the context was successfully deleted from the workspace.
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
DELETE /semantic-workspaces-
service/rest/workspaces/test/context/testContext HTTP/1.1

Host: localhost:8080

Accept: 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"?>

<response><cleared>true</cleared></response>
```

14.5.1.15 List Contexts

Verb	URI	Description
GET	/semantic-workspaces-service/rest/workspaces/{WORKSPACE_NAME}/context/list	List the contexts included in a specific workspace

Response codes:

- HTTP/1.1 200 - If the contexts list was successfully retrieved from the workspace.
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
GET      /semantic-workspaces-service/rest/workspaces/test/context/list
HTTP/1.1

Host: localhost:8080

Accept: 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"?>

<response><contexts><context>http://semanticas.lab.fi-
ware.eu/semantic-
workspaces/testContext</context></contexts></response>
```

14.5.1.16 Add Statement

Verb	URI	Description
POST	/semantic-workspaces-service/rest/workspaces/{WORKSPACE_NAME}/context/{CONTEXT_NAME}/statement	Add a statement (RDF triple) into a specific workspace

Response codes:

- HTTP/1.1 200 - If the statement was successfully added into the workspace.
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
POST      /semantic-workspaces-
service/rest/workspaces/{WORKSPACE_NAME}/context/{CONTEXT_NAME}/statem
ent 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
```

14.5.1.17 *Remove Statement*

Verb	URI	Description
DELETE	/semantic-workspaces-service/rest/workspaces/{WORKSPACE_NAME}/context/{CONTEXT_NAME}/statement	Remove a statement (RDF triple) from a specific workspace

Response codes:

- HTTP/1.1 200 - If the statement was successfully deleted from the workspace.
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
DELETE /semantic-workspaces-
service/rest/workspaces/{WORKSPACE_NAME}/context/{CONTEXT_NAME}/statement 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
```

15 FIWARE ArchitectureDescription Data SemanticSupport OMV_Open_Specification

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](#)

16 FIWARE OpenSpecification Data StreamOriented

Name	FIWARE.OpenSpecification.Data.StreamOriented		
Chapter	Data/Context Management,		
Catalogue-Link Implementation	to <Kurento>	Owner	URJC NAEVATEC, Luis López Fernández

16.1 Preface

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

16.2 Copyright

Copyright © 2010-2014 by [URJC](#). All Rights Reserved.

16.3 Legal Notice

Please check the following [Legal Notice](#) to understand the rights to use these specifications.

16.4 Overview

The Stream Oriented GE provides a framework devoted to simplify the development of complex interactive multimedia applications through a rich family of APIs and toolboxes, which include:

- **Content API:** an API bringing all the power of Java EE technologies to the multimedia arena. This API will be particularly natural and intuitive for developers familiar with the Servlet API.
- **Media API:** a Java API exposing a toolbox of Media Elements that can be chained for creating complex multimedia processing pipelines. The abstraction and modularity of the Media API makes possible for non-expert developers to create complex and interoperable multimedia applications.

- **Open API:** a REST-like API based on JSON RPC 2.0 opening the Stream Oriented GE capabilities through a standard HTTP and websocket network connection.
- **Open Thrift API:** a network API exposing the Stream Oriented GE capabilities through Thrift RPCs. This API makes simple the creation of applications on any of the languages and frameworks supported by Thrift including Python, PHP, Node JS, Ruby, etc.
- **HTML5 SDK:** a toolbox of JavaScript libraries providing access to the Stream Oriented GE capabilities through any HTML5 compatible browser.

Thanks to these, the Stream Oriented GE provides developers with a set of robust end-to-end interoperable multimedia communication capabilities to deal with the complexity of transport, encoding/decoding, processing and rendering tasks in an easy and efficient way.

16.5 Basic Concepts

16.5.1 Signaling and media planes

The Stream Oriented GE, as most multimedia communication technologies out there, is built upon two concepts that are key to all interactive communication systems:

- **Signaling Plane.** Module in charge of the management of communications, that is, it provides functions for media negotiation, QoS parametrization, call establishment, user registration, user presence, etc.
- **Media Plane.** Module in charge of the media itself. So, functionalities such as media transport, media encoding/decoding and media processing are part of it.

16.5.2 Media elements and media pipelines

The Stream Oriented GE is based on two concepts that act as building blocks for application developers:

- **Media Element.** A Media element is a functional unit performing a specific action on a media stream. Media elements are a way of modularizing the capabilities of the Stream Oriented GE, so that every capability is represented as a self-contained “black box” (the media element) to the application developer, who does not need to understand the low-level details of the element for using it. Media elements are capable of receiving media from other elements (through media sinks) and of sending media to other elements (through media sources). Depending on their function, media elements can be split into different groups:
 - **Input Media Elements (IME):** Media elements capable of receiving media and injecting it into a pipeline. There are several types of IMEs. File IMEs take the media from a file, Network IMEs take the media from the network, and Capture IMES are capable of capturing the media stream directly from a camera or other kind of hardware resource.
 - **Processing Media Elements (PME):** Media elements in charge of transforming or analysing media. Hence there are PMEs for performing operations such as transcoding, mixing, muxing, analyzing, augmenting, etc.

- **Output Media Elements (OPE):** Media elements capable of taking a media stream out of the pipeline. Again, there are several types of OPEs specialized in files, network, screen, etc.



Figure 1.

A media element is a functional unit providing a specific media capability, which is exposed to application developers as a "black box"

- **Media Pipeline:** A Media Pipeline is a chain of media elements, where the output stream generated by one element (source) is fed into one or more other elements input streams (sinks). Hence, the pipeline represents a "machine" capable of performing a sequence of operations over a stream.

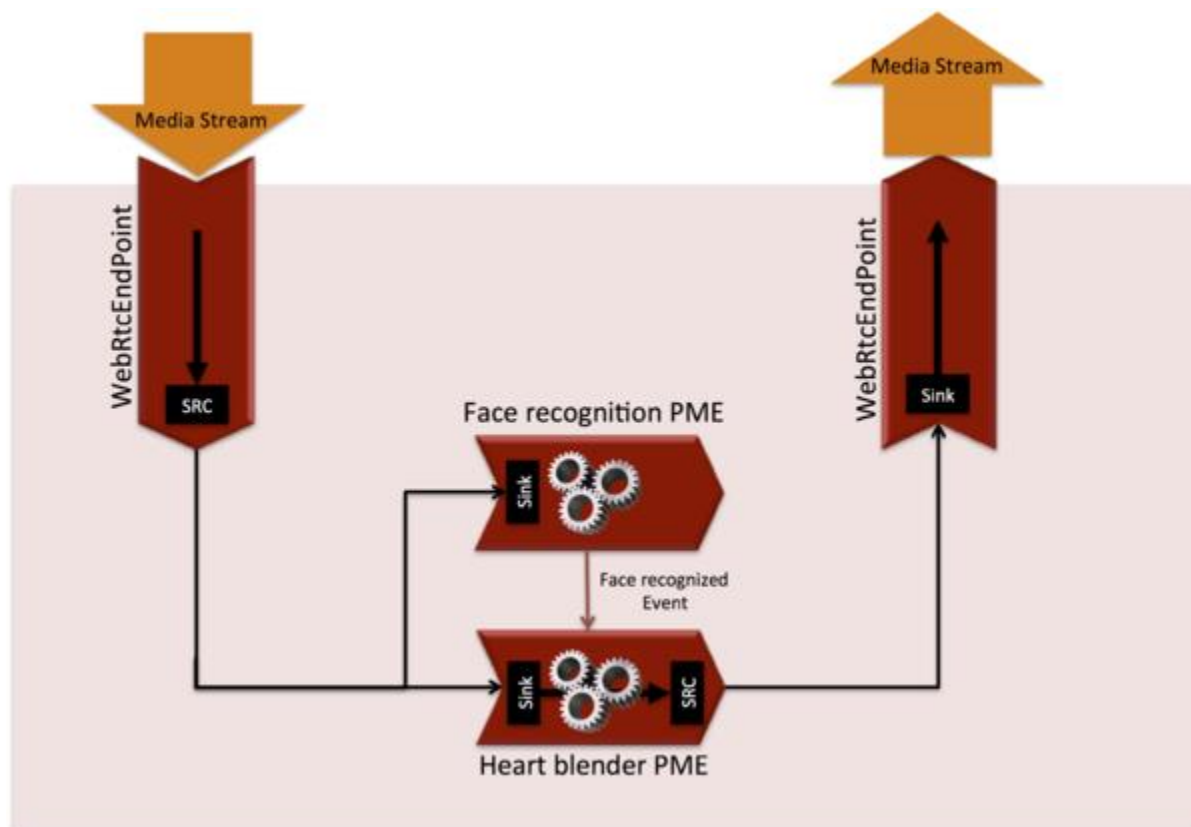


Figure 2.

Example of a Media Pipeline implementing an interactive multimedia application receiving media from a video source, injecting a love-heart animation in the video stream when a specific face has been recognized and sending the resulting media to a video sink

16.5.3 Agnostic media adaptor

Using the Stream Oriented GE APIs, developers are able to compose the available media elements, getting the desired pipeline. There is a challenge in this scheme, as different media elements might require different input media formats than the output produced by their preceding element in the chain. For example, if we want to connect a WebRTC (VP8 encoded) or a RTP (H.264/H.263 encoded) video stream to a face recognition media element implemented to read raw RGB format, a transcoding is necessary.

Developers, specially during the initial phases of application development, might want to simplify development and abstract this heterogeneity, the Stream Oriented GE provides an automatic converter of media formats called the “agnostic media adaptor”. Whenever a media element’s source is connected to another media element’s sink, our framework verifies if media adaption and transcoding is necessary and, in case it is, it transparently incorporates the appropriate transformations making possible the chaining of the two elements into the resulting pipeline.

Hence, this “agnostic media adaptor” capability fully abstracts all the complexities of media codecs and formats. This may significantly accelerate the development process, specially when developers are not multimedia technology experts. However, there is a price to pay. Transcoding may be a very CPU expensive operation. The inappropriate design of pipelines that chain media elements in a way that unnecessarily alternate codecs (e.g. going from H.264, to raw, to H.264 to raw again) will lead to very poor performance of applications.

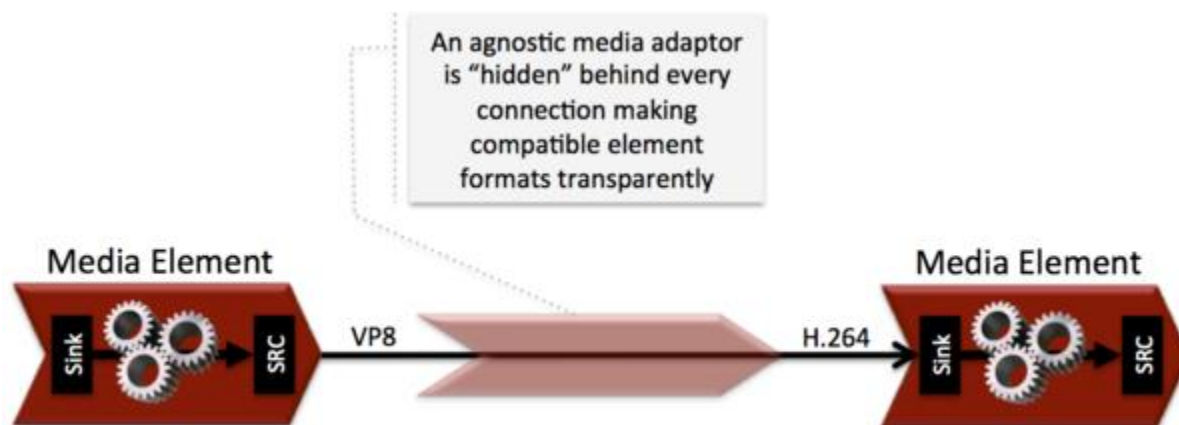


Figure 3.

The agnostic media capability adapts formats between heterogeneous media elements making transparent for application developers all complexities of media representation and encoding.

16.6 Stream-oriented GE Architecture

16.6.1 High level architecture

The conceptual representation of the GE architecture is shown in the following figure.

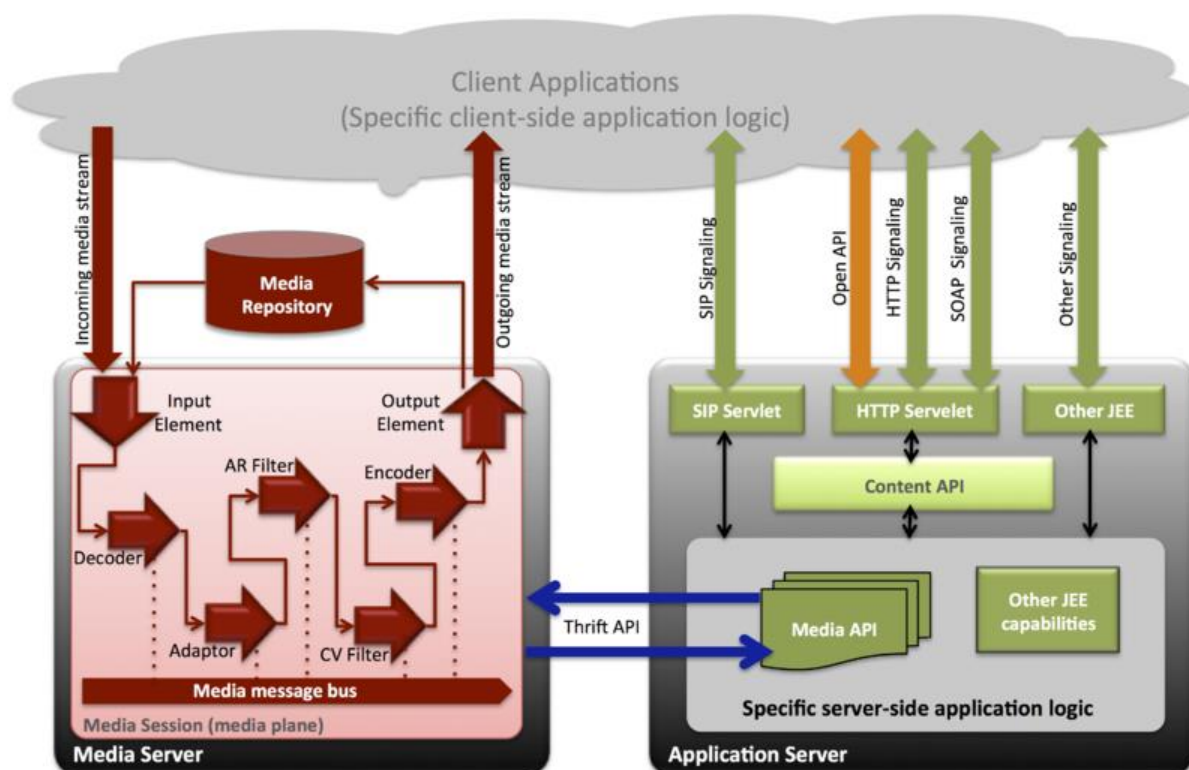


Figure 4.

The Stream Oriented GE architecture follows the traditional separation between signaling and media planes.

The right side of the picture shows the Application Server, which is in charge of the signaling plane and contains the business logic and connectors of the particular multimedia application being deployed. It is based on Java EE and includes well known and mature technologies such as HTTP and SIP Servlets, Web Services, database connectors, messaging services, etc. Thanks to this, this plane provides access to the multimedia signaling protocols commonly used by end-clients such as SIP, RESTful and raw HTTP based formats, SOAP, RMI, CORBA or JMS. These signaling protocols are used by client applications to command the creation of media sessions and to negotiate their desired characteristics on their behalf. Hence, this is the part of the architecture, which is in contact with application developers and, for this reason, it needs to be designed pursuing simplicity and flexibility. On the left side, we have the Media Server, which implements the media plane capabilities providing access to the low-level media features: media transport, media encoding/decoding, media transcoding, media mixing, media processing, etc. The Media Server must be capable of managing the multimedia streams with minimal latency and maximum throughput. Hence, in opposition to the Application Server, the Media Server does not need to be specifically designed for being simple to use or to control by application developers, but on the other hand, must be optimized for efficiency.

16.6.2 APIs and interfaces exposed by the architecture

The capabilities of the media plane (Media Server) and signaling plane (Application Server) are exposed through a number of APIs, which provide increasing abstraction levels. These APIs are nested in an onion-like layered architecture, where each level uses the services exposed by its immediate inner layer and is used by its outer layer, so that external levels are more abstract and easier to use by developers than internal levels. This scheme is shown in the picture below:

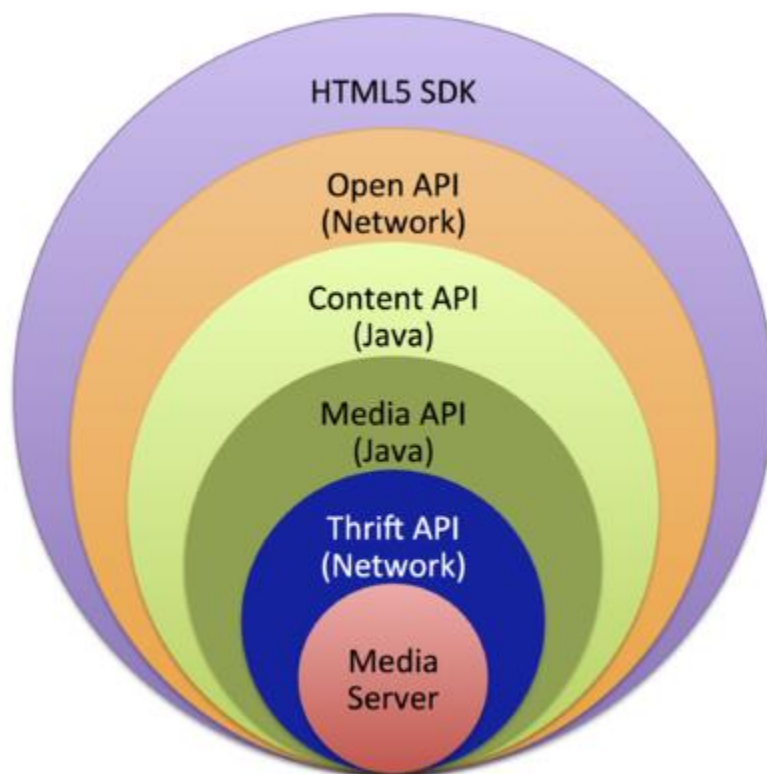


Figure 5.

The Stream Oriented Generic enabler has an onion-like architecture with APIs providing different abstraction levels.

Following this, the role of the different APIs can be summarized in the following way:

- **Thrift API:** Is a network API exposing the Media Server Capabilities through Thrift RPCs. Thrift acts as a middleware making possible the invocation of methods and constructors on the Media Server from stubs. In an architectural perspective, Thrift could be replaced by any other middleware providing synchronous and Asynchronous RPC invocation in an interoperable way (at least between C++ and Java) without requiring any modifications on the rest of API layers. This API makes possible the creation and management of media elements and pipelines by using references (ids). It is not a full abstract API given that non-trivial Media Server features such as distributed garbage collection and security mechanisms are explicitly exposed. Accessing the Thrift API is possible from any of the computer languages and frameworks supported by Thrift.

- **Media API:** Is a Java SE layer which consumes the Thrift API and exposes its capabilities through a simple-to-use modularity based on Java POJOs representing media elements and media pipelines. This API is abstract in the sense that all the non-intuitive inherent complexities of the internal Media Server workings are abstracted and developers do not need to deal with them when creating applications. Using the Media API only requires adding the appropriate dependency to a maven project or to download the corresponding jar into the application developer CLASSPATH. In the future, further Media APIs can be created exposing the same kind of modularity in other languages such as Python, C/C++, PHP, etc. It is important to remark that the Media API is a media-plane control API. In other words, its objective is to expose the capability of managing media objects, but it does not provide any signaling plane capabilities.
- **Content API:** Is a Java EE layer, which consumes the Media API and exposes its capabilities through a simple modularity based on two types of objects: *ContentHandlers* and *ContentSessions*. *ContentHandlers* are abstractions extending the Java EE Servlet API making possible the creation of multimedia applications just by managing signaling events happening into a session (e.g. *onContentRequest*, *onContentTerminated*, etc.) *ContentSessions* represent specific client applications accessing to the infrastructure and have an associated state. The Content API is a signaling plane API, which makes possible to react to signaling messages received from the client and to execute the appropriate application logic (e.g. authenticate, connect to a database, execute a web service, use the Media API, etc.) at the appropriate instants. Content API developers require a Java EE compatible Application Server like JBoss or Tomcat.
- **Open API:** is a network API exposing the capabilities of the Content API through a REST-like protocol based on the JSON RPC standard over http. To some extent, the Open API is the signaling protocol associated to the Content API. In addition, the Open API provides a mechanism for accessing and managing Media API capabilities directly over websocket transport.
- **HTML5 SDK:** is an SDK consuming the Open API and exposing all the capabilities of the framework to all kinds of clients providing the required HTML5 features (i.e. video tag, WebRTC, WebSockets and AJAX). Hence, the HTML5 SDK could, at least in principle, be used in server side infrastructures such as Node.js and in client side WWW browsers. Using the APIs exposed by this SDK requires a Stream Oriented GE server infrastructure (Media Server and Application Server) in execution. The deployment of the Content API, or any application using it, automatically brings to the Java EE container all the required JavaScript files that can be imported by the HTML5 application.

Details and examples on how to use these APIs can be found at the corresponding Stream Oriented GE Developer's Guides. From an architectural perspective, the only relevant aspect is that application developers can use any of these APIs for creating their multimedia enabled applications. This opens a wide spectrum of potential usage scenarios ranging from WWW applications (written using the HTML5 SDK), desktop applications (written using directly the Java Media API), distributed applications (written using Thrift or Open APIs, etc.) This idea is represented in the following picture:

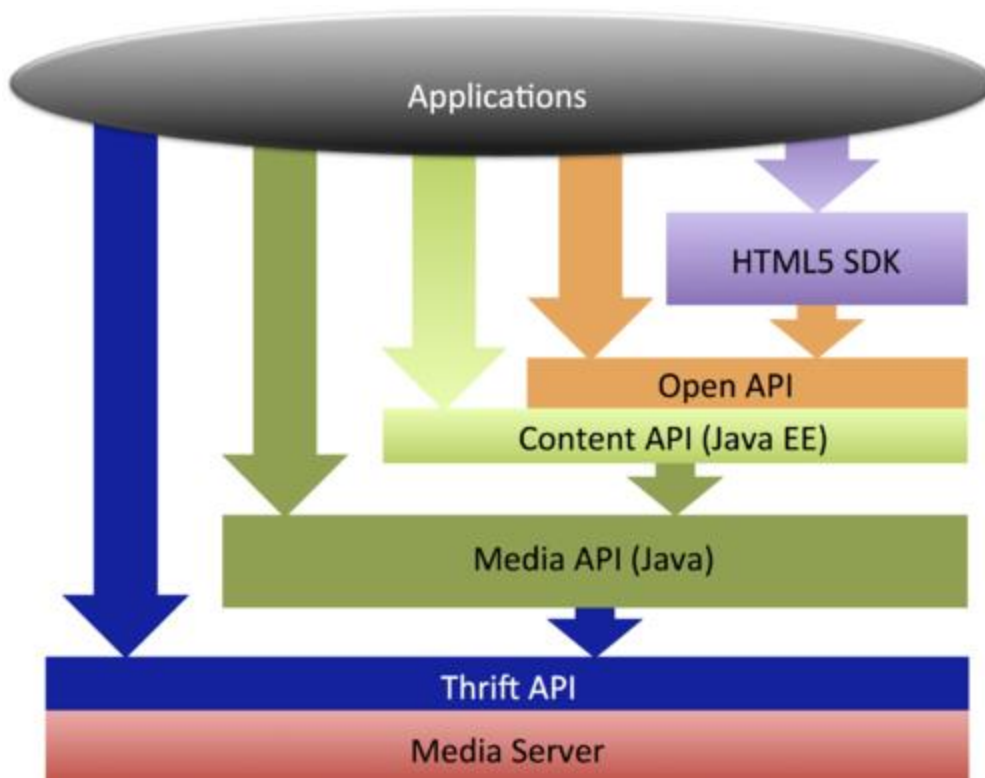


Figure 6.

Application developers can use any of the available layered APIs for creating their applications. Upper layers show higher abstraction and require lower the associated development effort. The arrows in the figure refer to method calls.

16.6.3 Creating applications on top of the Stream Oriented GE Architecture

The Stream Oriented GE Architecture has been specifically designed following the architectural principles of the WWW. For this reason, creating a multimedia applications basing on it is a similar experience to creating a web application using any of the popular web development frameworks.

At the highest abstraction level, web applications have an architecture comprised of three different layers:

- **Presentation layer:** Here we can find all the application code which is in charge of interacting with end users so that information is represented in a comprehensive way user input is captured. This usually consists on HTML pages.
- **Application logic:** This layer is in charge of implementing the specific functions executed by the application.
- **Service layer:** This layer provides capabilities used by the application logic such as databases, communications, security, etc.

Following this parallelism, multimedia applications created using the Stream Oriented GE also respond to the same architecture:

- **Presentation layer:** Is in charge of multimedia representation and multimedia capture. It is usually based on specific build-in capabilities of the client. For example, when creating a browser-based application, the presentation layer will use capabilities such as the <video> tag or the WebRTC PeerConnection and MediaStreams APIs.
- **Application logic:** This layer provides the specific multimedia logic. In other words, this layer is in charge of building the appropriate pipeline (by chaining the desired media elements) that the multimedia flows involved in the application will need to traverse.
- **Service layer:** This layer provides the multimedia services that support the application logic such as media recording, media ciphering, etc. The Media Server (i.e. the specific media elements) is the part of the Stream Oriented GE architecture in charge of this layer.

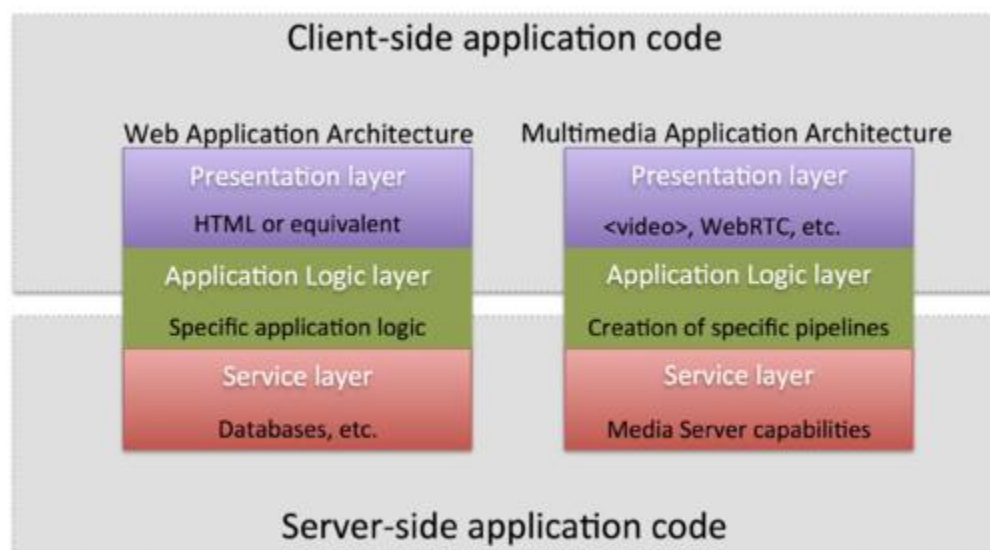


Figure 7.

Applications created using the Stream Oriented GE (right) have an equivalent architecture to standard WWW applications (left). Both types of applications may choose to place the application logic at the client or at the server code.

The interesting aspect of this discussion is that, as happens with WWW development, Stream Oriented GE applications always place the Presentation layer at the client side and the Service layer at the server side. However, the Application Logic layer, in both cases, can be located at either of the sides or even distributed between them.

This means that Stream Oriented GE developers can choose to include the code creating the specific media pipeline required by their applications at the client side (directly through Open network APIs or in a more abstract manner through the HTML5 SDK) or can place it at the server side (using for that the Content and Media APIs).

Both options are valid but each of them drives to different development styles. Having said this, it is important to note that in the WWW developers usually tend to maintain client side code as simple as possible, bringing most of their application logic to the server. Reproducing this kind of development experience, the most common (and recommended) way of using the Stream Oriented GE is by locating the multimedia application logic at the server side, so that the specific media pipelines are created using the Java Media API upon signaling events managed by the Content API. In the rest of this document, we assume that this is the standard way in which developers use the Stream Oriented APIs.

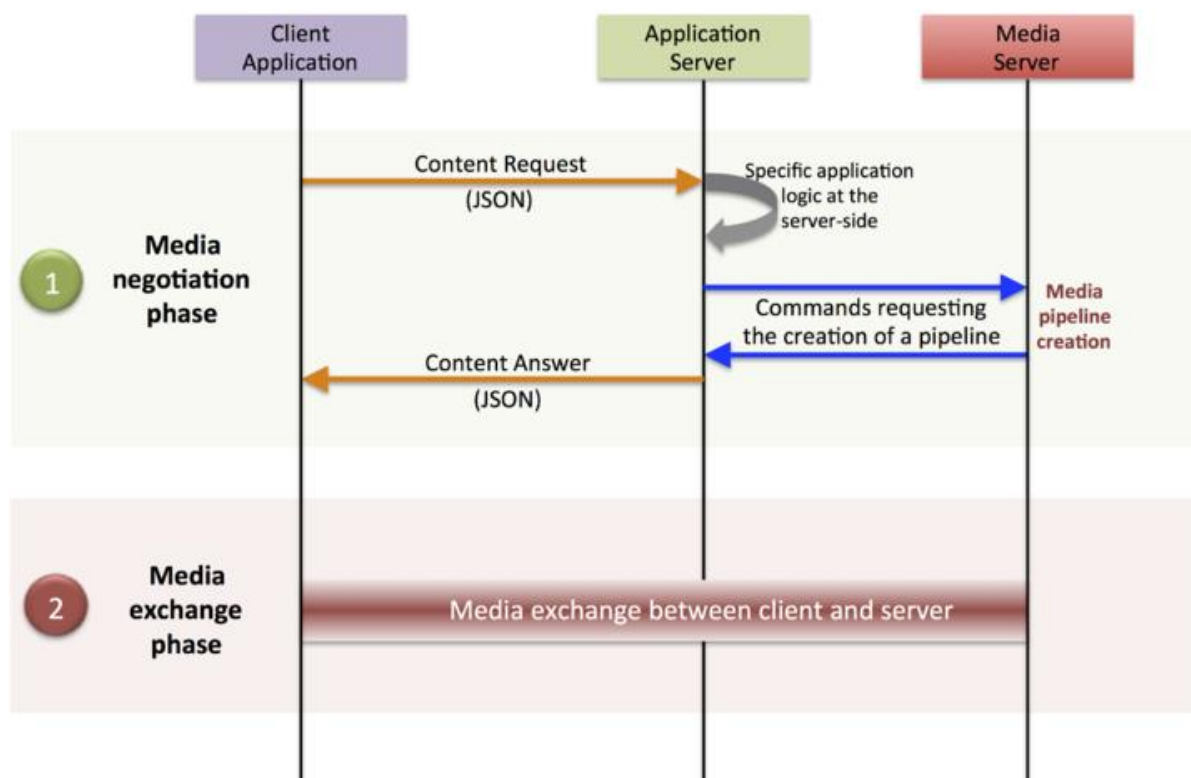
16.7 Main Interactions

16.7.1 Interactions from a generic perspective

As can be observed in Figure 4 above, a Stream Oriented GE application involves interactions among three main modules:

- **Client Application:** which involves the native multimedia capabilities of the client platform plus the specific client-side application logic consuming the client-side Stream Oriented GE APIs (i.e. HTML5 SDK, Open API, etc.)
- **Application Server:** which involves a Java EE application server and the server-side application logic consuming the server-side Stream Oriented GE APIs (i.e. Content API and Media API)
- **Media Server:** which receives commands for creating specific multimedia capabilities (i.e. specific pipelines adapted to the needs of specific applications)

The interactions maintained among these modules depend on the specificities of each application. However, in general, for most applications they can be reduced to the following conceptual scheme:

**Figure 8.**

Main interactions occur in two fases: negotiation and media exchange. Remark that the color of the different arrows and boxes is aligned with the architectural figures presented above, so that, for example, orange arrows show exchanges belonging to the Open API, blue arrows show exchanges belonging to the Thrift API, red boxes are associated to the Media Server and green boxes with the Application Server.

16.7.1.1 1. Media negotiation phase

As it can be observed, at a first stage, a client (a browser in a computer, a mobile application, etc.) issues a message requesting some kind of capability from the Stream Oriented GE. This message is based on a JSON RPC V2.0 representation and fulfills the Open API specification. It can be generated directly from the client application or, in case of web applications, indirectly consuming the abstract HTML5 SDK. For instance, that request could ask for the visualization of a given video clip.

When the Application Server receives the request, if appropriate, it will carry out the specific server side application logic, which can include Authentication, Authorization and Accounting (AAA), CDR generation, consuming some type of web service, etc.

After that, the Application Server processes the request and, according to the specific instructions programmed by the developer, commands the Media Server to instantiate the suitable media elements and to chain them in an appropriate media pipeline. Once the pipeline has been created successfully the server responds accordingly and the Application Server forwards the successful response to the client, showing it how and where the media service can be reached.

During the above mentioned steps no media data is really exchanged. All the interactions have the objective of negotiating the whats, hows, wheres and whens of the media exchange. For this reason, we call it the negotiation phase. Clearly, during this phase only signaling protocols are involved.

16.7.1.2 **2. Media exchange phase**

After that, a new phase starts devoted to producing the actual media exchange. The client addresses a request for the media to the Media Server using the information gathered during the negotiation phase. Following with the video-clip visualization example mentioned above, the browser will send a GET request to the IP address and port of the Media Server where the clip can be obtained and, as a result, an HTTP request with the media will be received.

Following the discussion with that simple example, one may wonder why such a complex scheme for just playing a video, when in most usual scenarios clients just send the request to the appropriate URL of the video without requiring any negotiation. The answer is straightforward. The Stream Oriented GE is designed for media applications involving complex media processing. For this reason, we need to establish a two-phase mechanism enabling a negotiation before the media exchange. The price to pay is that simple applications, such as one just downloading a video, also need to get through these phases. However, the advantage is that when creating more advanced services the same simple philosophy will hold. For example, if we want to add augmented reality or computer vision features to that video-clip, we just need to create the appropriate pipeline holding the desired media element during the negotiation phase. After that, from the client perspective, the processed clip will be received as any other video.

16.7.2 Specific interactions for commonly used services

Regardless of the actual type of session, all interactions follow the pattern described in section above. However, most common services respond to one of the following three main categories:

16.7.2.1 **HTTP content player sessions**

This type of session emerges when clients use the Stream Oriented GE to receive media through an HTTP response. The client sends a JSON request identifying the desired content and, as a result, it receives an URL where the content can be found. This URL is associated to a pipeline where the media processing logic is executed. The Application Server is in charge of commanding the creation of that media pipeline following the instructions provided by the application developer. The Application Server can interrogate that pipeline for obtaining the URL it is exposing to the world. This URL travels at the end

of the negotiation to the client, which can recover the stream by connecting to it. The following image shows the interactions taking place in this kind of session.

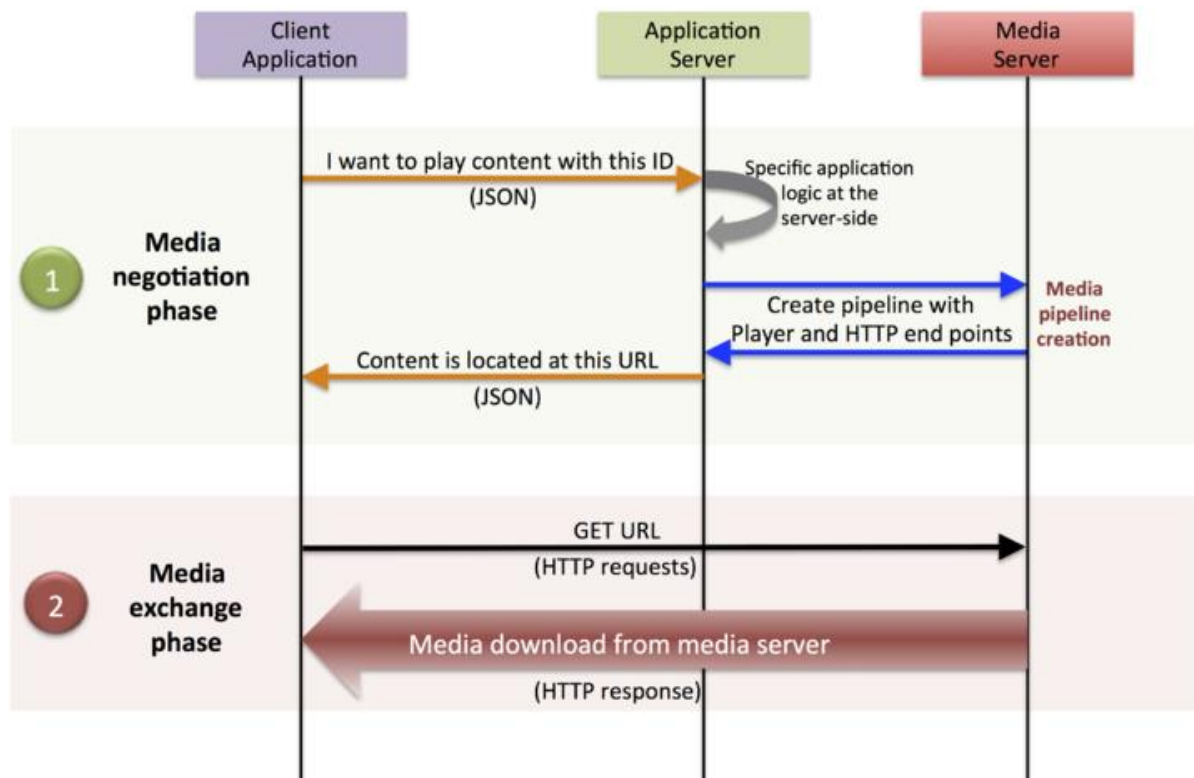


Figure 9.

Main interactions in a Stream Oriented GE session devoted to playing an HTTP media stream.

Clearly, the specific media stream that the client receives depends on the pipeline serving it. For HTTP content playing sessions, the usual pipeline may follow the scheme depicted in the figure below, where a video clip is recovered from a media repository (e.g. the file system) and it is fed into a filter performing specific processing on it (e.g. augmenting the media, recognizing objects of faces through computer vision, adding subtitles, modifying the color palette, etc.) At the end of the pipeline an element called “HttpGetEndpoint” adapts the media and sends it as an HTTP answer upon client requests. This basic pipeline can be modified by the developer adding additional elements at wish, which can be done creating the server-side application logic.

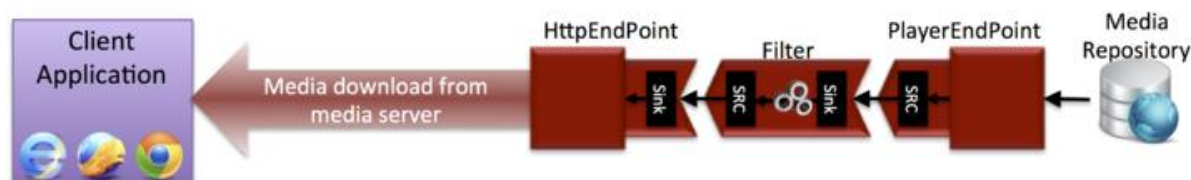


Figure 10.

Example of pipeline for an HTTP content player session.

16.7.2.2 *HTTP content recording sessions*

HTTP recording sessions are equivalent to playing sessions although, in this case, the media goes from the client to the server using POST HTTP method. The negotiation phase hence starts with the client requesting to upload the content and the Application Server creating the appropriate pipeline for doing it. This pipeline will always start with an `HttpPostEndpoint` element as the one shown in Figure 10. Further elements can be connected to that endpoint for filtering media, processing it or storing it into a media repository. The specific interactions taking place in this type of session are shown in the figure below



Figure 11.

Example of pipeline for an HTTP content recorder session.

16.7.2.3 *Content sessions for real time communications*

The Stream Oriented GE allows the establishment of real time multimedia session between a peer client and the Media Server directly through the use of RTP/RTCP or through WebRTC. In addition, the Media Server can be used to act as media proxy for making possible the communication among different peer clients, which are mediated by the Stream Oriented GE infrastructure. Hence, the GE can act as a conference bridge (Multipointing Control Unit), as a machine-to-machine communication system, as a

video call recording system, etc. As shown in the picture, the client exposes its media capabilities through an SDP (Session Description Protocol) payload encapsulated in a JSON object request. Hence, the Application Server is able to instantiate the appropriate media element (either RTP or WebRTC end points), and to require it to negotiate and offer a response SDP based on its own capabilities and on the offered SDP. When the answer SDP is obtained, it is given back to the client and the media exchange can be started. The interactions among the different modules are summarized in the following picture

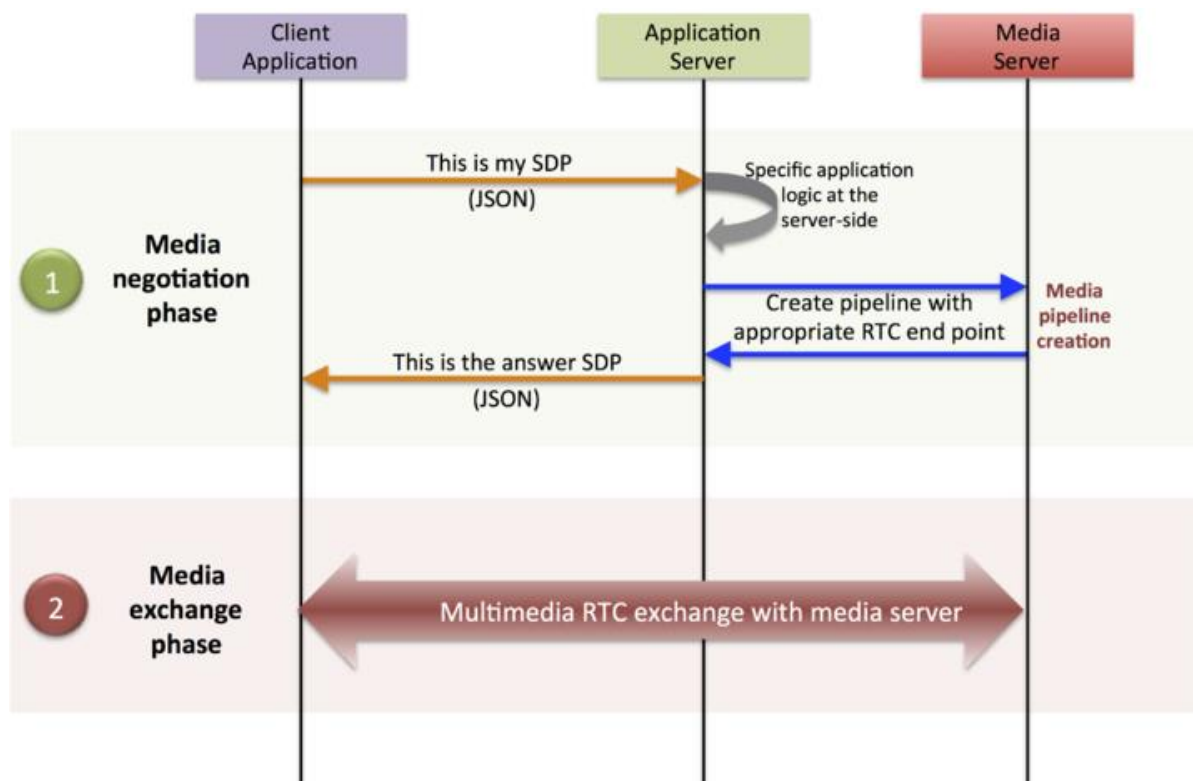


Figure 12.

Interactions taking place in a Real Time Communications (RTC) session. During the negotiation phase, a Session Description Protocol (SDP) message is exchanged offering the capabilities of the client. As a result, the Media Server generates an SDP answer that can be used by the client for establishing the media exchange.

As with the rest of examples shown above, the application developer is able to create the desired pipeline during the negotiation phase, so that the real time multimedia stream is processed accordingly to the application needs. Just as an example, imagine that we want to create a WebRTC application recording the media received from the client and augmenting it so that if a human face is found, a hat will be rendered on top of it. This pipeline is schematically shown in the figure below, where we assume that the Filter element is capable of detecting the face and adding the hat to it.

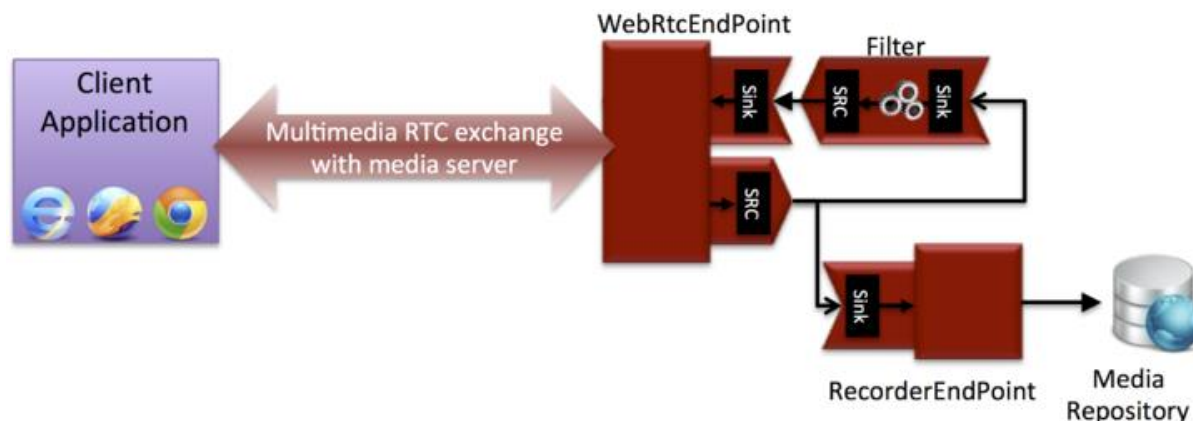


Figure 13.

During the negotiation phase, the application developer can create a pipeline providing the desired specific functionality. For example, this pipeline uses a WebRtcEndpoint for communicating with the client, which is connected to a RecorderEndpoint storing the received media stream and to an augmented reality filter, which feeds its output media stream back to the client. As a result, the end user will receive its own image filtered (e.g. with a hat added onto her head) and the stream will be recorded and made available for further recovery into a repository (e.g. a file).

16.8 Basic Design Principles

The Stream-oriented GE is designed based on the following main principles:

- Signaling and Media are two separate planes and the GE is designed according to that split.
- Media and Application servers can be collocated or distributed among different machines.
- A single Application Server can invoke the services of more than one Media Server. The opposite also applies, that is, a Media Server can attend the requests of more than one Application Server.
- The GE is suitable to be integrated into cloud environments to act as a PaaS (Platform as a Service).
- Chaining Media Elements in the way of Media Pipelines is an intuitive approach to challenge the complexities of multimedia communications.
- In a Media Pipeline there exists a global clock suitable for the synchronization of different media elements
- Developers do not need to be aware of internal Media Server complexities, all the applications are deployed in the JEE Application Server.

- Client-side SDKs are provided to simplify the application development on smartphones and WWW desktop environments.
- The GE provides end-to-end communication capabilities so developers do not need to deal with the complexity of transporting, encoding/decoding and rendering media on client devices.
- The GE enables not only interactive interpersonal communications (e.g. Skype-like with conversational call push/reception capabilities), but also human- to-machine (e.g. Video on Demand through real-time streaming) and machine-to-machine (e.g. remote video recording, multisensory data exchange) communications.
- Modularization achieved through media elements and pipelines allows defining the media processing functionality of an application through a “graph-oriented” language, where the application developer is able to create the desired logic just by chaining the appropriate functionalities.
- The GE is able to generate rich and detailed information for QoS monitoring, billing and auditing.
- The GE supports seamless IMS integration.
- The GE provides a transparent media adaptation layer to make the convergence among different devices having different requirements in terms of screen size, power consumption, transmission rate, etc. possible.

16.9 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.

16.9.1 Open API Specifications

- [StreamOriented Open API Specification](#)

16.10 Re-utilised Technologies/Specifications

The StreamOriented GE relies on the following specifications:

- HTTP
- TLS
- JSON-RPC V2.0
- WebSockets
- RTP
- SRTP
- DTLS
- H.264
- MPEG4
- VP8
- Opus
- AMR
- WebRTC
- HTML5

On the other hand, Kurento GEi reuses the following technologies:

- JBoss
- Spring
- GStreamer
- OpenCV
- Thrift
- maven
- Selenium

16.11 Terms and definitions

This section comprises a summary of terms and definitions introduced during the previous sections. It intends to establish a vocabulary that will be help to carry out discussions internally and with third

parties (e.g., Use Case projects in the EU FP7 Future Internet PPP). For a summary of terms and definitions managed at overall FI-WARE level, please refer to [FIWARE Global Terms and Definitions](#)

- **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 **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 be 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 two 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.

17 StreamOriented Open API Specification

17.1.1 Legal Notice

Please check the [Legal Notice](#) to understand the rights to use FI-WARE Open Specifications.

17.1.2 Introduction

The Stream Oriented API is a REST-like resource-oriented API accessed via HTTP/HTTPS that uses JSON-RPC V2.0 based representations for information exchange. This document describes the API exposed by the Application Server as defined in the [GE Architecture Description](#).

17.1.3 Intended Audience

This specification is intended for both software developers and implementors of this GE. For developers, this document details the REST-like API to build interactive multimedia applications compliant with the [GE Architecture Description](#). Implementors can build their GEi APIs based on the information contained in this specification.

Before reading this document it is recommended to read first the [GE Architecture Description](#) and the [Programmers Guide](#). Moreover, the reader should be also familiar with:

- REST web services
- [HTTP/1.1 \(RFC2616\)](#)
- [WebSockets \(RFC6455\)](#)
- [JSON](#) data serialization format.

17.1.4 Conventions used in this 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*.
- Variables are represented between brackets, e.g. *{id}*, and in italic font. When the reader finds one, it can assume that the variable can be changed for any value.

17.2 API General Features

17.2.1 Authentication

Stream-oriented GE Open API can be offered in two modes: with or without authentication. When authentication is enabled, each request requires the inclusion of an access token. This access token will be granted to the client browser by the IdM GE, as described in [the IdM GE Specification](#). To obtain the token, the client application will redirect the client browser to the IdM GE to perform the authentication and authorization steps of OAuth2. The token thus obtained must be included when stabilising Open API websocket connection to server.

17.2.2 Representation Transport

Resource representation may be transmitted between client and server by using directly HTTP 1.1 protocol, as defined by IETF RFC-2616 or through a WebSocket transport, as defined by IETF RFC-6455. 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.

17.2.3 Representation Format

Stream Oriented REST-like APIs support JSON as representation format for request and response parameters following the recommendations in the proposal [JSON-RPC over HTTP](#).

When using HTTP 1.1 transports, the format of the requests is specified by using the *Content-Type* header with a value of *application/json-rpc* and is required for requests containing a body. The format required for the response is specified in the request by setting the *Accept* header to the value *application/json-rpc*, that is, request and response bodies are serialized using the same format.

17.2.3.1 ***Request object***

An *RPC call* is represented by sending a *Request object* to a server. The *Request object* has the following members:

- *jsonrpc*: a string specifying the version of the JSON-RPC protocol. It must be exactly "2.0".
- *method*: a string containing the name of the method to be invoked.
- *params*: a structured value that holds the parameter values to be used during the invocation of the method.

- *id*: an identifier established by the client that contains a string or number. The server must reply with the same value in the *Response object*. This member is used to correlate the context between both objects.

17.2.3.2 **Successful Response object**

When an *RPC call* is made the server replies with a *Response object*. In the case of a successful response, the *Response object* will contain the following members:

- *jsonrpc*: a string specifying the version of the JSON-RPC protocol. It must be exactly "2.0".
- *result*: its value is determined by the method invoked on the server. In case the connection is rejected, it's returned an object with a *rejected* attribute containing an object with a *code* and *message* attributes with the reason why the session was not accepted, and no *sessionId* is defined.
- *id*: this member is mandatory and it must match the value of the *id* member in the *Request object*.

17.2.3.3 **Error Response object**

When an *RPC call* is made the server replies with a *Response object*. In the case of an error response, the *Response object* will contain the following members:

- *jsonrpc*: a string specifying the version of the JSON-RPC protocol. It must be exactly "2.0".
- *error*: an object describing the error through the following members:
 - *code*: an integer number that indicates the error type that occurred.
 - *message*: a string providing a short description of the error.
 - *data*: a primitive or structured value that contains additional information about the error. It may be omitted. The value of this member is defined by the server.
- *id*: this member is mandatory and it must match the value of the *id* member in the *Request object*. If there was an error in detecting the *id* in the *Request object* (e.g. Parse Error/Invalid Request), it equals to null.

17.2.4 Limits

Media processing is very CPU intensive and therefore the developer should be aware that the creation of multiple simultaneous sessions can exhaust server resources. Thus, limits are not specified for 1.0.0 version of the Stream Oriented GE.

17.2.5 Extensions

Querying extensions is not supported in current version of the Stream Oriented GE.

17.3 API Specification

This section details the actual APIs of each of the managers defined in this GE, namely, the Content Manager API and the Media Manager API. It is recommended to review the [Programmers Guide](#) before proceeding with this section. The Stream Oriented GE API is split into two different sub-APIs, which satisfy different types of requirements: the Content API and the Media API. The following sections introduce both.

17.3.1 Content API

The Content API is based on HTTP 1.1. transports and is exposed in the form of four services: *HttpPlayer*, *HttpRecorder*, *RtpContent* and *WebRtcContent* described in the following subsections.

17.3.1.1 *HttpPlayer*

This service allows requesting a content to be retrieved from a Media Server using HTTP pseudostreaming.

Verb	URI	Description
POST	<code>/{CONTEXT-ROOT}/{APP_LOGIC_PATH}/{ContentID}</code>	Performs an RPC call regarding <code>{ContentID}</code> . The <i>Request object</i> is processed by the <i>HttpPlayer</i> application handler tied to <code>{APP_LOGIC_PATH}</code> in the <code>{CONTEXT-ROOT}</code> of the application.

The *Request object* (body of the HTTP request) can contain one of these four methods: *start*, *poll*, *execute*, and *terminate*.

(1)*start*

Requests the retrieval of the content. The parameter *constraints* indicates the kind of media (audio or/and video) to be received. In the case of *HttpPlayer*, the values for these constraints for audio and video should be *recvonly*. The following example shows a *Request object* requesting to receive audio and video:

```
{
  "jsonrpc": "2.0",
  "method": "start",
```

```
"params":
{
  "constraints":
  {
    "audio": "recvonly",
    "video": "recvonly"
  }
},
"id": 1
}
```

The *Response object* contains a *sessionId* to identify the session and the actual URL to retrieve the content from:

```
{
  "jsonrpc": "2.0",
  "result":
  {
    sessionId": 1234,
    "url": "http://mediaserver/a13e9469-fec1-4eee-b40c-8cd90d5fc155"
  },
  "id": 1
}
```

(2)poll

This method allows emulating *push events* coming from the server by using a technique known as *long polling*. With long polling, the client requests information from the server in a way similar to a normal polling; however, if the server does not have any information available for the client, instead of sending an empty response, it holds the request and waits for information to become available until a timeout is expired. If the timeout is expired before any information has become available the server sends an empty response and the client re-issues a new poll request. If, on the contrary, some information is

available, the server pushes that information to the client and then the client re-issues a new poll request to restart the process.

The *params* includes an object with only a *sessionId* attribute containing the ID for this session.

```
{
  "jsonrpc": "2.0",
  "method": "poll",
  "params":
  {
    "sessionId": 1234
  },
  "id": 1
}
```

The *Response object* has a *contentEvents* attribute containing an array with the latest MediaEvents, and a *controlEvents* attribute containing an array with the latest control events for this session, or an empty object if none was generated. Each control event can has an optional data attribute containing an object with a *code* and a *message* attributes.

```
{
  "jsonrpc": "2.0",
  "result":
  {
    "contentEvents":
    [
      { "type": "typeOfEvent1",
        "data": "dataOfEvent1" },
      { "type": "typeOfEvent2",
        "data": "dataOfEvent2" }
    ]
  }
}
```



```
],  
  "controlEvents":  
  [  
    {  
      "type": "typeOfEvent1",  
      "data":  
      {  
        "code": 1,  
        "message": "license plate"  
      }  
    }  
  ]  
},  
  "id": 1  
}
```

(3)execute

Execute a command on the server. The *param* object has a *sessionId* attribute containing the ID for this session, and a *command* object with a *type* string attribute for the command type and a *data* attribute for the command specific parameters.

```
{  
  "jsonrpc": "2.0",  
  "method": "execute",  
  "params":  
  {  
    "sessionId": 1234,  
    "command":
```

```
{
  "type": "commandType",
  "data": ["the", "user", "defined", "command", "parameters"]
},
{id": 1
}
```

The *Response object* is an object with only a *commandResult* attribute containing a string with the command results.

```
{
  "jsonrpc": "2.0",
  "result":
  {
    "commandResult": "Everything has gone alright"
  },
  "id": 1
}
```

(4)terminate

Requests the termination of the session identified by *sessionId* so the server can release the resources assigned to it:

```
{
  "jsonrpc": "2.0",
  "method": "terminate",
  "params":
  {
```

```

    "sessionId": 1234,
    "reason":
    {
        "code": 1,
        "message": "User ended session"
    }
}

```

The *Response object* is an empty object:

```

{
    "jsonrpc": "2.0",
    "result": {},
    "id": 2
}

```

(5) Simplified alternative approach

The *HttpPlayer* service just described is consistent with the rest of APIs defined in the Stream Oriented GE. However, it is recommended to also expose a simpler API as described here not requiring the use of JSON.

Verb	URI	Description
GET	<code>/{{CONTEXT-ROOT}}/{{APP_LOGIC_PATH}}/{{ContentID}}</code>	Requests <code>{{ContentID}}</code> to be served according to the application handler tied to <code>{{APP_LOGIC_PATH}}</code> in the <code>{{CONTEXT-ROOT}}</code> of the application

Successful Response Codes: 200 OK, 307 Temporary Redirect (to the actual content).

Error Response Codes: 404 Not Found, 500 Internal Server Error.

17.3.1.2 *HttpRecorder*

This service allows the upload of a content through HTTP to be stored in a Media Server.

Verb	URI	Description
POST	<code>/{CONTEXT-ROOT}/{APP_LOGIC_PATH}/{ContentID}</code>	Performs an RPC call regarding <code>{ContentID}</code> . The <i>Request object</i> is processed by the <i>HttpRecorder</i> application handler tied to <code>{APP_LOGIC_PATH}</code> in the <code>{CONTEXT-ROOT}</code> of the application.

The *Request object* (body of the HTTP request) can contain one of these four methods: *start*, *poll*, *execute*, and *terminate*.

(1)start

Requests the storage of the content. The parameter *constraints* indicates the kind of media (audio or/and video) to be sent. In the case of *HttpRecorder*, the values for these constraints for audio and video should be *sendonly*. The following example shows a *Request object* requesting to send audio and video:

```
{
  "jsonrpc": "2.0",
  "method": "start",
  "params":
  {
    "constraints":
    {
      "audio": "sendonly",
      "video": "sendonly"
    }
  },
  "id": 1
}
```

The *Response object* contains a *sessionId* to identify the session and the actual URL to upload the content to:

```
{
```

```

    "jsonrpc": "2.0",
    "result":
    {
        "url": "http://mediaserver/a13e9469-fec1-4eee-b40c-8cd90d5fc155",
        "sessionId": 1234
    },
    "id": 1
}

```

(2)poll, execute, and terminate

These operations work in the same way than *HttpPlayer*. Therefore, for an example of *Request object* and *Response object* see the sections of *poll*, *execute*, and *terminate* respectively in *HttpPlayer*.

(3)Simplified alternative approach

The *HttpRecorder* service just described is consistent with the rest of APIs defined in the Stream Oriented GE. However, it is recommended to also expose a simpler API as described here not requiring the use of JSON.

Verb	URI	Description
POST	<code>/{{CONTEXT-ROOT}}/{{APP_LOGIC_PATH}}/{{ContentID}}</code>	Uploads <code>{ContentID}</code> to be stored according to the application handler tied to <code>{APP_LOGIC_PATH}</code> in the <code>{CONTEXT-ROOT}</code> of the application

The request body of this method is the content to be uploaded.

Successful Response Codes: 200 OK, 307 Temporary Redirect (to the actual storage server)

Error Response Codes: 404 Not Found, 500 Internal Server Error

17.3.1.3 *RtpContent*

This service allows establishing an *RTP content session* between the client performing the request and a Media Server.

Verb	URI	Description
POST	<code>/{{CONTEXT-ROOT}}/{{APP_LOGIC_PATH}}/{{ContentID}}</code>	Performs an RPC call regarding <code>{ContentID}</code> . The <i>Request object</i> is processed by the <i>RTPContent</i> application handler tied to <code>{APP_LOGIC_PATH}</code> in the

	{ContentID}	{CONTEXT-ROOT} of the application.
--	-------------	------------------------------------

The *Request object* (body of the HTTP request) can contain one of these four methods: *start*, *poll*, *execute*, and *terminate*.

(1)start

Requests the establishment of the RTP session. The parameter *sdp* contains the client SDP (Session Description Protocol) offer, that is, a description of the desired session from the caller's perspective. The parameter *constraints* indicates the media (audio or/and video) to be received, sent, or sent and received by setting their values to *recvonly*, *sendonly*, *sendrecv* or *inactive*. The following example shows a *Request object* requesting bidirectional audio and video (i.e. *sendrecv* for both audio and video):

```
{
  "jsonrpc": "2.0",
  "method": "start",
  "params":
  {
    "sdp": "Contents_of_Caller_SDP",
    "constraints":
    {
      "audio": "sendrecv",
      "video": "sendrecv"
    }
  },
  "id": 1
}
```

The *Response object* contains the Media Server SDP answer, that is, a description of the desired session from the callee's perspective, and a *sessionId* to identify the session:

```
{
```

```

    "jsonrpc": "2.0",
    "result":
    {
        "sdp": "Contents_of_Callee_SDP",
        "sessionId": 1234
    },
    "id": 1
}

```

(2)poll, execute, and terminate

These operations work in the same way than *HttpPlayer* and *HttpRecorder*. Therefore, for an example of *Request object* and *Response object* see the sections of *poll*, *execute*, and *terminate* respectively in *HttpPlayer*.

17.3.1.4 **WebRtcContent**

Conceptually, *RtpContent* and *WebRtcContent* are very similar, the main difference is the underlying protocol to exchange media, so all the descriptions in the section above apply to *WebRtcContent*.

17.3.2 Media API

The Media API provides full control of the Media Server through *Media Elements*, which are the building blocks providing a specific media functionality. They are used to send, receive, process and transform media. The Media API provides a toolbox of Media Elements ready to be used. It also provides the capability of creating *Media Pipelines* by joining Media Elements of the toolbox.

The Media API requires full-duplex communications between client and server infrastructure. For this reason, the Media API is based on WebSocket transports and not on plain HTTP 1.1 transports as the Content API does.

Previous to issuing commands, the Media API client requires establishing a WebSocket connection with the server infrastructure.

Verb	URI	Description
NA	<i>ws://{SERVER_IP}:{SERVER_PORT}/thrift/ws/websocket</i>	Establishment of WebSocket connection for the exchange of full-duplex JSON-RPC messages.

Once the WebSocket has been established, the Media API offers five different types of request/response messages:

- *create*: Instantiates a new pipeline or media element in the media server.
- *invoke*: Calls a method of an existing media element.
- *subscribe*: Creates a subscription to a media event in a media element.
- *unsubscribe*: Removes an existing subscription to a media event.
- *release*: Explicit termination of a media element.

The Media API allows to servers send requests to clients:

- *onEvent*: This request is sent from server to clients when a media event occurs.

17.3.2.1 **Create**

Create message requests the creation of an element of the MediaAPI toolbox. The parameter *type* specifies the type of the object to be created. The parameter *creationParams* contains all the information needed to create the object. Each object type needs different *creationParams* to create the object. These parameters are defined later in this document. Finally, a *sessionId* parameter is included with the identifier of the current session. The value of this parameter is sent by the server to the client in each response to the client. Only the first request from client to server is allowed to not include the *sessionId* (because at this point is unknown for the client).

The following example shows a Request object requesting the creation of an object of the type *PlayerEndPoint* within the pipeline 6829986 and uri <http://host/app/video.mp4> in the session c93e5bf0-4fd0-4888-9411-765ff5d89b93:

```
{
  "jsonrpc": "2.0",
  "method": "create",
  "params": {
    "type": "PlayerEndPoint",
    "creationParams": {
      "pipeline": "6829986",
```



```
    "uri": "http://host/app/video.mp4"
  },
  "sessionId": "c93e5bf0-4fd0-4888-9411-765ff5d89b93"
},
"id": 1
}
```

The *Response object* contains the object id of the new object in the field *value*. This object id has to be used in other requests of the protocol (as we will describe later). As stated before, the *sessionId* is also returned in each response.

The following example shows a typical response to a create message:

```
{
  "jsonrpc": "2.0",
  "result": {
    "value": "442352747",
    "sessionId": "c93e5bf0-4fd0-4888-9411-765ff5d89b93"
  },
  "id": 1
}
```

17.3.2.2 *Invoke*

Invoke message requests the invocation of an operation in the specified object. The parameter *object* indicates the id of the object in which the operation will be invoked. The parameter *operation* carries the name of the operation to be executed. Finally, the parameter *operationParams* has the parameters needed to execute the operation. The object specified has to understand the operation name and parameters. Later in this document it is described the valid operations for all object types.

The following example shows a *Request object* requesting the invocation of the operation *connect* on the object *442352747* with parameter *sink 6829986*. The *sessionId* is also included as is mandatory for all requests in the session (except the first one).

```
{
  "jsonrpc": "2.0",
  "method": "invoke",
  "params": {
    "object": "442352747",
    "operation": "connect",
    "operationParams": {
      "sink": "6829986"
    },
    "sessionId": "c93e5bf0-4fd0-4888-9411-765ff5d89b93"
  },
  "id": 2
}
```

The *Response object* contains the value returned while executing the operation invoked in the object or nothing if the operation doesn't return any value.

The following example shows a typical response while invoking the operation *connect* (that doesn't return anything):

```
{
  "jsonrpc": "2.0",
  "result": {
    "sessionId": "c93e5bf0-4fd0-4888-9411-765ff5d89b93"
  },
  "id": 2
}
```

17.3.2.3 *Release*

Release message requests the release of the specified object. The parameter *object* indicates the id of the object to be released.

```
{
  "jsonrpc": "2.0",
  "method": "release",
  "params": {
    "object": "442352747",
    "sessionId": "c93e5bf0-4fd0-4888-9411-765ff5d89b93"
  },
  "id": 3
}
```

The *Response object* only contains the *sessionId*. The following example shows the typical response of a release request:

```
{
  "jsonrpc": "2.0",
  "result": {
    "sessionId": "c93e5bf0-4fd0-4888-9411-765ff5d89b93"
  },
  "id": 3
}
```

17.3.2.4 *Subscribe*

Subscribe message requests the subscription to a certain kind of events in the specified object. The parameter *object* indicates the id of the object to subscribe for events. The parameter *type* specifies the type of the events. If a client is subscribed for a certain type of events in an object, each time an event is

fired in this object, a request with method *onEvent* is sent to the client. This kind of request is described few sections later.

The following example shows a *Request object* requesting the subscription of the event type *EndOfStream* on the object *311861480*. The *sessionId* is also included.

```
{
  "jsonrpc": "2.0",
  "method": "subscribe",
  "params": {
    "object": "311861480",
    "type": "EndOfStream",
    "sessionId": "c93e5bf0-4fd0-4888-9411-765ff5d89b93"
  },
  "id": 4
}
```

The *Response object* contains the subscription identifier. This value can be used later to remove this subscription.

The following example shows the response of subscription request. The *value* attribute contains the subscription id:

```
{
  "jsonrpc": "2.0",
  "result": {
    "value": "353be312-b7f1-4768-9117-5c2f5a087429",
    "sessionId": "c93e5bf0-4fd0-4888-9411-765ff5d89b93"
  },
  "id": 4
}
```

```
}
```

17.3.2.5 *Unsubscribe*

Unsubscribe message requests the cancelation of a previous event subscription. The parameter subscription contains the *subscription* id received from the server when the subscription was created.

The following example shows a *Request object* requesting the cancelation of the subscription *353be312-b7f1-4768-9117-5c2f5a087429*.

```
{
  "jsonrpc": "2.0",
  "method": "unsubscribe",
  "params": {
    "subscription": "353be312-b7f1-4768-9117-5c2f5a087429",
    "sessionId": "c93e5bf0-4fd0-4888-9411-765ff5d89b93"
  },
  "id": 5
}
```

The *Response object* only contains the *sessionId*. The following example shows the typical response of an unsubscription request:

```
{
  "jsonrpc": "2.0",
  "result": {
    "sessionId": "c93e5bf0-4fd0-4888-9411-765ff5d89b93"
  },
  "id": 5
}
```

17.3.2.6 *onEvent*

When a client is subscribed to a type of events in an object, the server send an *onEvent* notification each time an event of that type is fired in the incumbent object. This is possible because the Media API is implemented with websockets and there is a full duplex channel between client and server. The notification that server send to client has all the information about the event:

- *data*: Information about this specific of this type of event.
- *source*: the object source of the event.
- *type*: The type of the event.
- *subscription*: subscription id for which the event is fired.

The following example shows a notification sent for server to client to notify an event of type *EndOfStream* in the object *311861480* with subscription *353be312-b7f1-4768-9117-5c2f5a087429*.

```
{
  "jsonrpc": "2.0",
  "method": "onEvent",
  "params": {
    "value": {
      "data": {
        "source": "311861480",
        "type": "EndOfStream"
      },
      "object": "311861480",
      "subscription": "353be312-b7f1-4768-9117-5c2f5a087429",
      "type": "EndOfStream",
    },
    "sessionId": "4f5255d5-5695-4e1c-aa2b-722e82db5260"
  }
}
```

```
}  
  
}
```

In jsonrpc format, the notifications are different from request in that notifications haven't got an id field. Also, the notifications cannot be responded. For this reason, in the example before, there is no id in the message.

17.3.2.7 *Error responses*

If errors arise processing a request, there is a generic error response, in which an error code and a description message is sent, as follows:






```
{  
  "jsonrpc": "2.0",  
  "error":  
  {  
    "code": -32601,  
    "message": "Error description"  
  },  
  "id": 2  
}
```









17.3.2.8 *Media Element toolbox*




The Media Element toolbox provided by Media API is divided into Endpoints, Filters and Hubs.

- Endpoint offers capabilities to work with protocols and codecs ([HttpEndpoint](#), [RtpEndpoint](#) and [WebRtcEndpoint](#)) and also media repository handling ([PlayerEndpoint](#) and [RecorderEndpoint](#)).
- Filters are responsible of media processing, such as computer vision (Face detection, pointer tracking or bar and QR code reading) and augmented reality (Chroma filtering or face overlay filtering).
- Hubs offer capabilities to connect several inputs and outputs and create different types of connections between them ([Composite](#), [Dispatcher](#), [DispatcherOneToMany](#)).

Therefore, the Open API protocol specification provides capabilities to create and handle these Media Elements. The following table shows a description at a glance of the Media Elements provided by Media API.

Type	Capability	Name	Icon	Description
Endpoint	Protocols and Codecs	HttpGetEndpoint		This type of Endpoint provide unidirectional communications. Its MediaSink are associated with the HTTP GET method. It contains source MediaPad for audio and video, delivering media using HTML5 pseudo-streaming mechanism.
		HttpPostEndpoint		This type of Endpoint provide unidirectional communications. Its MediaSource are related to HTTP POST method. It contains sink MediaPad for audio and video, which provide access to an HTTP file upload function.
		RtpEndpoint		Endpoint that provides bidirectional content delivery capabilities with remote networked peers through RTP protocol. It contains paired sink and source MediaPad for audio and video.
		WebRtcEndpoint		This Endpoint offers media streaming using WebRTC.
	Media Repository	PlayerEndpoint		It provides function to retrieve contents from seekable sources in reliable mode (does not discard media information) and inject them into KMS. It contains one MediaSourcefor each media type

				detected.
		RecorderEndpoint		Provides function to store contents in reliable mode (doesn't discard data). It contains MediaSink pads for audio and video.
Filter	Computer Vision	FaceOverlayFilter		It detects faces in a video feed. The face is then overlaid with an image.
		PointerDetectorFilter		It detects pointers in a video feed. The detection of this Filter is based on color tracking in a video feed.
		PointerDetectorAdvFilter		It detects pointers in a video feed. The detection of this Filter is based on color tracking for round shapes (e.g. a ball) in a video feed.
		ZBarFilter		This Filter detects QR and bar codes in a video feed. When a code is found, the filter raises a CodeFound .
		PlateDetectorFilter		This Filter detects vehicle plates in a video feed.
	Augmented Reality	ChromaFilter		This type of Filter makes transparent a color range in the top layer, revealing another image behind.
		JackVaderFilter		Filter that detects faces in a video feed. Those on the right half of the feed are overlaid with a pirate hat, and those on the left half are covered by a Darth Vader helmet. This is an example filter, intended

				to demonstrate how to integrate computer vision capabilities into the multimedia infrastructure.
Hub	Media mix and distribution	Composite		A Hub that mixes the audio stream of its connected sources and constructs a grid with the video streams of its connected sources into its sink.
		Dispatcher		A Hub that allows routing between arbitrary port pairs.
		DispatcherOneToMany		A Hub that sends a given source to all the connected sinks.

(1)Media Element descriptions

Each Media Element accessible through the Media API has its specific capabilities, which are accessible through the JSON-RPC methods shown above. In this section, we show the specific operations and parameters that each Media Element accepts. Introducing this information directly onto the protocol specification will significantly decrease readability. On the sake of simplicity, we are going to present Media Element descriptions directly as a JavaScript API of Media Elements consuming the JSON-RPC Media API.

For example, the following JavaScript code that creates a media element:

```
var pipeline = ...;

PlayerEndpoint.create(pipeline, {uri:
"http://files.kurento.org/video/small.webm"},

function(error, player)

{

...
}
```

```
});
```

Is translated to the following create request:

```
{
  "jsonrpc": "2.0",
  "method": "create",
  "params": {
    "type": "PlayerEndPoint",
    "creationParams": {
      "pipeline": "6829986",
      "uri": "http://files.kurento.org/video/small.webm"
    },
    "sessionId": "c93e5bf0-4fd0-4888-9411-765ff5d89b93"
  },
  "id": 1
}
```

In the same sense, the following method invocation:

```
var httpGet = ...;
httpGet.getUrl(function(error, url)
{
  ...
});
```

Is translated to the following invoke request:

```
{
```

```

    "jsonrpc": "2.0",
    "method": "invoke",
    "params": {
      "object": "442352747",
      "operation": "getUrl",
      "sessionId": "c93e5bf0-4fd0-4888-9411-765ff5d89b93"
    },
    "id": 2
  }

```

Mapping from the JavaScript API to the JSON-RPC API is immediate:

- From the JSON-RPC message perspective, *create* methods are converted to *create* messages specifying in the *type* field the type of object to be built.
- The parameters of the create methods are converted to *creationParams* fields on the message.
- Method invocations are just *invoke* request specifying the method name in the *operation* field.
- The JavaScript API is object oriented. In this sense, if a *parent* element has a method, that method is valid for all its children.

Note: Parameters starting with "?" are optionals. *=*value** after a parameter indicates the default value.

PlayerEndpoint

Retrieves content from seekable sources in reliable mode (does not discard media information) and inject them into the Media Server. It contains one [MediaSource](#) for each media type detected.

Parent Class [UriEndpoint](#)

- [MediaPipeline](#) mediaPipeline: The [MediaPipeline](#) this PlayerEndpoint belongs to.
- Constructor Parameters
- String uri: URI that will be played
 - boolean ?useEncodedMedia=False: use encoded instead of raw media. If the parameter is false then the element uses raw media. Changing this parameter can affect stability severely, as lost key frames lost will not be regenerated.

Changing the media type does not affect to the result except in the performance (just in the case where original media and target media are the same) and in the problem with the key frames. We strongly recommended not to use this parameter because correct behaviour is not guaranteed.	
Declared Methods	<ul style="list-style-type: none"> • <code>void play()</code>: Starts to send data to the endpoint MediaSource
Declared Events	<ul style="list-style-type: none"> • EndOfStream

MediaObject	
Base for all objects that can be created in the media server.	
Declared Methods	<ul style="list-style-type: none"> • <code>MediaPipeline getMediaPipeline()</code>: Returns the pipeline to which this MediaObject belong, or the pipeline itself if invoked over a MediaPipeline. returns MediaPipeline: the MediaPipeline this MediaObject belongs to. If called on a MediaPipeline it will return this.
	<ul style="list-style-type: none"> • <code>MediaObject getParent()</code>: Returns the parent of this media object. The type of the parent depends on the type of the element that this method is called upon: The parent of a MediaPad is its MediaElement; the parent of a MediaMixer or a MediaElement is its MediaPipeline. A MediaPipeline has no parent, i.e. the method returns null. returns MediaObject: the parent of this MediaObject or null if called on a MediaPipeline.
Declared Events	<ul style="list-style-type: none"> • Error

HttpGetEndpoint	
An <code>HttpGetEndpoint</code> contains SOURCE pads for AUDIO and VIDEO, delivering media using HTML5 pseudo-streaming mechanism. This type of endpoint provide unidirectional communications. Its MediaSink is associated with the HTTP GET method	
Parent Class HttpEndpoint	
Constructor Parameters	<ul style="list-style-type: none"> • MediaPipeline <code>mediaPipeline</code>: the MediaPipeline to which the endpoint belongs
	<ul style="list-style-type: none"> • <code>boolean ?terminateOnEOS=False</code>: raise a MediaSessionTerminated

event when the associated player raises a [EndOfStream](#), and thus terminate the media session

- `MediaProfileSpecType ?mediaProfile=WEBM:` the [MediaProfileSpecType](#) (WEBM, MP4...) for the endpoint
- `int ?disconnectionTimeout=2:` disconnection timeout in seconds. This is the time that an http endpoint will wait for a reconnection, in case an HTTP connection is lost.

Declared
Events

WebRtcEndpoint

WebRtcEndpoint interface. This type of `Endpoint` offers media streaming using WebRTC.

Parent Class [SdpEndpoint](#)

Constructor Parameters

- [MediaPipeline](#) `mediaPipeline`: the [MediaPipeline](#) to which the endpoint belongs

Declared Events

SessionEndpoint

Session based endpoint. A session is considered to be started when the media exchange starts. On the other hand, sessions terminate when a timeout, defined by the developer, takes place after the connection is lost.

Parent Class [Endpoint](#)

Declared Events

- [MediaSessionTerminated](#)
- [MediaSessionStarted](#)

Hub

A Hub is a routing [MediaObject](#). It connects several [endpoints](#) together

Parent Class [MediaObject](#)

Declared Events

ZBarFilter

This filter detects [QR](#) codes in a video feed. When a code is found, the filter raises a [CodeFound](#) event.

Parent Class [Filter](#)

Constructor Parameters

- [MediaPipeline](#) mediaPipeline: the [MediaPipeline](#) to which the filter belongs

Declared Events

- [CodeFound](#)

Filter

Base interface for all filters. This is a certain type of [MediaElement](#), that processes media injected through its [MediaSink](#), and delivers the outcome through its [MediaSource](#).

Parent Class [MediaElement](#)

Declared Events

Endpoint

Base interface for all end points. An Endpoint is a [MediaElement](#) that allow the Media Server to interchange media contents with external systems, supporting different transport protocols and mechanisms, such as [RTP](#), [WebRTC](#), [HTTP](#), file:/ URLs... An Endpoint may contain both sources and sinks for different media types, to provide bidirectional communication.

Parent Class [MediaElement](#)

Declared Events

HubPort

This [MediaElement](#) specifies a connection with a [Hub](#)

Parent Class [MediaElement](#)

Constructor Parameters

- [Hub](#) hub: [Hub](#) to which this port belongs

Declared Events

PointerDetectorAdvFilter

This type of [Filter](#) detects UI pointers in a video feed.

Parent Class [Filter](#)

	<ul style="list-style-type: none"> • MediaPipeline mediaPipeline: the MediaPipeline to which the filter belongs
Constructor Parameters	<ul style="list-style-type: none"> • WindowParam calibrationRegion: region to calibrate the filter • PointerDetectorWindowMediaParam[] ?windows: list of detection windows for the filter. • void addWindow(window) Adds a new detection window for the filter to detect pointers entering or exiting the window PointerDetectorWindowMediaParam window: The window to be added • void clearWindows() Removes all pointer detector windows • void trackColorFromCalibrationRegion() This method allows to calibrate the tracking color. The new tracking color will be the color of the object in the colorCalibrationRegion. • void removeWindow(windowId) Removes a window from the list to be monitored String windowId: the id of the window to be removed
Declared Methods	
Declared Events	<ul style="list-style-type: none"> • WindowIn • WindowOut

UriEndpoint

Interface for endpoints the require a URI to work. An example of this, would be a [PlayerEndpoint](#) whose URI property could be used to locate a file to stream through its [MediaSource](#)

Parent Class [Endpoint](#)

Declared Methods	<ul style="list-style-type: none"> • String getUri() Returns the uri for this endpoint. returns String: the uri as a String
------------------	--

- `void pause()` Pauses the feed
- `void stop()` Stops the feed

Declared Events

HttpPostEndpoint

An [HttpPostEndpoint](#) contains SINK pads for AUDIO and VIDEO, which provide access to an HTTP file upload function. This type of endpoint provides unidirectional communications. Its [MediaSources](#) are accessed through the [HTTP](#) POST method.

Parent Class [HttpEndpoint](#)

- [MediaPipeline](#) mediaPipeline: the [MediaPipeline](#) to which the endpoint belongs
- `int ?disconnectionTimeout=2`: This is the time that an http endpoint will wait for a reconnection, in case an HTTP connection is lost.
- `boolean ?useEncodedMedia=False`: configures the endpoint to use encoded media instead of raw media. If the parameter is not set then the element uses raw media. Changing this parameter could affect in a severe way to stability because key frames lost will not be generated. Changing the media type does not affect to the result except in the performance (just in the case where original media and target media are the same) and in the problem with the key frames. We strongly recommended not to use this parameter because correct behaviour is not guaranteed.

Declared Events

- [EndOfStream](#)

RtpEndpoint

Endpoint that provides bidirectional content delivery capabilities with remote networked peers through RTP protocol. An [RtpEndpoint](#) contains paired sink and source [MediaPad](#) for audio and video.

Parent Class [SdpEndpoint](#)

- [MediaPipeline](#) mediaPipeline: the [MediaPipeline](#) to which the endpoint belongs

Declared Events

MediaPad

A [MediaPad](#) is an element's interface with the outside world. The data streams flow from the [MediaSource](#) pad to another element's [MediaSink](#) pad.

Parent
Class [MediaObject](#)

- Declared Methods
- `MediaElement getElement()` Obtains the [MediaElement](#) that encloses this pad. returns [MediaElement](#): the element
 - `MediaType getMediaType()` Obtains the type of media that this pad accepts. returns `MediaType`: One of [MediaType.AUDIO](#), [MediaType.DATA](#) or [MediaType.VIDEO](#)
 - `String getMediaDescription()` Obtains the description for this pad. This method does not make a request to the media server, and is included to keep the symmetry with the rest of methods from the API. returns `String`: The description

Declared
Events

PointerDetectorFilter

This type of [Filter](#) detects pointers in a video feed.

Parent Class [Filter](#)

- Constructor Parameters
- [MediaPipeline](#) `mediaPipeline`: the [MediaPipeline](#) to which the filter belongs
 - `PointerDetectorWindowMediaParam[] ?windows`: list of detection windows for the filter to detect pointers entering or exiting the window
- Declared Methods
- `void addWindow(window)` Adds a pointer detector window. When a pointer enters or exits this window, the filter will raise an event indicating so. `PointerDetectorWindowMediaParam window`: the detection window
 - `void clearWindows()` Removes all pointer detector windows
 - `void removeWindow(windowId)` Removes a pointer detector window

	String windowId: id of the window to be removed
Declared	• WindowIn
Events	• WindowOut

MediaSource	
Special type of pad, used by a media element to generate a media stream.	
Parent Class MediaPad	
Declared Methods	<ul style="list-style-type: none"> MediaSink[] <code>getConnectedSinks()</code> Gets all the MediaSinks to which this source is connected. returns MediaSink![]: the list of sinks that the source is connected to void <code>connect(sink)</code> Connects the current source with a MediaSink MediaSink sink: The sink to connect this source
Declared Events	

ChromaFilter	
ChromaFilter interface. This type of Filter makes transparent a colour range in the top layer, revealing another image behind	
Parent Class Filter	
Constructor Parameters	<ul style="list-style-type: none"> MediaPipeline mediaPipeline: the MediaPipeline to which the filter belongs WindowParam window: Window of replacement for the ChromaFilter String ?backgroundImage: url of image to be used to replace the detected background
Declared Methods	<ul style="list-style-type: none"> void <code>setBackground(uri)</code> Sets the image to show on the detected chroma surface. String uri: URI where the image is located void <code>unsetBackground()</code> Clears the image used to be shown behind the chroma surface.

Declared Events

MediaPipeline

A pipeline is a container for a collection of [MediaElements](#) and [MediaMixers](#). It offers the methods needed to control the creation and connection of elements inside a certain pipeline.

Parent Class [MediaObject](#)

Declared Events

MediaSink

Special type of pad, used by a [MediaElement](#) to receive a media stream.

Parent Class [MediaPad](#)

Declared
Methods

- `void disconnect(src)` Disconnects the current sink from the referred [MediaSource](#) [MediaSource](#) src: The source to disconnect
- `MediaSource getConnectedSrc()` Gets the [MediaSource](#) that is connected to this sink. returns [MediaSource](#): The source connected to this sink

Declared
Events**Dispatcher**

A [Hub](#) that allows routing between arbitrary port pairs

Parent Class [Hub](#)

Constructor
Parameters

- [MediaPipeline](#) mediaPipeline: the [MediaPipeline](#) to which the dispatcher belongs

Declared
Methods

- `void connect(source, sink)` Connects each corresponding [MediaType](#) of the given source port with the sink port. [HubPort](#) source: Source port to be connected [HubPort](#) sink: Sink port to be connected

Declared
Events

DispatcherOneToMany

A [Hub](#) that sends a given source to all the connected sinks

Parent Class [Hub](#)

Constructor Parameters

- [MediaPipeline](#) mediaPipeline: the [MediaPipeline](#) to which the dispatcher belongs

Declared Methods

- `void setSource(source)` Sets the source port that will be connected to the sinks of every [HubPort](#) of the dispatcher [HubPort](#) source: source to be broadcasted
- `void removeSource()` Remove the source port and stop the media pipeline.

Declared Events

Composite

A [Hub](#) that mixes the [MediaType.AUDIO](#) stream of its connected sources and constructs a grid with the [MediaType.VIDEO](#) streams of its connected sources into its sink

Parent Class [Hub](#)

Constructor Parameters

- [MediaPipeline](#) mediaPipeline: the [MediaPipeline](#) to which the dispatcher belongs

Declared Events

JackVaderFilter

Filter that detects faces in a video feed. Those on the right half of the feed are overlaid with a pirate hat, and those on the left half are covered by a Darth Vader helmet. This is an example filter, intended to demonstrate how to integrate computer vision capabilities into the multimedia infrastructure.

Parent Class [Filter](#)

Constructor Parameters

- [MediaPipeline](#) mediaPipeline: the [MediaPipeline](#) to which the filter belongs

Declared Events

HttpEndpoint

Endpoint that enables the Media Server to work as an HTTP server, allowing peer HTTP clients to access media.

Parent Class [SessionEndpoint](#)

Declared Methods

- `String getUrl()` Obtains the URL associated to this endpoint. returns String: The url as a String

Declared Events

SdpEndpoint

Implements an SDP negotiation endpoint able to generate and process offers/responses and that configures resources according to negotiated Session Description

Parent Class [SessionEndpoint](#)

Declared Methods

- `String generateOffer()` Request a SessionSpec offer. This can be used to initiate a connection. returns String: The SDP offer.
- `String processOffer(offer)` Request the NetworkConnection to process the given SessionSpec offer (from the remote User Agent) String offer: SessionSpec offer from the remote User Agent. returns String: The chosen configuration from the ones stated in the SDP offer
- `String processAnswer(answer)` Request the NetworkConnection to process the given SessionSpec answer (from the remote User Agent). String answer: SessionSpec answer from the remote User Agent. returns String: Updated SDP offer, based on the answer received.
- `String getLocalSessionDescriptor()` This method gives access to the SessionSpec offered by this NetworkConnection. Note: This method returns the local MediaSpec, negotiated or not. If no offer has been generated yet, it returns null. If an offer has been generated it returns the offer and if an answer has been processed it returns the negotiated local SessionSpec. returns String: The last agreed SessionSpec
- `String getRemoteSessionDescriptor()` This method gives access to the remote session description. Note: This method returns the media previously agreed after a complete offer- answer exchange. If no media has been agreed yet, it returns

	null. returns String: The last agreed User Agent session description
Declared Events	

FaceOverlayFilter	
FaceOverlayFilter interface. This type of Filter detects faces in a video feed. The face is then overlaid with an image.	
Parent Class	Filter
Constructor	
Parameters	<ul style="list-style-type: none"> • MediaPipeline mediaPipeline: pipeline to which this Filter belongs
Declared Methods	<ul style="list-style-type: none"> • void unsetOverlaidImage() Clear the image to be shown over each detected face. Stops overlying of the faces. • void setOverlaidImage(uri, offsetXPercent, offsetYPercent, widthPercent, heightPercent) Sets the image to use as overlay on the detected faces. String uri: URI where the image is located float offsetXPercent: the offset applied to the image, from the X coordinate of the detected face upper right corner. A positive value indicates right displacement, while a negative value moves the overlaid image to the left. This offset is specified as a percentage of the face width. For example, to cover the detected face with the overlaid image, the parameter has to be 0.0. Values of 1.0 or -1.0 indicate that the image upper right corner will be at the face's X coord, +- the face's width. Note: The parameter name is misleading, the value is not a percent but a ratio float offsetYPercent: the offset applied to the image, from the Y coordinate of the detected face upper right corner. A positive value indicates up displacement, while a negative value moves the overlaid image down. This offset is specified as a percentage of the face width. For example, to cover the detected face with the overlaid image, the parameter has to be 0.0. Values of 1.0 or -1.0 indicate that the image upper right corner will be at the face's Y coord, +- the face's width. Note: The parameter name is misleading, the value is not a percent but a ratio float widthPercent: proportional width of the overlaid image, relative to the width of the detected face. A value of 1.0 implies that the overlaid image will have the same width as the detected face. Values greater than 1.0 are allowed, while negative values are forbidden. Note: The parameter name is misleading, the value is not a percent but a ratio float heightPercent: proportional height of the overlaid image, relative to the

Declared Events	height of the detected face. A value of 1.0 implies that the overlaid image will have the same height as the detected face. Values greater than 1.0 are allowed, while negative values are forbidden. Note: The parameter name is misleading, the value is not a percent but a ratio
--------------------	--

PlateDetectorFilter	
PlateDetectorFilter interface. This type of Endpoint detects vehicle plates in a video feed.	
Parent Class	Filter
Constructor Parameters	<ul style="list-style-type: none"> • MediaPipeline mediaPipeline: the parent MediaPipeline of this PlateDetectorFilter
Declared Events	<ul style="list-style-type: none"> • PlateDetected

RecorderEndpoint	
Provides function to store contents in reliable mode (doesn't discard data). It contains MediaSink pads for audio and video.	
Parent Class	UriEndpoint
Constructor Parameters	<ul style="list-style-type: none"> • MediaPipeline mediaPipeline: the MediaPipeline to which the endpoint belongs • String uri: URI where the recording will be stored • MediaProfileSpecType ?mediaProfile=WEBM: Choose either a WEBM or a MP4 profile for recording • boolean ?stopOnEndOfStream=False: Forces the recorder endpoint to finish processing data when an EOS is detected in the stream
Declared Methods	<ul style="list-style-type: none"> • void record() Starts storing media received through the MediaSink pad
Declared Events	

MediaElement

Basic building blocks of the media server, that can be interconnected through the API. A [MediaElement](#) is a module that encapsulates a specific media capability. They can be connected to create media pipelines where those capabilities are applied, in sequence, to the stream going through the pipeline. [MediaElement](#) objects are classified by its supported media type (audio, video, etc.) and the flow direction: [MediaSource](#) pads are intended for media delivery while [MediaSinks](#) behave as reception points.

Parent
Class

[MediaObject](#)

Declared
Methods

- `MediaSource[] getMediaSrcs()` Get the [sources](#) of this element. returns [MediaSource!\[\]](#): A list of sources. The list will be empty if no sources are found.
- `MediaSource[] getMediaSrcs(mediaType, description)` Get the media sources of the given type and description `MediaType mediaType`: One of [MediaType.AUDIO](#), [MediaType.VIDEO](#) or [MediaType.DATA](#) String `description`: A textual description of the media source. Currently not used, aimed mainly for [MediaType.DATA](#) sources. returns [MediaSource!\[\]](#): A list of sources. The list will be empty if no sources are found.
- `MediaSource[] getMediaSrcs(mediaType)` get media sources of the given type. `MediaType mediaType`: One of [MediaType.AUDIO](#), [MediaType.VIDEO](#) or [MediaType.DATA](#). returns [MediaSource!\[\]](#): A list of sources. The list will be empty if no sources are found.
- `MediaSink[] getMediaSinks()` Get the [sinks](#) of this element. returns [MediaSink!\[\]](#): A list of sinks. The list will be empty if no sinks are found.
- `MediaSink[] getMediaSinks(mediaType)` A list of sinks of the given [MediaType](#). The list will be empty if no sinks are found. `MediaType mediaType`: One of [MediaType.AUDIO](#), [MediaType.VIDEO](#) or [MediaType.DATA](#). returns [MediaSink!\[\]](#): A list of sinks. The list will be empty if no sinks are found.
- `MediaSink[] getMediaSinks(mediaType, description)` A list of sinks of the given [MediaType](#). The list will be empty if no sinks are found. `MediaType mediaType`: One of [MediaType.AUDIO](#), [MediaType.VIDEO](#) or [MediaType.DATA](#). String `description`: A textual description of the media source. Currently not used, aimed mainly for [MediaType.DATA](#) sources. returns [MediaSink!\[\]](#): A list of sinks. The list will be empty if no sinks are found.
- `void connect(sink, mediaType, mediaDescription)` perform

[connect\(sink,mediaType\)](#) if there is exactly one sink for the given type, and their mediaDescriptions are the same [MediaElement](#) sink: the target [MediaElement](#) from which [MediaSink](#) will be obtained MediaType: the [MediaType](#) of the pads that will be connected String mediaDescription: A textual description of the media source. Currently not used, aimed mainly for [MediaType.DATA](#) sources

- `void connect(sink, mediaType)` Connects every [MediaSource](#) of this element belonging to the specified [MediaType](#) to the corresponding [MediaSink](#) of the target [MediaElement](#). This method will throw an exception if any of the following occur: .. * The number of sources for the specified [MediaType](#) in this element is different than the number of sinks on the target element. * There are duplicate mediaDescriptions on this' element sources for the specified [MediaType](#). * There are duplicate mediaDescriptions on target's element sinks for the specified [MediaType](#). * Target sinks' media descriptions are different from this sources' media descriptions for the specified [MediaType](#) This method is not transactional. In case of exception some of this element sources may be connected with target sinks. [MediaElement](#) sink: the target [MediaElement](#) from which [MediaSink](#) will be obtained MediaType: the [MediaType](#) of the pads that will be connected
- `void connect(sink)` perform [connect\(sink,mediaType\)](#) for every available [MediaType](#) in this source [MediaElement](#) sink: the target [MediaElement](#) from which [MediaSink](#) will be obtained

Declared
Events

GStreamerFilter

This is a generic filter interface, that creates GStreamer filters in the media server.

Parent Class [Filter](#)

Constructor

Parameters

- [MediaPipeline](#) mediaPipeline: the [MediaPipeline](#) to which the filter belongs
- String command: command that would be used to instantiate the filter, as in [\[/gst-launch-1.0|gst-launch\]](#)

Declared Events

CrowdDetectorFilter

Filter that detects people agglomeration in video streams

Parent Class [Filter](#)

Constructor	<ul style="list-style-type: none"> • MediaPipeline mediaPipeline: the MediaPipeline to which the filter belongs
Parameters	<ul style="list-style-type: none"> • RegionOfInterest[] rois: Regions of interest for the filter • CrowdDetectorFluidity
Declared Events	<ul style="list-style-type: none"> • CrowdDetectorOccupancy • CrowdDetectorDirection

(2)Record Types

PointerDetectorWindowMediaParam

Data structure for UI Pointer detection in video streams. All the coordinates are in pixels. X is horizontal, Y is vertical, running from the top of the window. Thus, 0,0 corresponds to the top left corner.

	<ul style="list-style-type: none"> • String id: id of the window for pointer detection • int height: height in pixels • int width: width in pixels • int upperRightX: X coordinate in pixels of the upper left corner
Properties	<ul style="list-style-type: none"> • int upperRightY: Y coordinate in pixels of the upper left corner • String activeImage: uri of the image to be used when the pointer is inside the window • float imageTransparency: transparency ratio of the image • String image: uri of the image to be used for the window. If activeImage has been set, it will only be shown when the pointer is outside of the window.

WindowParam

Parameter representing a window in a video stream. It is used in command and constructors for media elements. All units are in pixels, X runs from left to right, Y from top to bottom.

Properties	<ul style="list-style-type: none"> • int topRightCornerX: X coordinate of the left upper point of the window
------------	---

- `int topRightCornerY`: Y coordinate of the left upper point of the window
- `int width`: width in pixels of the window
- `int height`: height in pixels of the window

RegionOfInterestConfig

Data structure for configuration of CrowdDetector regions of interest.

- `int occupancyLevelMin`: minimum occupancy percentage in the ROI to send occupancy events
- `int occupancyLevelMed`: send occupancy level = 1 if the occupancy percentage is between `occupancy_level_min` and this level
- `int occupancyLevelMax`: send occupancy level = 2 if the occupancy percentage is between `occupancy_level_med` and this level, and send occupancy level = 3 if the occupancy percentage is between this level and 100
- `int occupancyNumFramesToEvent`: number of consecutive frames that a new occupancy level has to be detected to recognize it as a occupancy level change. A new occupancy event will be send
- `int fluidityLevelMin`: minimum fluidity percentage in the ROI to send fluidity events
- `int fluidityLevelMed`: send fluidity level = 1 if the fluidity percentage is between `fluidity_level_min` and this level
- `int fluidityLevelMax`: send fluidity level = 2 if the fluidity percentage is between `fluidity_level_med` and this level, and send fluidity level = 3 if the fluidity percentage is between this level and 100
- `int fluidityNumFramesToEvent`: number of consecutive frames that a new fluidity level has to be detected to recognize it as a fluidity level change. A new fluidity event will be send
- `boolean sendOpticalFlowEvent`: Enable/disable the movement direction detection into the ROI
- `int opticalFlowNumFramesToEvent`: number of consecutive frames that a new direction of movement has to be detected to recognize a new movement direction. A new direction event will be send

- `int opticalFlowNumFramesToReset`: number of consecutive frames in order to reset the counter of repeated directions
- `int opticalFlowAngleOffset`: Direction of the movement. The angle could have four different values: left (0), up (90), right (180) and down (270). This cartesian axis could be rotated adding an angle offset

RegionOfInterest

Region of interest for some events in a video processing filter

- Properties
- `Point[] points`: list of points delimiting the region of interest
 - `RegionOfInterestConfig regionOfInterestConfig`: data structure for configuration of CrowdDetector regions of interest
 - `String id`: identifier of the region of interest

Point

Point in a physical screen, coordinates are in pixels with X left to right and Y top to down.

- Properties
- `int x`: X coordinate in pixels of a point in the screen
 - `int y`: Y coordinate in pixels of a point in the screen

(3)Enum Types

MediaProfileSpecType

Media Profile. Currently WEBM and MP4 are supported.

- Properties
- WEBM
 - MP4

MediaType

Type of media stream to be exchanged. Can take the values AUDIO, DATA or VIDEO.

- Properties
- AUDIO
 - DATA
 - VIDEO

*(4)Events***PlateDetected**

Event raised by a [PlateDetectorFilter](#) when a plate is found in the data streamed.

Parent Class [Media](#)

Properties

- `String plate` : Plate identification that was detected by the filter

EndOfStream

Event raised when the stream that the element sends out is finished. An element receiving this event will generally just process any buffered data, and then forward the event further downstream.

Parent Class [Media](#)

Properties

CodeFound

Event raised by a [ZBarFilter](#) when a code is found in the data being streamed.

Parent Class [Media](#)

Properties

- `String codeType` : type of [QR](#) code found
- `String value` : value contained in the [QR](#) code

Error

An error related to the MediaObject has occurred

Parent Class [Media](#)

Properties

- `MediaObject object` : [MediaObject](#) where the error originated
- `String description` : Textual description of the error
- `int errorCode` : Server side integer error code
- `String type` : Integer code as a String

MediaSessionTerminated

Event raised when a session is terminated. This event has no data.

Parent Class [Media](#)

Properties

WindowIn

Event generated when an object enters a window.

Parent Class [Media](#)

Properties

- `String windowId` : Opaque String indicating the id of the window entered

MediaSessionStarted

Event raised when a session starts. This event has no data.

Parent Class [Media](#)

Properties

WindowOut

Event generated when an object exits a window.

Parent Class [Media](#)

Properties

- `String windowId` : Opaque String indicating the id of the window entered

Media

Base for all events raised by elements in the Media Server.

Parent Class [Media](#)

Properties

- `MediaObject source` : Object that raised the event
- `String type` : Type of event that was raised

CrowdDetectorFluidity

Event raise when a level of fluidity is detected in a ROI

Parent Class [Media](#)

- `float fluidityPercentage` : Percentage of fluidity in the ROI
- Properties
- `int fluidityLevel` : Level of fluidity in the ROI
 - `String roiID` : Opaque String indicating the id of the involved ROI

CrowdDetectorOccupancy

Event raise when a level of occupancy is detected in a ROI

Parent Class [Media](#)

- `float occupancyPercentage` : Percentage of occupancy in the ROI
- Properties
- `int occupancyLevel` : Level of occupancy in the ROI
 - `String roiID` : Opaque String indicating the id of the involved ROI

CrowdDetectorDirection

Event raise when a movement direction is detected in a ROI

Parent
Class [Media](#)

- `float directionAngle` : Direction angle of the detected movement in the ROI
- Properties
- `String roiID` : Opaque String indicating the id of the involved ROI

18 FIWARE OpenSpecification Data SemanticContextExt

Name	FIWARE.OpenSpecification.Data.SemanticContextExt		
Chapter	Data/Context Management,		
Catalogue-Link Implementation	to <FLOD>	Owner	ORANGE, Fano Ramparany

18.1 Preface

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

18.2 Copyright

Orange (FT), Telecom Italia (TI)

18.3 Legal Notice

Please check the following [Legal Notice](#) to understand the rights to use these specifications.

18.4 Overview

This enabler extends the functionalities of a publish/subscribe context broker, by making it possible to query the Context Broker through a SPARQL interface. SPARQL is the standard query language for the semantic web.

In the current implementation, this semantic extension has been brought to the CAP Context Broker through interfacing with the CML/CQL context modeling and querying languages. However, its

architecture has been designed in such a way that it can extend other PubSub GE implementations. Future releases of the Semantic Extension will be able to interface with the FIWARE-NGI API.

The following subsections introduce the SPARQL syntax and other basic concepts involved in the enabler and introduce its design architecture.

18.5 Basic Concepts

SPARQL (SPARQL Protocol and RDF Query Language) is the query language and protocol for searching, modifying and creating RDF data. RDF (Resource Description Framework) is a modeling language developed and promoted by the W3C organisation to describe resources in the web (i.e. web site content, web services).

Using RDF and more generally semantic modeling languages ensures the interoperability between the providers and the consumers of such descriptions. In particular, this makes it possible for a machine to process automatically such description.

A comprehensive description and specification of the SPARQL language can be found here:

http://www.w3.org/2009/sparql/wiki/Main_Page

We simply give here a short introduction to the language, which should be enough for understanding the rest of this document.

SPARQL and SQL syntax have much in common, as you can see with the following simple query sample:

```
SELECT ?mob
WHERE
{ ?mob rdf:type ps:EmeiEntity }
```

Which aims at retrieving instances of the class "EmeiEntity".

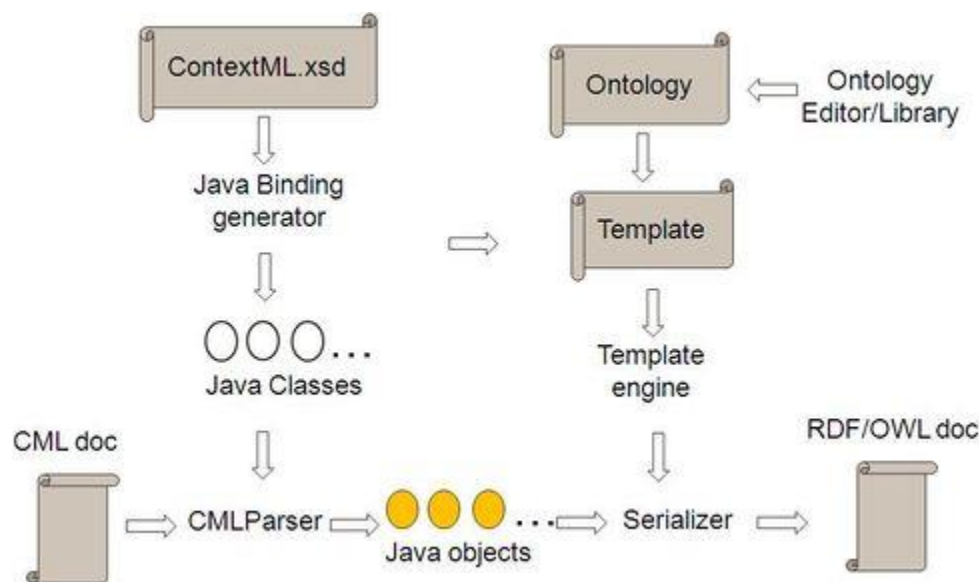
As suggested by this example, a SPARQL query has two parts:

- a "SELECT" block which specifies variables for which we seek a value. In this example there is only one variable "?mob". Variables have a name which always start with a "?" (like in the Prolog programming language)
- A "WHERE" block which specifies a graph fragment containing the variables that could be matched against the RDF model. In this example the graph fragment is a single arc where the variable "?mob" is the subject, "rdf:type" is the predicate and "ps:EmeiEntity" is the object.

A graph fragment is specified as a set of arcs (RDF triples) where the two nodes and the link between the nodes can be variables (such as "?mob" in our example), or fully instantiated (such as "rdf:type" or "ps:EmeiEntity").

18.6 Main Interactions

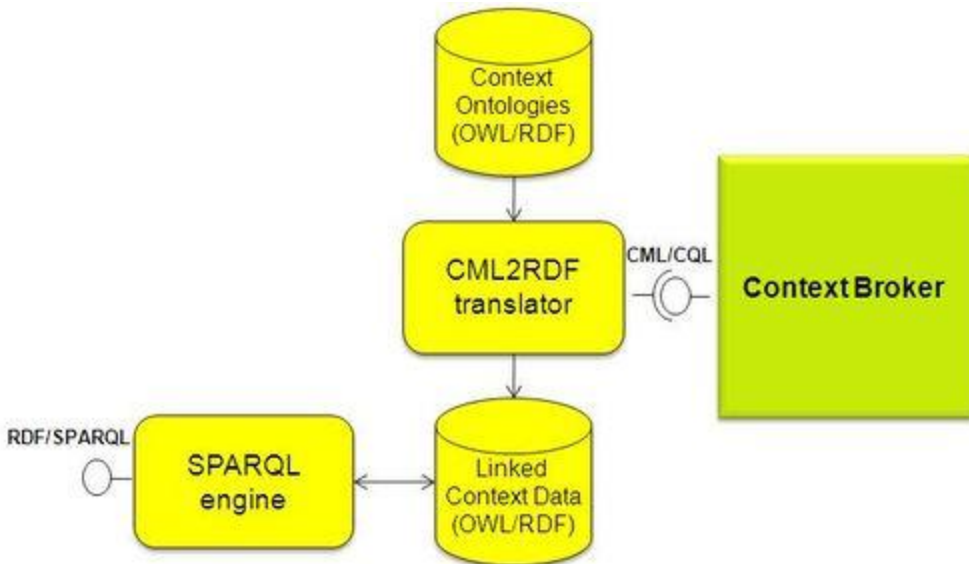
- The CML2RDFTranslator as been designed by composing a CML parser an object serializer and a template engine. The parser interprets a CML document as a structured object which is then serialized into a XML document which complies to the OWL/RDF XML encoding. This compliance is ensured by guiding the serialization along a template document which contains static OWL XML fragments and script programming instructions which traverse the structured object which the parser creates. Values returned by these instructions take the place of the instructions themselves in the template. The overall process is depicted in the following diagram:



- The OWL document produced by the CML2RDFTranslator process is stored in a RDF database.
- The SPARQL REST API makes it possible to query the RDF database.

18.7 Basic Design Principles

The Semantic Extension component interacts with the PubSub GE using the native CML/CQL protocol as depicted in the high level functional architecture diagram shown below.



Semantic Extension functional architecture

The component provides two main functions:

- The CML2RDFTranslator
- The SPARQL engine

The CML2RDFTranslator receives CML data from the Context Broker and convert it into a RDF document which complies to domain specific ontologies. For instance if the CML data contains geographical positioning information it will be converted into a RDF document that instantiates a Geospatial Ontology that standardizes the coordinate properties such as "longitude" and "latitude" and specifies the datatypes and units that the values of these properties should be expressed in and interpreted as. The resulting RDF document is stored temporarily in the RDF cache "Linked Context Data".

The "Context Ontologies" are organized in a modular way. For instance, they include "base ontologies" covering temporal, spatial, social and quality of information concepts, as well as "domain ontologies", such as smarthome, transportation ontologies or as in our examples the physical addressing ontology. For example, the concept "Entity" is defined in a domain ontology but is subsumed by the concept "Thing" which is defined in a base ontology. Likewise, timestamps are defined in the time ontology.

The PubSub semantic extension provides a REST API. It can be invoked by using the GET method with the unique and mandatory "query" parameter, which value should be a SPARQL query.

The [SPARQL API User Manual](#) provides more guidance on how to use this API.

18.8 Detailed Specifications

Following is a list of Open Specifications linked to this Generic Enabler. Specifications labeled as "DRAFT" are planned for future Major Releases of FI-WARE but they are provided for the sake of future users.

18.8.1 Open API Specifications

- [Publish/Subscribe Semantic Extension Open RESTful API Specification](#)

18.9 Re-utilised Technologies/Specifications

This GE relies on the W3C specifications SPARQL, RDF and OWL.

18.10 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 **avalue**. 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 be 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 two 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.

19 Publish/Subscribe Semantic Extension Open RESTful API Specification

19.1 Introduction to the Context Broker (CB) Semantic Extension GE REST API

19.1.1 The Semantic Extension API

This API enables applications to retrieve context information using the semantic web standard protocol SPARQL and modeling language RDF.

19.1.2 Intended Audience

This document is primarily intended for Future Internet application developers who want to interface their application with the context broker using semantic web protocols, mainly using SPARQL. To use this information, the reader should have a general understanding of the GE service [reference to the GE Open Specification](#) and be familiar to the following technologies:

- RESTful web services
- HTTP/1.1
- XML

19.1.3 API Change History

Revision Date	Changes Summary
March 05, 2014	Creation

19.1.4 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](#).

19.1.5 Additional Resources

19.2 API Operations

19.2.1 Context Semantic query

The CB semantic extension provides a REST API. It can be invoked by using the GET method with the unique and mandatory "query" parameter, which value should be a SPARQL query.

Verb	URI example	Description
GET	http://{SemanticExtensionIPAddress}:{SemExtPort}/queryservice/sparql?query={SPARQLQuery}	Get results of a {SPARQLQuery} request

Where:

```
{SemanticExtensionIPAddress}
```

is the IP address of the SemanticExtension service. In the testbed, the global instance of this service can be used for testing purposes. Its IP address is: 130.206.82.176.

```
{SemExtPort}
```

is the portnumber where the SemanticExtension service will listen to its clients. The default portnumber is 9000.

```
{SPARQLQuery}
```

is the SPARQL query.

19.2.2 SPARQL

is a protocol for querying RDF documents. It includes:

- a language for expressing queries
- a set of methods for submitting these queries
- interaction structures for organising the interaction between the server managing the RDF documents and the client querying about these documents.

CB Semantic Extension adopts the language part of the specification. For a detailed and complete description of SPARQL you could refer to: [\[1\]](#)

19.2.3 Ontologies

OWL is a language for specifying ontologies. An ontology is a formal description of the conceptualization of a application domain. Basically it enables to define classes (e.g. the class of "human") and instances (e.g. "John Smith"), properties (e.g. "age"), relations (e.g. "isFatherOf") between concepts and properties of relations (e.g. "bijective").

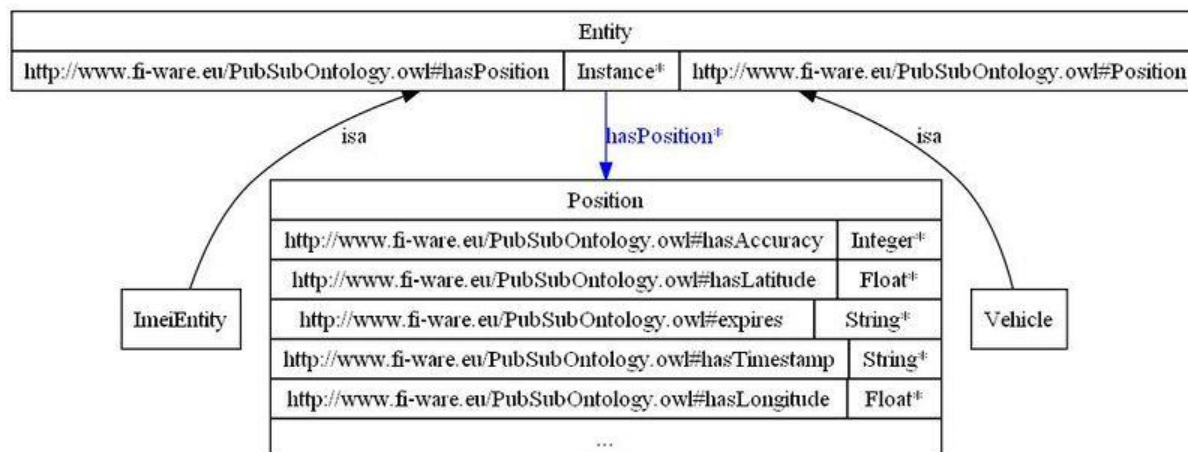
For a detailed and complete description of OWL you could refer to: [\[2\]](#)

CB Semantic Extension adopt the OWL language. Its architecture is designed in such a way that, although few domain ontologies (such as geographical location and postal address ontologies) are already supported, it is possible to integrate new domain ontologies (see CB Semantic Extension programming guide document). The best practice recommendation here is to search for an existing ontology, preferably one coming from a large base community (e.g. the geographical location ontology that CB Semantic Extension support is the one published by the OGC), or a defacto standard, instead of creating one from scratch.

In the following section we describe the domain ontologies that CB Semantic Extension currently supports:

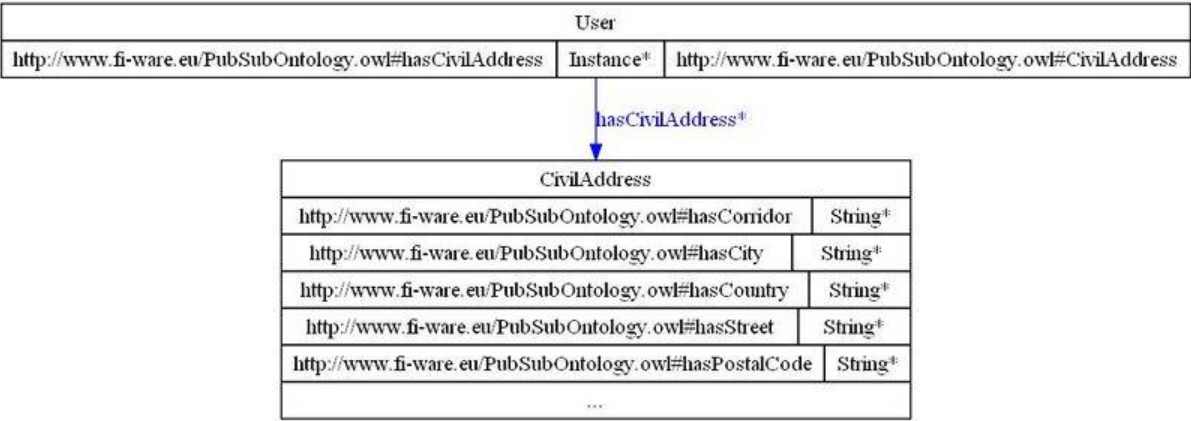
19.2.3.1 Geographical Location Ontology

Source: W3C Geospatial Vocabulary [\[3\]](#)



19.2.3.2 Postal Addressing Ontology

Source: C-CAST IST EU project [\[4\]](#)



20 FIWARE OpenSpecification Data UnstructuredDataAnalysis

Name	FIWARE.OpenSpecification.Data.UnstructuredDataAnalysis		
Chapter	Data/Context Management,		
Catalogue-Link Implementation	to <UDA>	Owner	Atos Origin, Jose Maria Fuentes Lopez

20.1 Preface

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

20.1.1 Copyright

- Copyright © 2013 by [Atos Origin](#)

20.1.2 Legal Notice

Please check the following [Legal Notice](#) to understand the rights to use these specifications.

20.1.3 Overview

The Unstructured Data Analysis is the generic enabler focusing on gathering and analysing unstructured data. The target user as well as a detailed description are provided in the following.

20.1.3.1 **Target usage**

Target users are mainly data analysis and data visualization developers that need to analyze unstructured web resources.

20.1.3.2 **Unstructured Data Analysis GE Description**

The Unstructured Data Analysis is a composition of technologies in order to gather and analyse unstructured data. In a specific way, this generic enable is responsible for acquiring unstructured data from several data sources, preparing it for the analysis, and executed the linguistic and statistical analysis required by the tendencies detection). This GE is running continuously, polling the Web (through RSS feeds) for recent content, turning it into a stream of processed text documents.

The documents gathered by this GE are provided in a semi-structured fashion (following the RSS structure) such as titles, publication dates, and other metadata are clearly indicated. Furthermore, the Web pages related with the RSS feeds are also gathered. Finally, the trending topics are identified in texts and article bodies using linguistic and statistical analysis. The GE provides a REST API that allows define the data sources and get the results of the analysis.

The web content contains a “noise” that needs to be identified and removed before the content can be analysed. For this reason, a pipeline is executed which consists of (i) unstructured data acquisition (ii) data cleaning, (iii) data storage (iii) natural-language processing analysis, (iv) statistical analysis and (v) Results storage.

This GE has been developed based on the experiences obtained in the FIRST [FIRST] and ALERT [ALERT] projects.

20.1.3.3 **Example Scenario**

Unstructured data is defined as all the set of data that do not follow a predefined data model, these data comes in different formats (text, document, image, and video). The amount of unstructured data is by far greater than structured data. According to a 2011 [IDC study](#)[IDC2011], the 90 percent of all data created in the next decade will be unstructured data. This scenario aims the creation of novel tools to handle and analyze these data.

The GE provides a method to analyze text-based unstructured data. Analyzing web data (RSS and web pages) in order to detected emerging tendencies key words. A tendency is defined as the meaningful n-grams (a subsequence of n words) which have becoming relevant in a community during a specific time period. This functionality is useful for market surveillance and others related.

For use these functionalities, the user (an expert in media analysis) will define the sources to be explored (RSS feeds), creating a project and adding the data sources to it using a REST API provided by the GE. Once the user has added a new source, the GE will start to monitor the feeds obtaining the new entries, getting the web pages related to those entries, preprocess the data, store it and start the analysis. The results of the analysis and the retrieve data can be accessed also through the API.

The ALERT project provides an example scenario of this. This project aims to develop tools for the analysis of open sources (FLOSS) communities, which are composed of developers, users and community manager and others. The results of these analysis are valuable information to community managers in the decision making process. One of the results of this project was the [OCELOT](#) component, a tool for the analysis of unstructured data contained in FLOSS communities (forums, mailing list, bug tracker system, source code management) in order to detect emerging concepts (key terms) to be used to extend ontological resources used in other analysis (see video demo [\[1\]](#)). This component was improved the FI-WARE project and it is included inside the Unstructured Data Analysis GE.

20.1.4 Basic Concepts

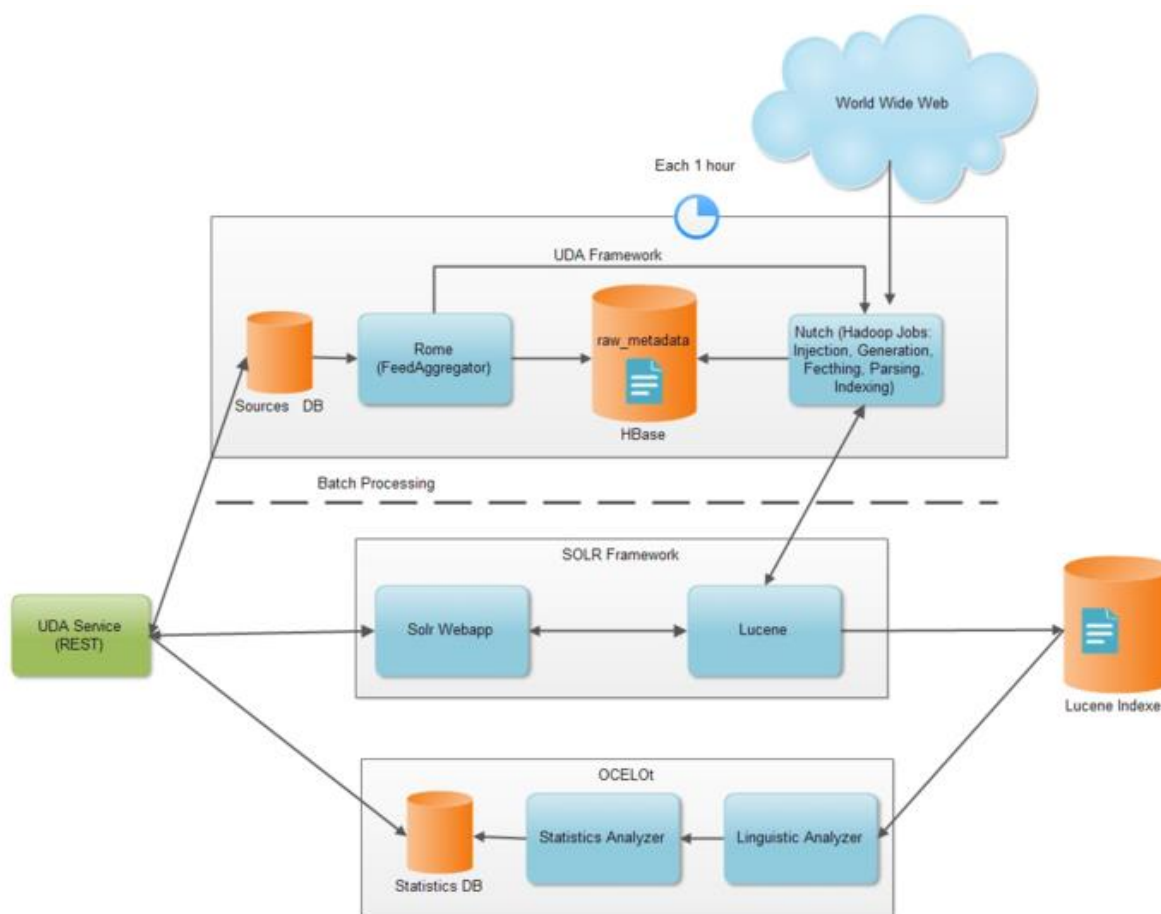
This section introduces the basic concepts related to the Unstructured Data Analysis GE in order to facilitate the understanding of this description.

1. **Really Simple Syndication (RSS)**: RSS is a XML-based format used to publish new contents in order to provide a standard way to publish and consume those contents. A RSS feed is a XML file published through a HTTP server and using RSS format. This feed is composed by entries each represents new content. An entry could be associated a web page.
2. **Web Crawling**: Crawling is the process to automatically detect and retrieve new of web resources in order to be gathered and processed for a specific purpose. The software used for the crawling process is known as web bots.
3. **Emerging term (Tendency)**: An emerging term is a meaningful n-gram (a subsequence of words) which has been gained relevance in a community; the emerging terms are a key element in the detection of new tendencies.
4. **Natural Language Processing (NLP)**: NLP is the convergence of linguistic, computer sciences and statistic to provide strategies to extract knowledge from data expressed in natural language (written or spoken). This GE uses strategies for the tokenization, lemmatization and Part-of-Speech Tagging of the gathered data.

20.1.5 Unstructured Data Analysis GE Architecture

The objective of the Unstructured Data Analysis GE (also known as UDA GE) is to facilitate the monitoring process of web sources in order to detect potential tendencies providing tool for gathering, preprocess, store and analyze the unstructured data contained in those sources

In order to satisfy the previous objective, the UDA GE is composed by a set of components. Next figure presents the UDA GE Infrastructure architecture.



UDA architecture

The Unstructured Data Analysis GE is composed by a set of modules, each one play a specific role in order to provide the functionalities of the GE.

The UDA REST API is the interface provided by the GE to interact with any user or Software component that requires the functionalities of this GE. This API provides the operations that allows creating new unstructured data analysis projects, adding sources (RSS feeds) to be analyzed. Also, this API allows obtaining the results of the analysis and the gathered documents.

Once the sources are included to a project, the GE begins the crawling process. This task is performed by the UDA Framework, which get the RSS entries from the feeds included in the project in order to process its content and also to retrieve the web pages contained in the feeds. The UDA Framework uses a customized version of [Apache Nutch](#) to execute this operation. All the retrieve content is cleaned in order to extract the plain text from the RSS/HTML raw data. The data (raw and cleaned data) is stored in an Hbase No-SQL database and also in a Lucene index. The Hbase data is used by the tendencies detection analysis and also it is available for other analysis. On the other hand, the data stored in the Lucene index is used to facilitate the retrieval of the documents gathered by the GE via the REST API.

Finally, the OCELOT (Online Semantic Concept Extractor based on Linked Open Data) component analyze unstructured data in order to detect emerging tendencies, these tendencies could be used to get

awareness about relevant challenges in a specific domain. OCELOt uses different natural language processing strategies (tokenization, lemmatization, Part-of-Speech Tagging) and statistical analysis to detect the tendencies. The results of the analysis are stored in a relational database, which are given to the user through the REST API.

20.1.6 Main Interactions

20.1.6.1 *Modules and Interfaces*

This section reports on the description of the Unstructured Data Analysis GE main functionality. The description of this functionality is based on the functionality provided by the baseline assets. Section Backend functionality describes functionality (methods) provided to agents in a service like style.

20.1.6.2 *Backend Functionality*

Backend functionality describes functionality provided by the GE 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, which provides the next operations:

1. **Create project:** Creates an unstructured data analysis project, a software abstraction that represent the monitoring and analysis of a set of data sources in a specific domain. To invoke the operation, a POST http request should be sent to [http://<ge url location>/uda-service/uda/\[PROJECT_NAME\]](http://<ge url location>/uda-service/uda/[PROJECT_NAME])
2. **Get project configuration:** Obtains the configuration (name, description, analyzed sources) of a specific project. To invoke the operation, a GET http request should be sent to [http://<ge url location>/uda-service/uda/\[PROJECT_NAME\]](http://<ge url location>/uda-service/uda/[PROJECT_NAME])
3. **Delete project:** Remove a project; this will stop the analysis of the sources contained in the project. To invoke the operation, a DELETE http request should be sent to [http://<ge url location>/uda-service/uda/\[PROJECT_NAME\]](http://<ge url location>/uda-service/uda/[PROJECT_NAME])
4. **Get project's data sources configuration:** Obtains all the sources analyzed in a specific project. To invoke the operation, a GET http request should be sent to [http://<ge url location>/uda-service/uda/\[PROJECT_NAME\]/sources](http://<ge url location>/uda-service/uda/[PROJECT_NAME]/sources)
5. **Add data source to a project:** Add a new source to be analyzed in a specific project. To invoke the operation, a POST http request should be sent to [http://<ge url location>/uda-service/uda/\[PROJECT_NAME\]/sources/\[SOURCE_NAME\]](http://<ge url location>/uda-service/uda/[PROJECT_NAME]/sources/[SOURCE_NAME])
6. **Get data source configuration:** Obtain the configuration of a source analyzed in a specific project. To invoke the operation, a GET http request should be sent to [http://<ge url location>/uda-service/uda/\[PROJECT_NAME\]/sources/\[SOURCE_NAME\]](http://<ge url location>/uda-service/uda/[PROJECT_NAME]/sources/[SOURCE_NAME])
7. **Remove a data source from a project:** Remove a source from a project; this will stop the data gathering and analysis from that source. To invoke the operation, a DELETE http request should be sent to [http://<ge url location>/uda-service/uda/\[PROJECT_NAME\]/sources/\[SOURCE_NAME\]](http://<ge url location>/uda-service/uda/[PROJECT_NAME]/sources/[SOURCE_NAME])

8. **Search in a project:** Executes a textual search over the documents gathered in a specific project using the Lucene query format[LuceneQuery] . To invoke the operation, a GET http request should be sent to [http://<ge url location>/uda-service/uda/\[PROJECT_NAME\]/search](http://<ge url location>/uda-service/uda/[PROJECT_NAME]/search)

All methods described can be invoked by means of regular HTTP requests either using a web browser (for those ones who rely on GET requests) or by an APIs such as Jersey.

20.1.6.3 Frontend Functionality

This GE does not provide a user interface.

20.1.7 Design Principles

The GE design principle is to use on well-known tools and standards in all phases of the processing:

1. The data is gathered from RSS/HTML standards sources, which are widely recognized and used across the world to publish unstructured data.
2. A web crawler based on Apache Nutch tool (An widely used crawler) is used to the retrieve the data, and also executes a preprocessing, the data is stored the data into an HBase (well-known NoSql database).
3. The linguistic and statistical analysis uses Stanford NLP framework for the analysis of English texts.
4. The GE provides a API based on the REST de facto standard to interact with other components

20.1.8 References

[IDC2011]	J. Gantz, D. Reinsel. Extracting Value from Chaos. IDC IVIEW [2]
[StanfordNLP]	Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003.Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In Proceedings of HLT-NAACL 2003 pages 252-259.
[REST2002]	R. T. Fielding, R. N. Taylor. Principled Design of the Modern Web Architecture. ACM Transactions on Internet Technology, Vol. 2, No. 2, May 2002, Pages 115–150
[LuceneQuery]	Lucene Query format, [3]
[FIRST]	The FIRST Project, [4]
[ALERT]	The ALERT Project, [5]

20.2 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.

20.2.1 Open API Specifications

- [Unstructured Data Analysis Open RESTful API Specification](#)

20.3 Re-utilised Technologies/Specifications

The Unstructured Data Analysis Generic Enabler will be based on the outcomings of the FIRST (FP7-257928) project, a European Commission FP7 funded Specific Targeted Research Project started on October 1st 2010, specifically in the FIRST Analytical Pipeline. A high level description of this pipeline can be found in [FIRST Analytical Pipeline high level description](#).

20.4 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 be 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 two 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.

21 Unstructured Data Analysis Open RESTful API Specification

21.1 Introduction to the Unstructured Data Analysis API

Please check the [FI-WARE Open Specification Legal Notice \(implicit patents license\)](#) to understand the rights to use FI-WARE Open Specifications.

21.1.1 Unstructured Data Analysis API Core

The Unstructured Data Analysis API is a RESTful, resource-oriented API accessed via HTTP that uses JSON-based representations for information interchange. This API provides the means to effectively manage unstructured data analysis projects and can be used to integrate these functionalities into third party applications.

21.1.2 Intended Audience

This specification is intended for data analysis and visualization developers that need to analyze unstructured web resources. This document provides a full specification of how to interoperate with this GE. To use this information, the reader should firstly have a general understanding of the [\[1\]](#).

21.1.3 API Change History

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

Revision Date	Changes Summary
March 04, 2014	Initial API version

21.1.4 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 [\[2\]](#).

21.1.5 Additional Resources

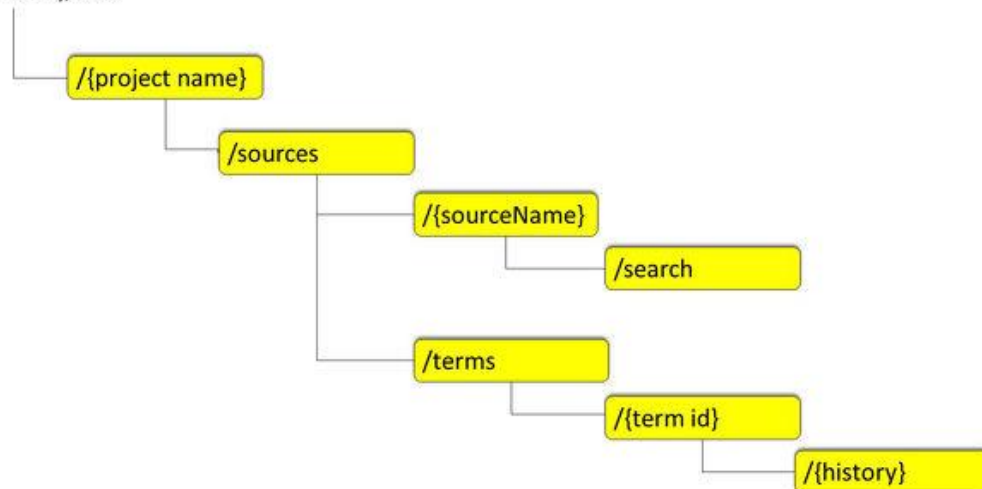
For more details about the Unstructured Data Analysis GE, please refer to [\[3\]](#) and also to the Architecture description mentioned before.

21.2 General Ontology Registry API Information

21.2.1 Resources Summary

Unstructured Data Analysis GE

//{serverRoot}/uda



21.2.2 Representation Format

The Unstructured Data Analysis API supports JSON based representation formats.

21.2.3 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.

21.2.4 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.

21.2.5 Links and References

No additional links or references are provided.

21.3 API Operations

The following section provides the detail for each RESTful operation giving the expected input and output for each URI.

21.3.1 Unstructured Data Analysis API Operations

21.3.1.1 *Create/Retrieve/Delete Project configuration*

Verb	URI	Description
POST	/project name}	Creates a new Unstructured Data Analysis project
GET	/project name}	Get the current configuration of a unstructured data analysis project (name, description, sources) encoded in a JSON format
DELETE	/project name}	Delete an unstructured data analysis project

Response codes:

- HTTP/1.1 200 - The operation has been executed
- HTTP/1.1 500 - If there are some unidentified error.

Request example (Create Project):

```
POST /Project HTTP/1.1
Host: localhost:8080
Content-Type: application/x-www-form-urlencoded
```

Response example:

```
HTTP/1.1 200 OK
{"message": "OK", "success": true}
```

Request example (Get Project):

```
GET /Project HTTP/1.1
```

```
Host: localhost:8080
```

Response example:

```
HTTP/1.1 200 OK

{"id":1,"description":"test
channel","name":"Test","sources":[{"id":"bbcnews","name":"bbcnews","ur
l":"http://feeds.bbc.co.uk/news/rss.xml"}]}
```

Request example (Delete Project):

```
GET /Project HTTP/1.1

Host: localhost:8080
```

Response example:

```
HTTP/1.1 200 OK

{"message":"OK","success":true}
```

21.3.1.2 Get Project Sources

Verb	URI	Description
GET	/Project name}/sources	Get the current RSS sources (name, URL, description) of the project encoded using JSON

Response codes:

- HTTP/1.1 200 - If the sources are successfully retrieved
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
GET /TestProject/sources HTTP/1.1

Host: localhost:8080
```

Response example:

```
HTTP/1.1 200 OK

[{"id":"1","name":"bbcnews","url":"http://feeds.bbc.co.uk/news/rss.xml"}]
```

21.3.1.3 Add/Get/Remove RSS sources in a project

Verb	URI	Description
POST	/Project {name}/sources/{Source name}	Add a new RSS source to an unstructured data analysis project.
GET	/Project {name}/sources/{Source name}	Get the source data (url, description).
DELETE	/Project {name}/sources/{Source name}	Remove the source from the project (no further data will be gathered from there)

Response codes:

- HTTP/1.1 200 - If the source is successfully processed by the GE
- HTTP/1.1 500 - If there are some unidentified error.

Request example (Add source):

```
POST /TestProject/sources/bbc HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: localhost:8080
description=BBC News&url=http://feeds.bbc.co.uk/news/rss.xml
```

Response example:

```
HTTP/1.1 200 OK
{"message":"OK","success":true}
```

Request example (Get source metadata):

```
GET /TestProject/sources/bbc HTTP/1.1
Host: localhost:8080
```

Response example:

```
HTTP/1.1 200 OK
{"id":"1","name":"bbcnews","url": "http://feeds.bbc.co.uk/news/rss.xml"}
```

Request example (Remove source):

```
DELETE /TestProject/sources/bbc HTTP/1.1
Host: localhost:8080
```

Response example:

```
HTTP/1.1 200 OK

{"message": "OK", "success": true}
```

21.3.1.4 Search data in a project

Verb	URI	Description
POST	/Project name)/search	Search a specific pattern (following the Lucene query format) into the gathered document of the project.

Response codes:

- HTTP/1.1 200 - If the search was executed successfully
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
POST /TestProject/search HTTP/1.1

Content-Type: application/x-www-form-urlencoded

Host: localhost:8080

query=olympics
```

Response example:

```
HTTP/1.1 200 OK

[
  {
    "id": "0a4f5aa106ae09db16d61c45abc2db47",
    "pubDate": "2014-02-27T13:33:32.000",
    "text": "BBC Sport - Sochi 2014: Russia Winter Olympics site Terms of Use About the BBC Privacy Accessibility.....",
    "title": "Wada to act over Russia 'gas' claims",
    "projectid": "1",
    "digest": "1ce7ae0582621568a75dd3bfe6256ae8",
    "url": "http://www.bbc.co.uk/sport/0/winter-olympics/26369004"
  }
]
```


21.3.1.5 *Get Trending Terms*

Verb	URI	Description
GET	<code>/[Project name]/terms</code>	Get the most relevant terms detected in the project between two dates

Response codes:

- HTTP/1.1 200 - If the terms was retrieved correctly
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
GET /TestProject/terms?startdate=2013-01-01&enddate=2013-01-01
Host: localhost:8080
```

Response example:

```
HTTP/1.1 200 OK
```

```
[{"term":{"id":"6d9c4e52f18f604187a77eac8944d909","token":"Computer","lemma":"Computer","posTag":"NN"},"date":"2013-01-01","occurrences":1521},
{"term":{"id":"af8939e16b69103e2a063b30ce293dc8","token":"Test","lemma":"Test","posTag":"NN"},"date":"2013-01-01","occurrences":985}]
```

21.3.1.6 *Get Term metadata*

Verb	URI	Description
GET	<code>/[Project name]/terms/{term id}</code>	Get the metadata of a term

Response codes:

- HTTP/1.1 200 - If the terms was retrieved correctly
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
GET /TestProject/terms/6d9c4e52f18f604187a77eac8944d909
```

Host: localhost:8080

Response example:

HTTP/1.1 200 OK

```
{ "id": "6d9c4e52f18f604187a77eac8944d909", "token": "Computer", "lemma": "Computer", "posTag": "NN" }
```

21.3.1.7 *Get Term history*

Verb	URI	Description
GET	<code>/[Project name]/terms/{term id}/history</code>	Get the occurrence history of a terms between two dates

Response codes:

- HTTP/1.1 200 - If the terms was retrieved correctly
- HTTP/1.1 500 - If there are some unidentified error.

Request example:

```
GET
/TestProject/terms/6d9c4e52f18f604187a77eac8944d909/history?stardate=2013-01-01&enddate=2013-01-01
```

Host: localhost:8080

Response example:

HTTP/1.1 200 OK

```
[{"date": "2013-01-01", "occurrences": 1521}]
```

22 FIWARE OpenSpecification Data Location

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

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

22.1.1 Copyright

- Copyright © 2012 by [Thales](#)

22.1.2 Legal Notice

Please check the following **FI-WARE Open Specification Legal Notice (essential patents license)** [\[1\]](#) to understand the rights to use these specifications.

22.1.3 Overview

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

- Third-party location clients

Third-party location clients can interact with the location platform using the **Mobile Location Protocol** (MLP, [\[2\]](#)) interface or **RESTful Network API for Terminal Location** ([\[3\]](#)) both standardized by Open Mobile Alliance (OMA, [\[4\]](#)).

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, [\[5\]](#)).

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

- 3G capable
- Wi-Fi optional
- Be equipped with an assisted GPS chipset
- Support Secure User Plane (SUPLv2) stack : No mobile in commercial market are implemented yet the v2 protocol version (only v1)

22.1.3.1 **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, Wi-Fi 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 Wi-Fi 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.

To cope with the lack of SUPLv2 commercial handsets, the location GE now includes a fleet simulation tool that simulates multiple mobile devices moving across routes or staying static. The fleet and each simulated device can be managed via a RESTful interface in order to fulfill demo requirements, as requested by UC projects.

Last but not least, the Location GE is compatible with an Android application being developed by TAS for FI-WARE (required Android version is version 2.3.3 minimum). Such application does not replace a real SUPLv2 stack but aims at demonstrating the features of the Location GE with a real handset. The architecture of such application is out of scope for this document but it is important to say that the architecture presented in the next sections addresses the interface requirements.

22.1.4 Basic Concepts

22.1.4.1 *Third-party location services*

Services provided for third-party location clients are standardized under the so-called "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 and 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.

Trigger service requires SUPL V2 interface on terminal.

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).

Periodic and Area (Circle) Notification Subscription requires SUPL V2 interface on terminal.

22.1.4.2 *Access control and privacy management*

Fortunately, not all applications can access to these location services. Strong access control and privacy management rules are applied to authorize a third-party to localize a particular end-user terminal. Each location request contains client credentials (login and password) and a service identifier. These values are used by the Location GE to ensure that the correct credentials are provided and that the requested service belongs to this client. Moreover, many service parameters (stored in Location GE database) are

used to accept location requests or not. For example, location requests are filtered based on service status (active or barred), type of request (single/tracking/emergency/all), level of accuracy (low/medium/high), etc. Lastly, end-user parameters (stored in Location GE database) are checked to ensure that the end-user consents to be localized by the requested service. For example, an end-user can authorize a specific service to localize him permanently, once or on selected time windows. The end-user can also override service parameters, such as level of accuracy to limit this service to Cell-ID positioning.

22.1.4.3 **Mobile end-user services**

Services provided for mobile end-users are standardized under the so-called "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 they 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.

Location request with transfer to third party and Periodic trigger requires SUPL V2 interface on terminal.

22.1.4.4 **Fleet Simulation**

The fleet simulation tool is used to demonstrate the Location GE features. It is not part of the Location GE core engine but rather stubs the SUPLv2 interface to simulate the behavior of a real handset moving across routes or staying static.

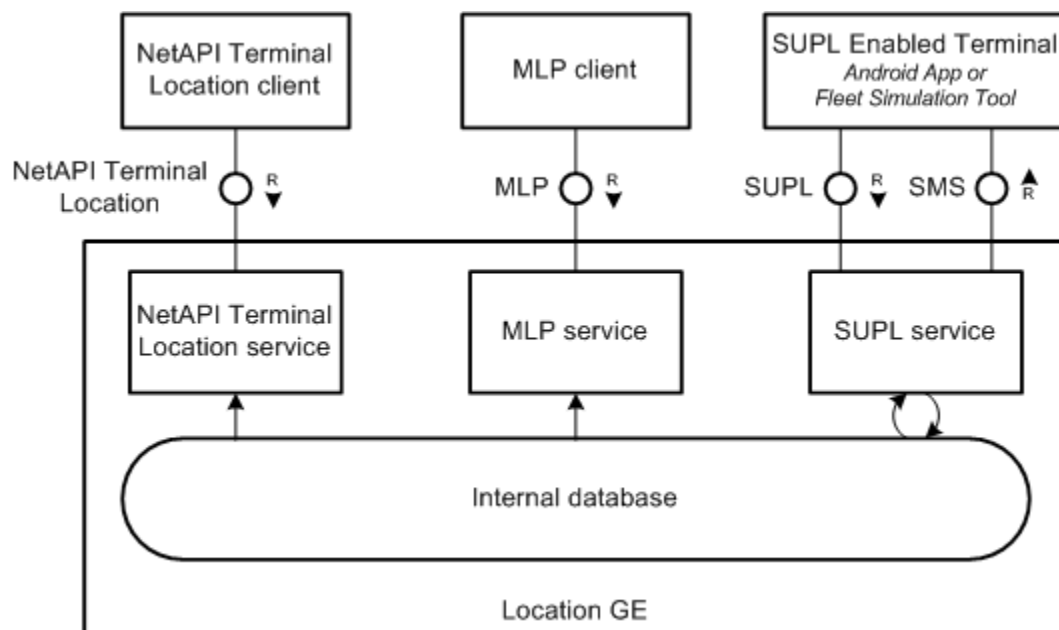
A RESTful interface is available to manage the fleet and each single simulated device which can be found at the following URL: [\[6\]](#).

Based on the simulated position, the Location GE responds to location requests with the most updated position of the simulated mobile. This facilitates the demonstration of single location retrieval, periodic tracking and geofencing use cases.

22.1.4.5 Interfaces and data model

(1) Location GE

The following diagram illustrates the interfaces previously presented:



The following services are the grounds of the Location GE:

- **MLP** service: made of an HTTP stack, it processes MLP compliant requests and after authorization of such request, it triggers the SUPL service to establish communication with the target handset (SMS) 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** service: 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 (SMS) for similar services to MLP.
- **SUPL** service: made of a TCP stack, this server is used both to establish communication with a target handset (SMS) and receive connection from the handset. The SUPL service 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 all services, contains the following data:

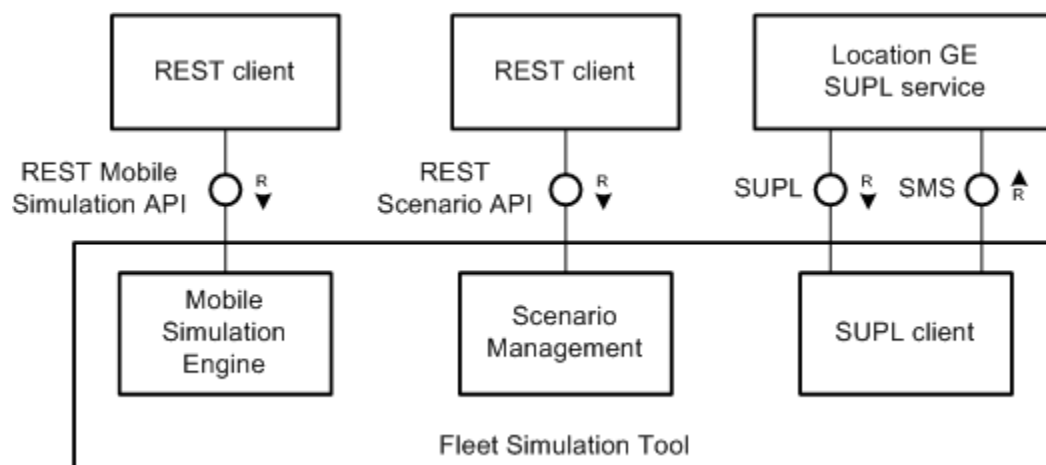
- **Network cell information:** cell identifiers associated with cell mast position and coverage radius to be currently provided by Telco. A dynamic provisioning is planned for future FI-WARE releases in order to build this database with GPS location and cell information retrieved from the SET.
- **Third-party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

(2) Fleet Simulation Tool

The Fleet Simulation Tool illustrated on the above diagram is a SUPLv2 client that has the ability to simulate movement and return associated positions and geofencing events. Its structure is shown below:



This simulation tool is composed of:

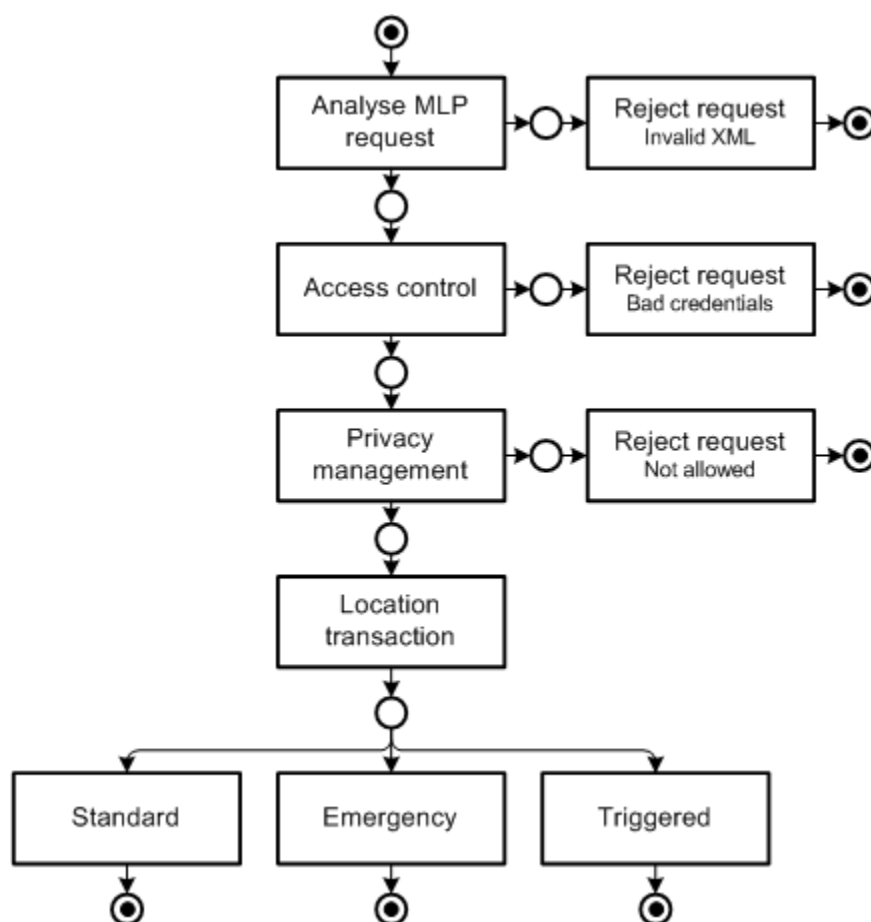
- **Mobile Simulation Engine**, which can be controlled via a dedicated RESTful interface in order to interact with a specific simulated handset. The services available on this interface range from adding/getting/deleting a path and starting/stopping the movement of the handset on the current path. The path is defined as a list of vertices with a context that includes identity and movement parameters.
- **Scenario Management** module also manageable via a dedicated RESTful interface. It offers the ability to select a predefined fleet simulation scenario and start/pause/stop that scenario.

- **SUPL client** module, which is in charge of handling SUPLv2 responses based on simulated terminal positions and SUPLv2 location requests. It supports single shot location retrieval, periodic reporting and geofencing events.

22.1.5 Main Interactions

22.1.5.1 MLP services

The MLP request processing is illustrated on the below diagram. Before processing the location transaction, various checks are performed to parse and authorize the request based on client credentials, service and end-user settings as presented before:



The following sub-sections present the XML structure of MLP requests and their associated responses.

(1) Access control and privacy management

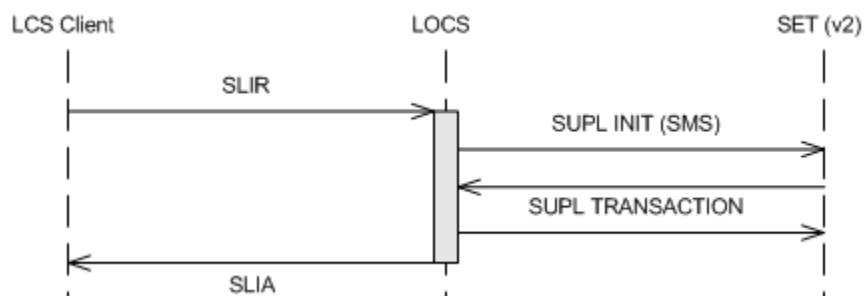
Each of the incoming MLP requests are checked for authentication and authorization before localizing the end-user. The following example shows the MLP request header:

```
<?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 the MSISDN. The <serviceid/> and target end-user MSISDN are utilized to check the end-user privacy policy previously presented.

(2) 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>
```

(3)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>

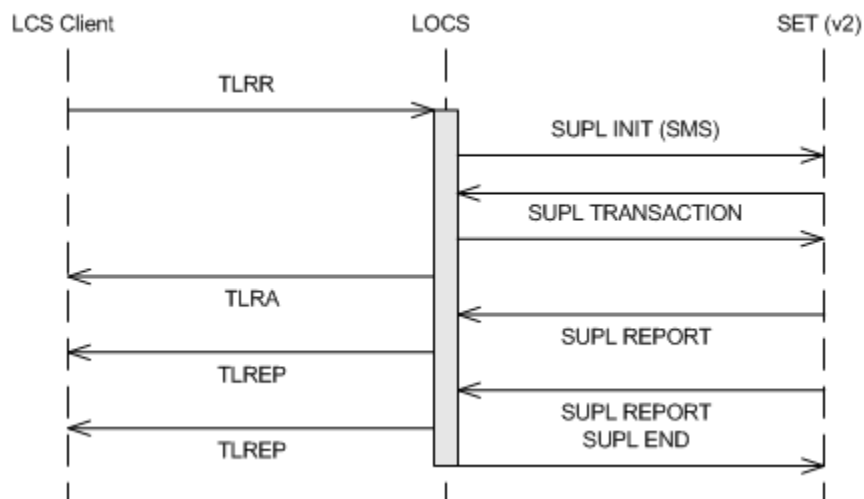
</eme_pos>

</eme_lia>

```

(4) 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>

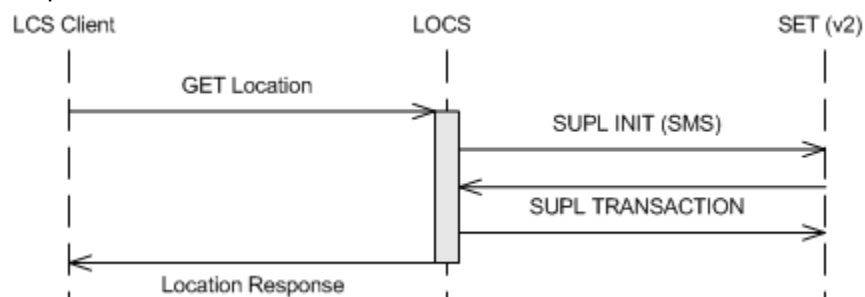
```

22.1.5.2 NetAPI Terminal Location services

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

(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 is provided in the API Specifications (see [references](#)). An example of a request is provided below:


```
GET
/location/v1/queries/location?requester=test:test&address=33611223344&
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>

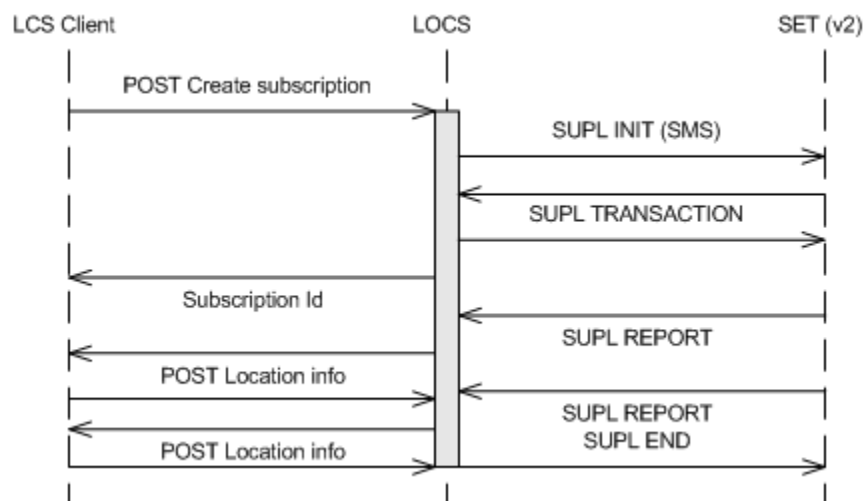
</tl:terminalLocation>

</tl:terminalLocationList>

```

(2) Location Subscriptions

This type of query is used to retrieve either periodic location reports or area entry/leaving/inside/outside location 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 is provided in API Specifications (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/LocationNot
ification</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/circle/sub003</tl:resourceURL>

  <tl:callbackReference>

<tl:notifyURL>http://application.example.com/notifications/LocationNotification</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>
```

22.1.5.3 *Fleet Simulation Tool*

(1) Mobile Simulation

Here is an example of a path addition:

Request :

```
POST /testtool/simulation/mobilepaths HTTP/1.1

Accept: application/xml
```

```
Host: example.com

Content-Length: nnnn

<?xml version="1.0" encoding="UTF-8"?>
<simulationFragment>
  <name>Test Route</name>
  <pathRoutes>
    <pathRoute>
      <context>
        <msisdn>33611223344</msisdn>
        <velocity>1.0</velocity>
        <autoMove>true</autoMove>
        <autoLoop>true</autoLoop>
        <logPath>false</logPath>
      </context>
      <positions>
        <position>
          <name>Position 0</name>
          <latitude>43.545571</latitude>
          <longitude>1.387802</longitude>
          <altitude>31.0</altitude>
        </position>
        <position>
          <name>Position 1</name>
          <latitude>43.5453</latitude>
          <longitude>1.3883</longitude>
          <altitude>31.0</altitude>
        </position>
      </positions>
    </pathRoute>
  </pathRoutes>
</simulationFragment>
```

```
</position>

<position>

  <name>Position 2</name>

  <latitude>43.545107</latitude>

  <longitude>1.388511</longitude>

  <altitude>31.0</altitude>

</position>

<position>

  <name>Position 3</name>

  <latitude>43.544992</latitude>

  <longitude>1.388417</longitude>

  <altitude>31.0</altitude>

</position>

<position>

  <name>Position 4</name>

  <latitude>43.545083</latitude>

  <longitude>1.387808</longitude>

  <altitude>31.0</altitude>

</position>

<position>

  <name>Position 5</name>

  <latitude>43.545240</latitude>

  <longitude>1.387856</longitude>

  <altitude>31.0</altitude>

</position>

</positions>

</pathRoute>
```

```
</pathRoutes>
</simulationFragment>
```

Response :

```
HTTP/1.1 201 Created
Content-Length: nnnn
Date: Thu, 02 Jun 2011 02:51:59 GMT
```

(2)Scenario Management

Here is an example of a scenario selection and start-up: *Request :*

```
PUT
http://127.0.0.1:8111/testtool/scenario/control?select=LocationGE-
TriggerSubscription HTTP/1.1
Host: example.com
```

Response :

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

Request :

```
PUT          http://127.0.0.1:8111/testtool/scenario/control?cmd=start
HTTP/1.1
Host: example.com
```

Response :


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

22.1.5.4 **SUPL Positioning**

(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).

(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' (Wi-Fi). 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.

(3)Location Technology selection

Today, C-ID (including Wi-Fi) 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

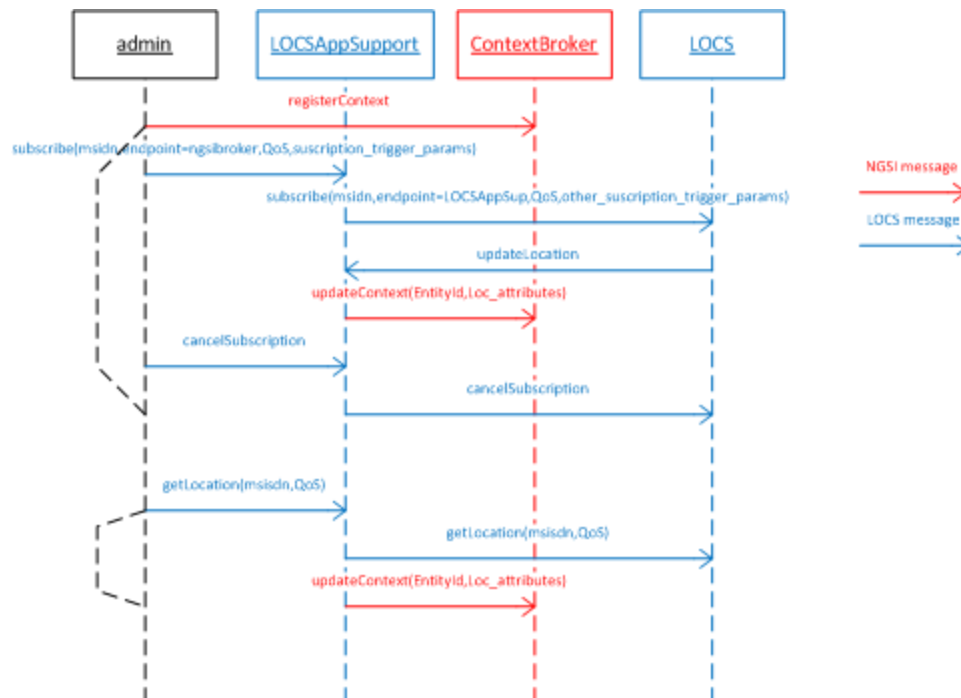
(4)NGSI Integration with Context Broker GE

In order to provide better decoupling between Location GE and location aware target applications, an optional NGSI Integration feature is available. In that case, applications receives positioning information through NGSI Pub/Sub API messages (FI-WARE NGSI API).

The Location GE NGSI integration schema can be summarized in the following sequence diagram covering :

- Initial entity registration
- Location query NGSI scenario
- Trigger Subscription NGSI scenario

The rationale is to insure complete compatibility with the OMA Terminal Location RESTFull API by adding limited extensions and profile format for callback data. In these scenarios, notification (end-point) URL refers always to Context Broker URL.



22.1.6 Basic Design Principles

The Location GE is based on existing OMA standards (refer to [7]):

- 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.

22.2 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)

22.3 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.

22.3.1 Open API Specifications

- [Location Server Open RESTful API Specification](#)

22.4 Re-utilised Technologies/Specifications

The following technologies/specifications are incorporated in this GE :

- Mobile Location Protocol (MLP), Open Mobile Alliance, as specified in OMA-TS-MLP-V3_2-20110719-A,
- Secure User Plane Location Protocol (SUPL), Open Mobile Alliance, as specified in OMA-TS-ULP-V2_0-20111222-D
- Radio Resource LCS Protocol (RRLP), 3GPP, as specified in 3GPP TS 44.031 V9.2.0 (2010-03)
- Terminal Location API, Open Mobile Alliance, as specified in REST_NetAPI_TerminalLocation_V1_0-20120207-C

22.5 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 **avalue**. 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 be 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 two 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 Location_Server_Open_RESTful_API_Specification

23.1 Dedicated API Introduction

Please check the following **FI-WARE Open Specification Legal Notice (essential patents license)** [\[1\]](#) to understand the rights to use these specifications.

23.2 Introduction to the Restful Network API for Terminal Location

23.2.1 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

23.2.2 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."

23.2.3 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	<ul style="list-style-type: none">• Initial Version
Sept 26, 2012	<ul style="list-style-type: none">• Complete Support for url-form-encoded and json API format• Add Periodic Subscription service API
Jan 30, 2014	<ul style="list-style-type: none">• Complete support for optional API extension for NGSI Pub/Sub integration

23.2.4 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.

23.2.5 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."

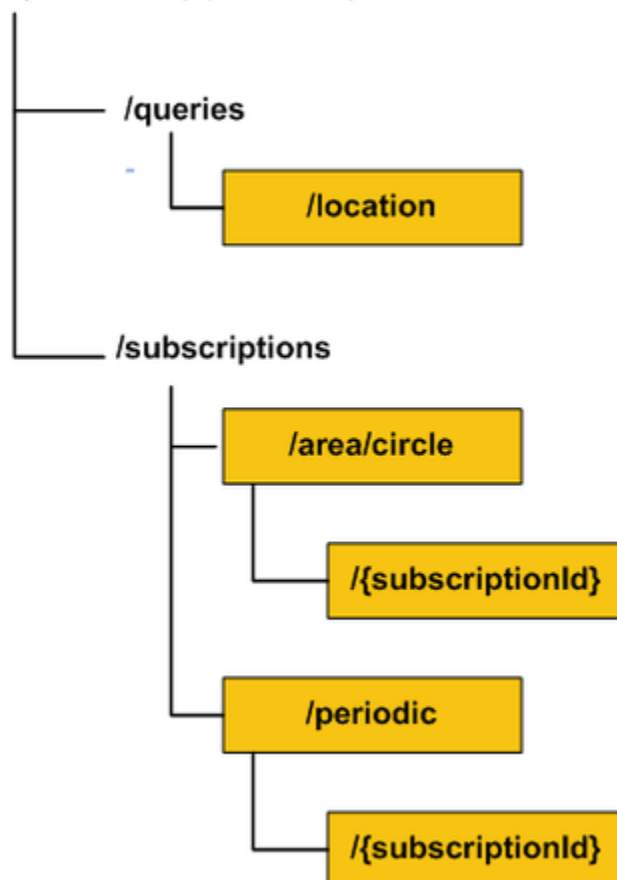
23.3 General Location Server REST API Information

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).

23.3.1 Resources Summary

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

//{serverRoot}/location/{apiVersion}



serverRoot = server base url (hostname+port)

23.3.2 Authentication

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

23.3.3 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 **application/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.

23.3.4 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.

23.3.5 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.

23.3.6 Links and References

None

23.3.7 Limits

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

23.3.8 Versions

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

23.3.9 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]

23.4 Data Types

23.4.1 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

23.4.2 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>

23.4.3 Structures

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

23.4.3.1 *Type:CallbackReference*

There is no constraints on content format of call back data , except if LOCS services are used with Context Broker. In that case, NGSI Integration is achieved by defining specific content for the following properties :

- **notifyURL** : Context Broker URL where location updates are notified.
- **callbackData** : NGSI entity id reference/location attribute to update. By default, ngsi:<entity id>/position is equivalent to ngsi:<entity id> only using default location attribute name.

Nota : locevent attribute name on entity is mandatory.

23.4.3.2 *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.

23.4.3.3 *Type:TerminalLocationList*

A type containing a list of terminal locations.

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

23.4.3.4 *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.

23.4.3.5 *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 Net API Common for details. In case of NGSI integration with Pub/Sub broker see Nota above.

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.

23.4.3.6 *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. In case of N GSI integration with Pub/Sub broker see Nota above.
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	Longitude 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.

23.4.3.7 *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.

23.4.3.8 *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.

23.4.3.9 *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.

23.5 API Operations

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

23.5.1 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>

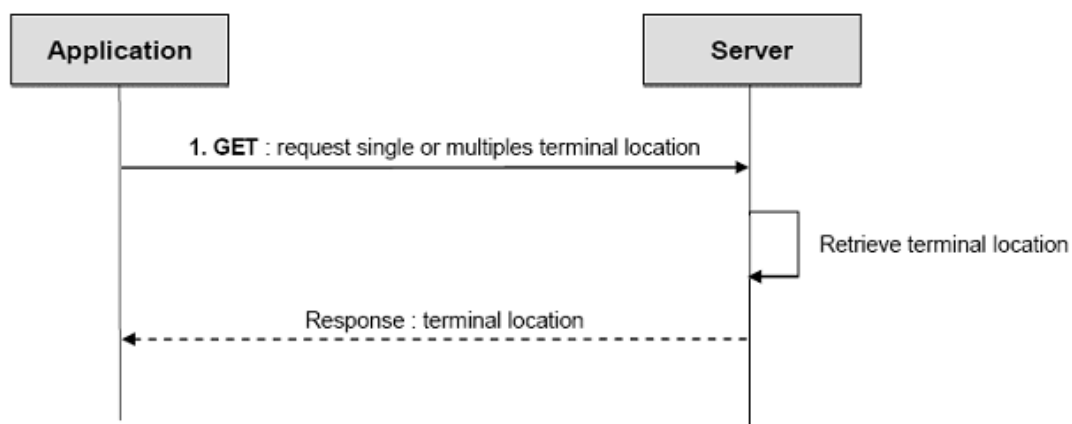


Figure 2 Location query

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.

23.5.1.1 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.
ngsiUpdateURL	xsd:string	Yes	URL of Context Broker Server. Only in case of LOCS NGSI integration to Context Broker.
ngsiEntity	xsd:string	Yes	description of NGSI registered entity position attribute to update.

23.5.1.2 *Response codes*

Code	Description
200	Request is OK
400	Request is KO

23.5.1.3 *Examples*

(1) 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=33611223344&
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=33611
223344&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>
```

Example 4 : Using NGSI integration

Request :

```
GET
/location/v1/queries/location?requester=test:test&address=33611223344&
requestedAccuracy=40&acceptableAccuracy=60

&maximumAge=1000&tolerance=DelayTolerant&ngsiUpdateURL=http://orion-
psb.testbed.fi-ware.eu:1026&ngsiEntity=33611223344 HTTP/1.1

Accept: application/xml

Host: example.com
```

Response :

```
HTTP/1.1 200 No Content
```

When the location session is complete, the following attribute position update notification is sent Context Broker end-point URL.

*(2)Application/json format**Request :*

```
GET
location/v1/queries/location?requester=test:test&address=33611223344&t
olerance=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"
}}
```

```
}}}
```

23.5.2 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 SubscriptionCancellationNotification	or signal notification

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

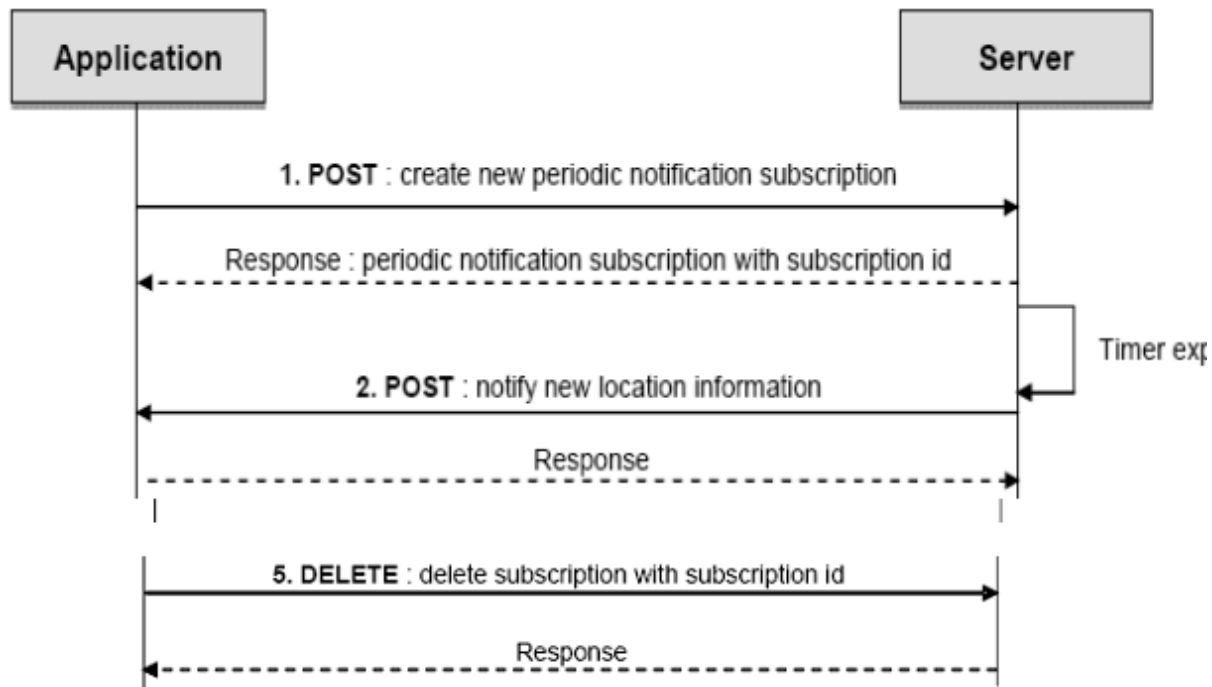
The resource:

- 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.

23.5.2.1 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.

23.5.2.2 *Response Codes*

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

23.5.2.3 *Examples*

(1) *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>

  <common:notifyURL>http://application.example.com/notifications/LocationNotification</common:notifyURL>

  <common:callbackData>4444</common:callbackData>

</tl:callbackReference>

  <tl:requester>test:test</tl:requester>
```

```
<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/periodic/  
sub0003</tl:resourceURL>  
  
  <tl:callbackReference>  
  
    <common:notifyURL>http://application.example.com/notifications/Locatio  
nNotification</common:notifyURL>  
  
    <common:callbackData>4444</common:callbackData>  
  
  </tl:callbackReference>
```

```
<tl:requester>test:test</tl:requester>
<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">
  <common:callbackData>4444</common: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
```

*(2)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%2FLocationNotification&

callbackData=4444&

requester=test%3Atest&

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/periodic/
sub0003</tl:resourceURL>

<tl:callbackReference>

<common:notifyURL>http://application.example.com/notifications/LocationNotification</common:notifyURL>

<common:callbackData>4444</common:callbackData>

</tl:callbackReference>

<tl:requester>test:test</tl:requester>

<tl:address>tel:+19585550100</tl:address>

<tl:requestedAccuracy>10</tl:requestedAccuracy>

<tl:frequency>10</tl:frequency>

<tl:duration>100</tl:duration>

</tl:periodicNotificationSubscription>
```

(3)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/sub0
03",

"requestedAccuracy": "10"

}}

```

23.5.3 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 SubscriptionCancellationNotification	or 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.

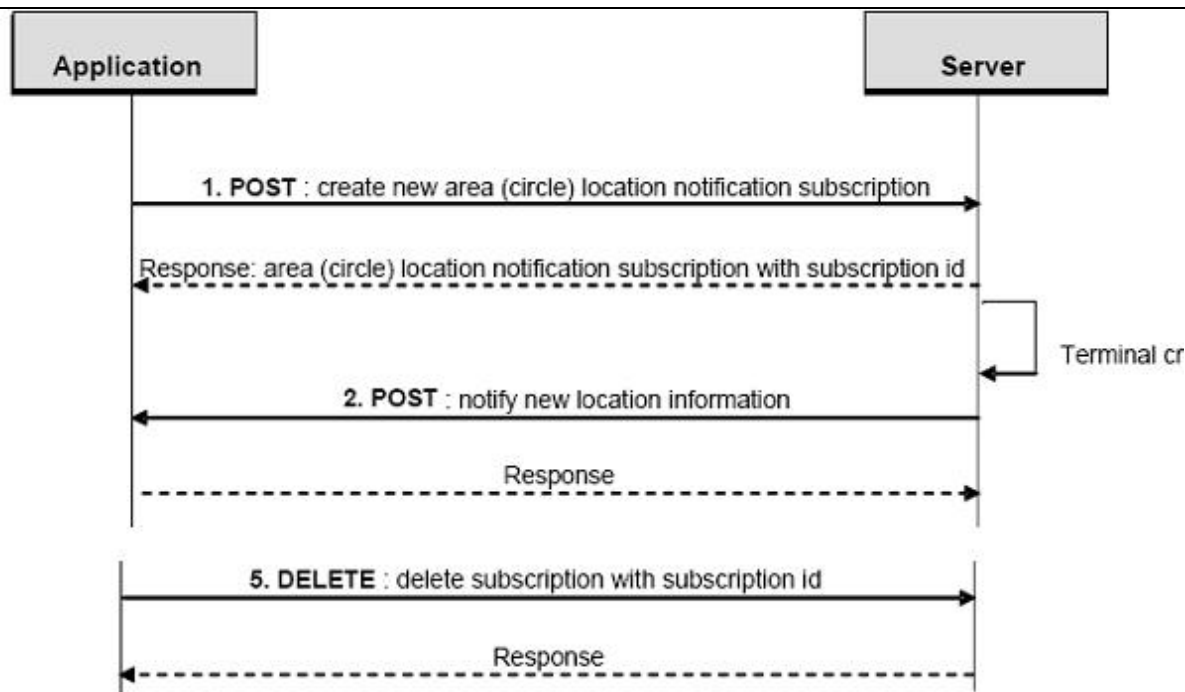
The resource:

- 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.

23.5.3.1 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.

23.5.3.2 *Response Codes*

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

23.5.3.3 *Examples*

(1) *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>

  <common:notifyURL>http://application.example.com/notifications/LocationNotification</common:notifyURL>

  <common:callbackData>4444</common:callbackData>
```

```

</tl:callbackReference>

<tl:requester>test:test</tl:requester>

<tl:address>tel:+19585550100</tl:address>

<tl:latitude>100.23</tl: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>5</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/circle/sub0003</tl:resourceURL>

<tl:callbackReference>

<common:notifyURL>http://application.example.com/notifications/LocationNotification</common:notifyURL>

<common:callbackData>4444</common:callbackData>

</tl:callbackReference>

<tl:requester>test:test</tl:requester>

<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>5</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">

  <common:callbackData>4444</common: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/sub0003"
"/>

</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
```

Example 3: Using NGSI integration with Context Broker**Request :**

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

<tl:circleNotificationSubscription
xmlns:tl="urn:oma:xml:rest:netapi:terminallocation:1"
xmlns:common="urn:oma:xml:rest:netapi:common:1">

  <tl:clientCorrelator>0003</tl:clientCorrelator>

  <tl:callbackReference>

    <common:notifyURL>http://orion-psb.testbed.fi-
ware.eu:1026</common:notifyURL>

    <common:callbackData>ngsi:33611223344</common:callbackData>

  </tl:callbackReference>

  <tl:requester>test:test</tl:requester>

  <tl:address>tel:+33611223344</tl:address>

  <tl:latitude>43.60091</tl:latitude>

  <tl:longitude>1.44299</tl:longitude>
```

```

        <tl:radius>1500</tl:radius>

        <tl:trackingAccuracy>50</tl:trackingAccuracy>

<tl:enteringLeavingCriteria>Entering</tl:enteringLeavingCriteria>

        <tl:frequency>60</tl:frequency>

        <tl:duration>120</tl:duration>

        <tl:count>2</tl:count>

</tl:circleNotificationSubscription>

```

Response :

```

HTTP/1.1 201 Created
Content-Type: application/xml
Location:
http://127.0.0.1:10000/location/v1/subscriptions/area/circle/sub003

<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>/location/v1/subscriptions/area/circle/sub003</tl:res
ourceURL>

    <tl:callbackReference>

        <common:notifyURL>http://orion-psb.testbed.fi-
ware.eu:1026</common:notifyURL>

        <common:callbackData>ngsi:33611223344</common:callbackData>

    </tl:callbackReference>

    <tl:requester>test:test

</tl:requester>

```

```
<tl:address>tel:33611223344</tl:address>
<tl:latitude>43.60091</tl:latitude>
<tl:longitude>1.44299</tl:longitude>
<tl:radius>1500.0</tl:radius>
<tl:trackingAccuracy>50.0</tl:trackingAccuracy>
<tl:enteringLeavingCriteria>Entering</tl:enteringLeavingCriteria>
<tl:checkImmediate>false</tl:checkImmediate>
<tl:frequency>60</tl:frequency>
<tl:duration>120</tl:duration>
<tl:count>2</tl:count>
</tl:circleNotificationSubscription>
```

LOCS sends the position/locevent update notification for entering events to Context Broker end-point URL.

(2)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%2FLocationNotification&
```



```
callbackData=4444&
requester=test%3Atest&
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/circle/sub0003</common:resourceURL>

<tl:callbackReference>

<common:notifyURL>http://application.example.com/notifications/LocationNotification</common:notifyURL>

<common:callbackData>4444</tl:callbackData>

</tl:callbackReference>

<tl:requester>test:test</tl:requester>

<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>

```

(3) 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/s
ub0003",
  "trackingAccuracy": "10"
}}
```

24 FIWARE OpenSpecification Data MetadataPreprocessing

Name	FIWARE.OpenSpecification.Data.MetadataPreprocessing		
Chapter	Data/Context Management,		
Catalogue-Link to Implementation	MetadataProcessor	Owner	Siemens AG, Peter Amon

24.1 Preface

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

24.2 Copyright

- Copyright © 2012-2014 by [SIEMENS](#)

24.3 Legal Notice

Please check the following [Legal Notice](#) to understand the rights to use these specifications.

24.4 Overview

24.4.1 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.

24.4.2 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 device (i.e., another GE or an application or another external 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 performing the transformation of the metadata format (e.g., defined by XML Schema), 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. For example, in the Use Case project OUTSMART (more specifically, in the Santander cluster), coma-separated sensor metadata is transformed into XML metadata by the Metadata Preprocessing GE for further processing and storage. The transformation task is described in more detail in the following. OUTSMART's advanced measure & managing system (AMMS) is constantly producing sensor data for the following format:

```
[...]
06E1E5A2108003821;29/05/2012;01;1;AI;000107;0
06E1E5A2108003821;29/05/2012;01;1;RI;000012;0
[...]
```

The format can be interpreted as follows: AMMS identifier; date; hour; quarter (of hour); type of measurement; data measured; error code. The Metadata Preprocessing GE transforms this data into XML for further processing.

```
<io>
  <obs from="urn:outsmart:06E1E5A2108003821">
    <stm>2012-05-29T10:00:00+02.00</stm>
    <what href="phenomenon:activepower"/>
    <param
href="identifier:UniversalIdentifierOfLogicalHub"><text>sc.poc.outsmar
t.eu</text></param>
    <data><quan uom="uom:watts/h">107</quan></data>
  </obs>
```

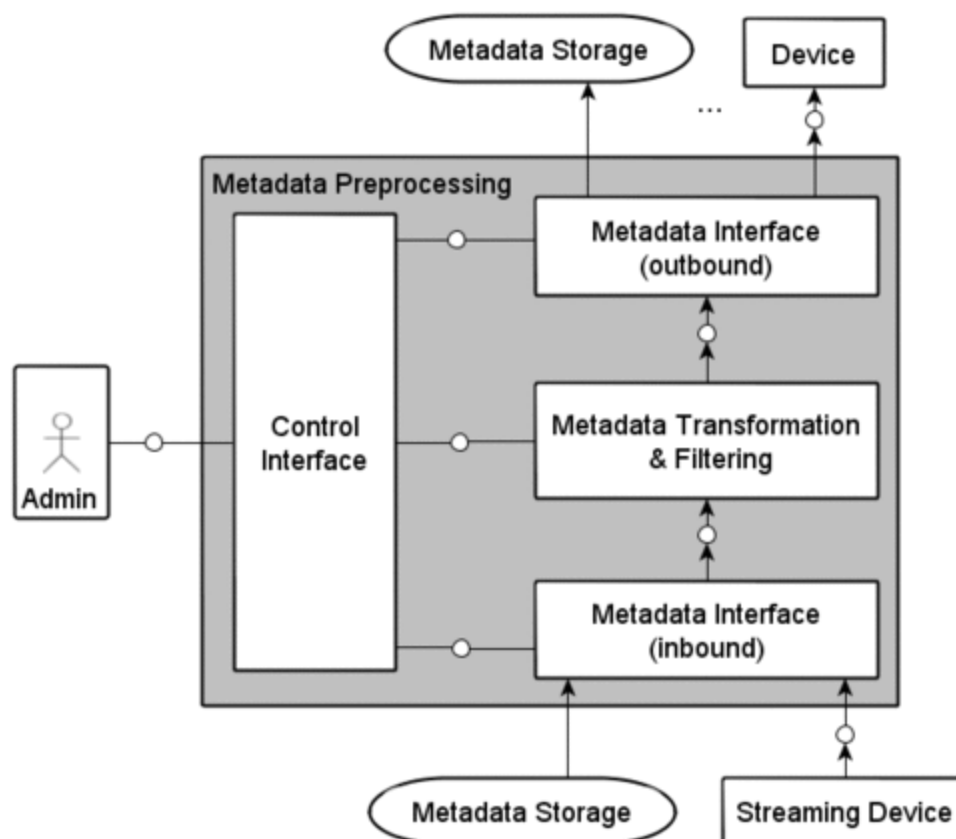
```
<obs from="urn:outsmart:06E1E5A2108003821">
  <stm>2012-05-29T10:00:00+02.00</stm>
  <what href="phenomenon:reactivepower"/>
  <param
href="identifier:UniversalIdentifierOfLogicalHub"><text>sc.poc.outsmar
t.eu</text></param>
  <data><quan uom="uom:Varh">12</quan></data>
</obs>
</io>
```

Note that a special transformation unit (implemented by the GE owner) was necessary to realize this task. (The transformation could not be specified by an XSLT stylesheet, since the input data was not in XML format.)

24.5 Basic Concepts

24.5.1 Functional components of the Metadata Preprocessing GE

The following figure depicts the components of the Metadata Preprocessing Generic Enabler. These functional blocks are the Control Interface, the Metadata Interface for inbound streams, the Metadata Transformation & Filtering component, and the 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 bullet points.

- 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, i.e., connected using the *configureInstance* method. Sinks receiving the outbound streams can connect using the information provided in the *getMetadata* method. More details on the APIs are described in the Section "Main Interactions".
- Metadata Interface (for inbound streams):** Different interchange formats (such as the ones for streaming or for file access) can be realized (i.e., configured or programmed into this interface at a later stage). At the current stage, the Real-time Transport Protocol (RTP) as standardized in [RFC 3550](#) [RFC3550] is supported as interchange format. Different packetization formats for the contained/packetized payload data (i.e., the XML metadata) depending on the application might be used. Usually, the Inbound Metadata Interface" as a (RTSP/RTP) streaming client connected to a (RTP/RTSP) streaming server (i.e., the Streaming Device).
- Metadata Transformation & Filtering:** The Metadata Transformation & Filtering 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, other kinds of transformations (other than XSLT) can also be applied

(e.g., text-to-XML transformation); however, dedicated changes to the enabler are needed for this. Metadata filtering is an optional step in the processing chain. The filtering can be used, e.g., for thinning and aggregating the metadata, or simple fact generation (i.e., simple reasoning on the transformed metadata). Filtering is usually done during transformation by using XSLT technology. The output of this step is a new encapsulation/formatting of the metadata received.

- **Metadata Interface** (for **outbound** streams): Through this interface, the transformed (and possibly filtered) metadata or metadata stream is accessed. For example, the "Device" connected to the Outbound Metadata Interface can be a (RTSP/RTP) streaming client. In this case, the Outbound Metadata Interface acts as a (RTSP/RTP) streaming server.

24.5.2 Realization by MetadataProcessor asset

The MetadataProcessor asset realizes a specific implementation of the Metadata Preprocessing GE. Timed metadata (i.e., metadata elements with associated timing information, e.g., a timestamp) is received over an RTSP/RTP interface (as specified in [RFC 2326](#) [RFC2326] and [RFC 3550](#) [RFC3550], respectively), which implements the metadata interface for inbound data/streams. Different RTP sessions can be handled; therefore metadata streams can be received from several components/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.

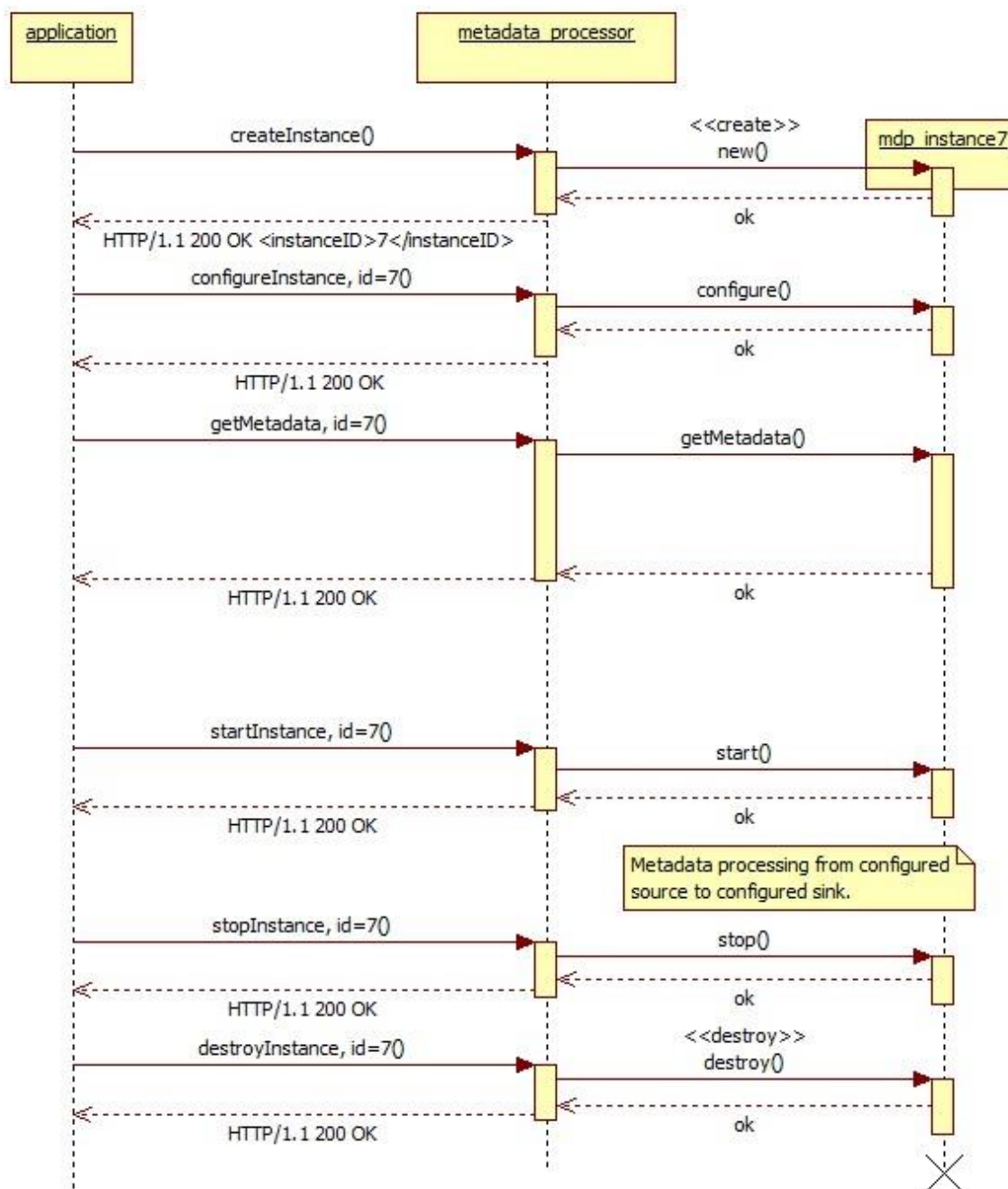
24.6 Main Interactions

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 or external applications/components). 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 instance 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).
- **getMetadata**: The URI (i.e., RTSP URL) of the metadata output stream is returned.

The following figure explains the main interactions in a (very general) example usage. 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**, **getMetadata**) is arbitrary. Note further that more than one sink can be added as receiving component, but only one source can be configured. (However, additional processing units for metadata transformation can be created using **createInstance**.) In a fifth step, the processing is started.



Example usage

After the processing is done, the specific instance of the GE is stopped. Note that the instance could be started again afterwards by re-using its instance ID. (A list of all existing instances can be retrieved using the **listInstances** request.) Also the processing of the source could be reconfigured and sinks can be

added or removed. As a final step in this example usage, the specific metadata preprocessing instance is destroyed. Note that it is not necessary to stop the instance before destroying it, since this will be done automatically. A simple but concrete example for metadata transformation and metadata filtering can be found in the Open RESTful API Specification of this GE.

24.7 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 is implemented as-a-service, usable from other web applications or components.
- Transformation is based on standardized and commonly used XML Stylesheet Language for Transformations (XSLT).

24.8 References

[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.
[XSLT]	W3C / M. Kay (editor), "XSL Transformations (XSLT) Version 2.0", http://www.w3.org/TR/xslt20/ , Jan. 2007.

24.9 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.

24.9.1 Open API Specifications

- [Metadata Preprocessing Open RESTful API Specification](#)

24.10 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](#).

24.11 Terms and definitions

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

- **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 **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 be 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 two 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 are linked to an event, this is 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.

25 Metadata Preprocessing Open RESTful API Specification

25.1 Introduction to the Metadata Preprocessing GE API

25.1.1 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 [Legal Notice](#) to understand the rights to use FI-WARE Open Specifications.

25.1.2 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.

25.1.3 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
Apr 25, 2013	Update for Release 2.2
Sep 27, 2013	Update for Release 3.1

Dec 15, 2013	Update for Release 3.2
Feb 3, 2014	Minor update for deliverable D.6.1.c

25.1.4 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](#).

25.1.5 Additional Resources

You can download the most current version of this document from the FIWARE API specification website at [Summary of FI-WARE Open Specifications](#). For more details about the Metadata Preprocessing GE service that this API is based upon, please refer to [Metadata Preprocessing GE Product Vision](#) and [Metadata Preprocessing GE Architecture](#).

25.2 General Metadata Preprocessing GE API Information

25.2.1 Resources Summary

The resource summary is shown in the following overview.

```
Metadata Preprocessing GE (server)
-----
//{serverRoot}/{assetName}
|
```

getVersion	--- /version	GET ->
listInstances	--- /instances/	GET ->
createInstance		POST ->
getInstanceInfo	--- {instanceID}	GET ->
destroyInstance		DELETE ->
startInstance	--- ?action=start	PUT ->
stopInstance	--- ?action=stop	PUT ->
getConfig	--- /config	GET ->
configureInstance		PUT ->
getMetadata	--- /metadata	GET ->
listSinks (removed)	--- /sinks	GET ->
addSink (removed)		POST ->

getSinkInfo (removed)	--- /{sinkID}	GET	->
removeSink (removed)		DELETE	->

25.2.2 Authentication

Authentication is not supported in the current version of the Metadata Preprocessing GE.

25.2.3 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).

25.2.4 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.

25.2.5 Resource Identification

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

25.2.6 Links and References

Request forwarding is not supported in the current version of the Metadata Preprocessing GE.

25.2.7 Limits

Limits are not yet identified or specified for the current version of the Metadata Preprocessing GE.

25.2.8 Versions

Querying the version is supported by the **getVersion** command of the Metadata Preprocessing GE, i.e., by placing the HTTP request "GET //{serverRoot}/mdp/version HTTP/1.1".

25.2.9 Extensions

Querying extensions is not supported in the current version of the Metadata Preprocessing GE.

25.2.10 Faults

25.2.10.1 *Synchronous Faults*

Synchronous 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]

25.2.10.2 *Asynchronous Faults*

Asynchronous fault elements are not sent by the current implementation of the Metadata Preprocessing GE.

25.3 API Operations

Please note that the XML body of the requests are sent in one line and are only broken into several lines for better readability.

25.3.1 Version

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

25.3.1.1 *getVersion*

Example:

```
GET //127.0.0.1/mdp/version HTTP/1.1
Accept: application/xml
```

Sample result:

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

<VersionResponse xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.datacontract.org/2004/07/mdpp.rest.Dto">

  <Version>3.2</Version>
</VersionResponse>
```

25.3.2 Management of instances

Verb	URI	Description
GET	<code>://{serverRoot}/{assetName}/instances</code>	listInstances : lists all instances (i.e., processing units) of the Metadata Preprocessing GE
POST	<code>://{serverRoot}/{assetName}/instances</code>	createInstance : creates an instance (i.e., a processing unit) of the Metadata Preprocessing GE
DELETE	<code>://{serverRoot}/{assetName}/instance/{instanceID}</code>	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

25.3.2.1 *listInstances*

Example:

```
GET //127.0.0.1/mdp/instances HTTP/1.1
Accept: application/xml
```

Sample result:

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

<InstancesResponse xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.datacontract.org/2004/07/mdpp.rest.Dto">

  <Instances>

    <InstanceInfo>

      <Id>0</Id>

      <MetadataUri>rtsp://127.0.0.1:1554/0</MetadataUri>

      <SourceUri>rtsp://127.0.0.1:1554/stream0</SourceUri>

    </InstanceInfo>

  </Instances>

</InstancesResponse>
```

Other sample result:

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

<InstancesResponse xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.datacontract.org/2004/07/mdpp.rest.Dto">

  <Instances />
</InstancesResponse>
```

25.3.2.2 *createInstance*

Example:

```
POST //127.0.0.1/mdp/instances HTTP/1.1
Accept: application/xml
```

Sample result:

```
HTTP/1.1 201 Created
Content-Type: application/xml

<InstancesResponse xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.datacontract.org/2004/07/mdpp.rest.Dto">

  <Instances>

    <InstanceInfo>

      <Id>0</Id>

      <MetadataUri>rtsp://127.0.0.1:1554/0</MetadataUri>

      <SourceUri>rtsp://127.0.0.1:1554/stream0</SourceUri>

    </InstanceInfo>

  </Instances>
</InstancesResponse>
```

25.3.2.3 *destroyInstance*

Example:

```
DELETE //127.0.0.1/mdp/instances/0 HTTP/1.1  
Accept: application/xml
```

Sample result:

```
HTTP/1.1 200 OK  
Content-Type: application/xml  
  
<ActionResponse      xmlns:i="http://www.w3.org/2001/XMLSchema-instance"  
xmlns="http://schemas.datacontract.org/2004/07/mdpp.rest.Dto">  
  <Action>Delete</Action>  
  <Id>0</Id>  
</ActionResponse>
```

25.3.2.4 *startInstance*

Example:

```
PUT //127.0.0.1/mdp/instances/0?action=start HTTP/1.1  
Accept: application/xml
```

Sample result:

```
HTTP/1.1 200 OK  
Content-Type: application/xml  
  
<ActionResponse      xmlns:i="http://www.w3.org/2001/XMLSchema-instance"  
xmlns="http://schemas.datacontract.org/2004/07/mdpp.rest.Dto">
```

```
<Action>Start</Action>

<Id>0</Id>

</ActionResponse>
```

25.3.2.5 *stopInstance*

Example:

```
PUT //127.0.0.1/mdp/instances/0?action=stop HTTP/1.1
Accept: application/xml
```

Sample result:

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

<ActionResponse      xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.datacontract.org/2004/07/mdpp.rest.Dto">

  <Action>Stop</Action>

  <Id>0</Id>

</ActionResponse>
```

25.3.3 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}/metadata	getMetadata : returns metadata URI (i.e., RTSP URL)

25.3.3.1 *getConfig*

Example:

```
GET //127.0.0.1/mdp/instances/0/config HTTP/1.1
Accept: application/xml
```

Sample result:

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

<ConfigurationResponse      xmlns:i="http://www.w3.org/2001/XMLSchema-
instance"
xmlns="http://schemas.datacontract.org/2004/07/mdpp.rest.Dto">

  <Config>

    <ProcessorCsv2Xml>

      <Enabled>true</Enabled>

      <FieldDelimiter>SEMICOLON</FieldDelimiter>

      <GroupEmptyFields>true</GroupEmptyFields>

      <NumHeaderRows>1</NumHeaderRows>

      <SkipHeaderRows>true</SkipHeaderRows>

      <TextDelimiter>SINGLE_QUOTE</TextDelimiter>

    </ProcessorCsv2Xml>

    <ProcessorXslt>

      <Enabled>false</Enabled>

      <Stylesheet>

PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4KPCEtLSBwZXJzb25saX
N0LnhtbCB0byBvYmplY3RsaXN0LnhtbCAtLT4KPHhzb
```



```

DpzdHlsZXNoZWV0IHZlcnNpb249IjEuMCIgeG1sbnM6eHNsPSJodHRwOi8vd3d3LnczLm9
yZy8xOTk5L1hTTC9UcmFuc2Zvcml0iPgogIDx4c2w6dG

VtcGxhdGUgbWF0Y2g9Ii9wZXJzb25fbGlzdCI+CiAgICA8b2JqZWNOX2xpc3Q+CiAgICA
gIDx4c2w6YXBwbHktdGVtcGxhdGVzIC8+CiAgICA8L29

iamVjdF9saXN0PgogIDwveHNsOnRlbXBsYXRlPgogIDx4c2w6dGVtcGxhdGUgbWF0Y2g9I
nBlcnNvb2I+CiAgICA8b2JqZWNOIHR5cGU9InBlcnNv

biI+CiAgICA8IDxpZD4KICA8ICA8eHNsOmFwcGx5LXRlbXBsYXRlc3Q9Im
lkIiAvPgogICA8L2lkPgogICA8bGFzZWw+C

iAgICA8ICA8PHhzbDphcHBseS10ZWlwbGF0ZXMgc2VsZWNOPSJuYW1lIiAvPgogICA8ICA
8L2xhYmVsPgogICA8PC9vYmplY3Q+CiAgPC94c2w6dG

    VtcGxhdGU+CjwveHNsOnN0eWxlc2hlZXQ+

    </Stylesheet>

    </ProcessorXslt>

    <SourceURI>rtsp://127.0.0.1:1554/stream0</SourceURI>

    </Config>

    <Id>0</Id>

</ConfigurationResponse>

```

Please note: for technical reasons the XSLT stylesheet is BASE64 encoded.

25.3.3.2 *configureInstance (XSLT)*

The following example demonstrates the configuration of an XSLT transformation, i.e., 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 //127.0.0.1/mdp/instances/0/config HTTP/1.1

Content-Type: application/xml

Content-Length: 544

```

```

<Configuration      xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.datacontract.org/2004/07/mdpp.rest.Dto">

  <Config>

    <ProcessorCsv2Xml>

      <Enabled>false</Enabled>

      <FieldDelimiter>SEMICOLON</FieldDelimiter>

      <GroupEmptyFields>true</GroupEmptyFields>

      <NumHeaderRows>1</NumHeaderRows>

      <SkipHeaderRows>false</SkipHeaderRows>

      <TextDelimiter>SINGLE_QUOTE</TextDelimiter>

    </ProcessorCsv2Xml>

    <ProcessorXslt>

      <Enabled>true</Enabled>

      <Stylesheet>

Cjw/eGlsIHZlcnNpb249IjEuMCIgZW5jb2Rpbmc9IlVURi04Ij8+CjwhLS0gcGVyc29ubG
1zdC54bWwgdG8gb2JqZWNObG1zdC54bWw

gLS0+Cjx4c2w6c3R5bGVzaGVldCB2ZXJzaW9uPSIxLjAiIHhtbG5zOnhzbD0iaHR0cDovL
3d3dy53My5vcmcvMTk5OS9YU0wvVHJhbn

Nmb3JtIj4KICA8eHNsOnRlbXBsYXRlIG1hdGNoPSIvcGVyc29uX2xpc3QiPgogICAgPG9i
amVjdF9saXN0PgogICAgICA8eHNsOmFwc

Gx5LXRlbXBsYXRlcYAvPgogICAgPC9vYmplY3RfbGlzdD4KICA8L3hzbDp0ZW1wbGF0ZT4
KICA8eHNsOnRlbXBsYXRlIG1hdGNoPSJw

ZXJzb24iPgogICAgPG9iamVjdCB0eXB1PSJwZXJzb24iPgogICAgICA8aWQ+CjAgICAgIC
AgPHhzbDphcHBseS10ZW1wbGF0ZXMgc2V

sZWN0PSJpZCIgLz4KICA8ICAgPC9pZD4KICA8ICAgPGxhYmVsPgogICAgICAgIDx4c2w6Y
XBwbHktdGVtcGxhdGVzIHNlbGVjdD0ibm

```

```
FtZSIgIz4KICAgICAgPC9sYWJlbD4KICAgIDwvb2JqZWNOPgogIDwveHNsOnRlbXBsYXRl
Pgo8L3hzbDpzdHlsZXNoZWV0Pgo=
```

```

    </Stylesheet>

    </ProcessorXslt>

    <SourceURI>rtsp://127.0.0.1:1554/stream0</SourceURI>

</Config>

<Id>0</Id>

</Configuration>
```

Please note: for technical reasons the XSLT stylesheet must be BASE64 encoded.

Please note: the “Content-Length” field must be present and set correctly.

Sample result:

```
HTTP/1.1 200 OK
```

```
Content-Type: application/xml
```

```

<ConfigurationResponse      xmlns:i="http://www.w3.org/2001/XMLSchema-
instance"
mlns="http://schemas.datacontract.org/2004/07/mdpp.rest.Dto">

    <Config>

        <ProcessorCsv2Xml>

            <Enabled>false</Enabled>

            <FieldDelimiter>SEMICOLON</FieldDelimiter>

            <GroupEmptyFields>true</GroupEmptyFields>

            <NumHeaderRows>1</NumHeaderRows>

            <SkipHeaderRows>true</SkipHeaderRows>

            <TextDelimiter>SINGLE_QUOTE</TextDelimiter>

        </ProcessorCsv2Xml>
```

```

<ProcessorXslt>

  <Enabled>true</Enabled>

  <Stylesheet>

PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4KPCEtLSBwZXJzb25saX
N0LnhtbCB0byBvYmplY3RsaxN0LnhtbCAtLT4KPHhzb

DpzdHlsZXNoZWV0IHZlcnNpb249IjEuMCIgeGlsbnM6eHNsPSJodHRwOi8vd3d3LnczLm9
yZy8xOTk5L1hTTC9UcmFuc2Zvcml0iPgogIDx4c2w6dG

VtcGxhdGUgbWF0Y2g9Ii9wZXJzb25fbGlzdCI+CiAgICA8b2JqZWN0X2xpc3Q+CiAgICAg
IDx4c2w6YXBwbHktdGVtcGxhdGVzIC8+CiAgICA8L29

iamVjdF9saXN0PgogIDwveHNsOnRlbXBsYXRlPgogIDx4c2w6dGVtcGxhdGUgbWF0Y2g9I
nBlcnNvb2I+CiAgICA8b2JqZWN0IHR5cGU9InBlcnNv

biI+CiAgICAgIDxpZD4KICAgICAgICA8eHNsOmFwcGx5LXRlbXBsYXRlc3Q9Im
lkIiAvPgogICAgICA8L2lkPgogICAgICA8bGFiZWw+C

iAgICAgICAgPHhzbDphcHBseS10ZW1wbGF0ZXMgc2VsZWV0PSJuYW1lIiAvPgogICAgICA
8L2xhYmVsPgogICAgPC9vYmplY3Q+CiAgPC94c2w6dG

  VtcGxhdGU+CjwveHNsOnN0eWxlc2hlZXQ+

  </Stylesheet>

</ProcessorXslt>

  <SourceURI>rtsp://127.0.0.1:1554/stream0</SourceURI>

</Config>

<Id>0</Id>

</ConfigurationResponse>

```

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.

25.3.3.3 *configureInstance (CSV2XML)*

Another configuration option that is supported by the Metadata Preprocessing GE is the transformation from the CSV (comma separated values) format to XML. An example is shown in the following.

```
PUT //127.0.0.1/mdp/instances/0/config HTTP/1.1
Content-Type: application/xml
Content-Length: 544
```

```
<Configuration      xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.datacontract.org/2004/07/mdpp.rest.Dto">

  <Config>

    <ProcessorCsv2Xml>

      <Enabled>true</Enabled>

      <FieldDelimiter>COMMA</FieldDelimiter>

      <GroupEmptyFields>true</GroupEmptyFields>

      <NumHeaderRows>1</NumHeaderRows>

      <SkipHeaderRows>>false</SkipHeaderRows>

      <TextDelimiter>SINGLE_QUOTE</TextDelimiter>

    </ProcessorCsv2Xml>

    <ProcessorXslt>

      <Enabled>>false</Enabled>

      <Stylesheet />

    </ProcessorXslt>

    <SourceURI>rtsp://127.0.0.1:1554/stream0</SourceURI>

  </Config>

  <Id>0</Id>

</Configuration>
```

Sample result:

HTTP/1.1 200 OK

Content-Type: application/xml

```
<ConfigurationResponse      xmlns:i="http://www.w3.org/2001/XMLSchema-
instance"
xmlns="http://schemas.datacontract.org/2004/07/mdpp.rest.Dto">

  <Config>
```

```

<ProcessorCsv2Xml>
  <Enabled>ture</Enabled>
  <FieldDelimiter>COMMA</FieldDelimiter>
  <GroupEmptyFields>true</GroupEmptyFields>
  <NumHeaderRows>1</NumHeaderRows>
  <SkipHeaderRows>>false</SkipHeaderRows>
  <TextDelimiter>SINGLE_QUOTE</TextDelimiter>
</ProcessorCsv2Xml>
<ProcessorXslt>
  <Enabled>>false</Enabled>
  <Stylesheet />
</ProcessorXslt>
<SourceURI>rtsp://127.0.0.1:1554/stream0</SourceURI>
</Config>
<Id>0</Id>
</ConfigurationResponse>

```

Note that CSV2XML transform and XSLT can be combined by enabling both filters.

25.3.3.4 *getMetadata*

Example:

```

GET //127.0.0.1/mdp/instances/0/metadata HTTP/1.1
Accept: application/xml

```

Sample result:

```

HTTP/1.1 200 OK
Content-Type: application/xml

```

```
<MetadataResponse    xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.datacontract.org/2004/07/mdpp.rest.Dto">

    <InstanceId>0</InstanceId>

    <MetadataUri>rtsp://127.0.0.1:1554/0</MetadataUri>

</MetadataResponse>
```

A metadata sink can connect to the RTSP address in the response.

26 FIWARE OpenSpecification Data CompressedDomainVideoAnalysis

Name	FIWARE.OpenSpecification.Data.CompressedDomainVideoAnalysis		
Chapter	Data/Context Management,		
Catalogue-Link to Implementation	Codoan	Owner	Siemens AG, Marcus Laumer

26.1 Preface

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

26.2 Copyright

Copyright © 2012-2014 by [Siemens AG](#)

26.3 Legal Notice

Please check the following [Legal Notice](#) to understand the rights to use these specifications.

26.4 Overview

In the media era of the web, much content is user-generated (UGC) and spans 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.

The Compressed Domain Video Analysis GE consists of a set of tools for analyzing video streams in the compressed domain, i.e., the received streams are either directly processed without prior decoding or just few relevant elements of the stream are parsed to be used within the analysis.

26.4.1 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. 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).

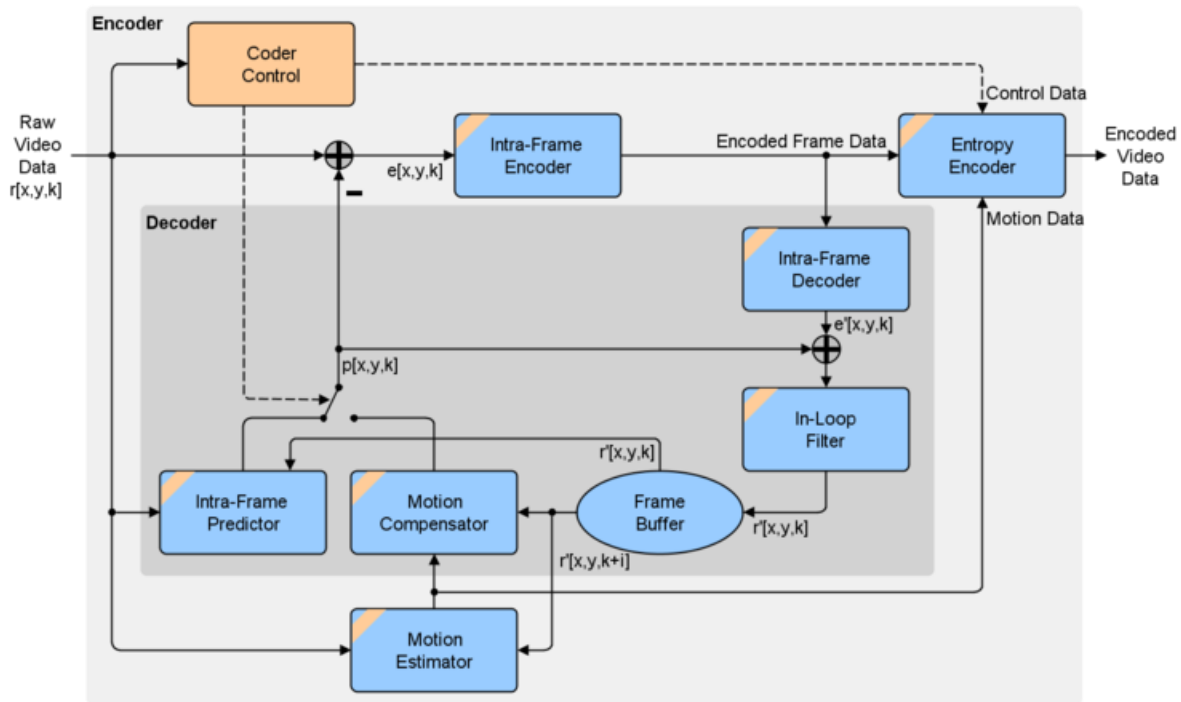
User roles in different industries addressed by this Generic Enabler are:

- 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.).

26.5 Basic Concepts

26.5.1 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:

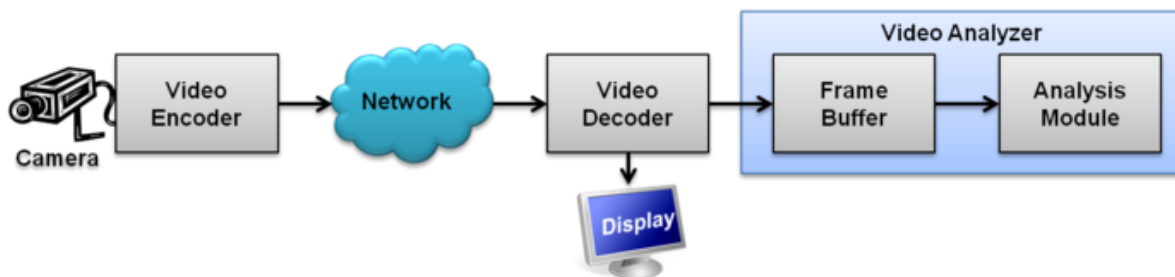
- **Coder Control:** Controls all other components to fulfill pre-defined stream properties, like a certain bit rate or quality. (Indicated by colored block corners)
- **Intra-Frame Encoder:** This component usually performs a transform to the frequency domain, followed by quantization and scaling of 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 modeling 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 neighbouring blocks of the current frame.
- **Entropy Encoder:** The information gathered during the encoding process is entropy encoded in this component. Usually, a resource-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.

26.5.2 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 applied. 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 hardly 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, automated analysis has become 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 easily 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.

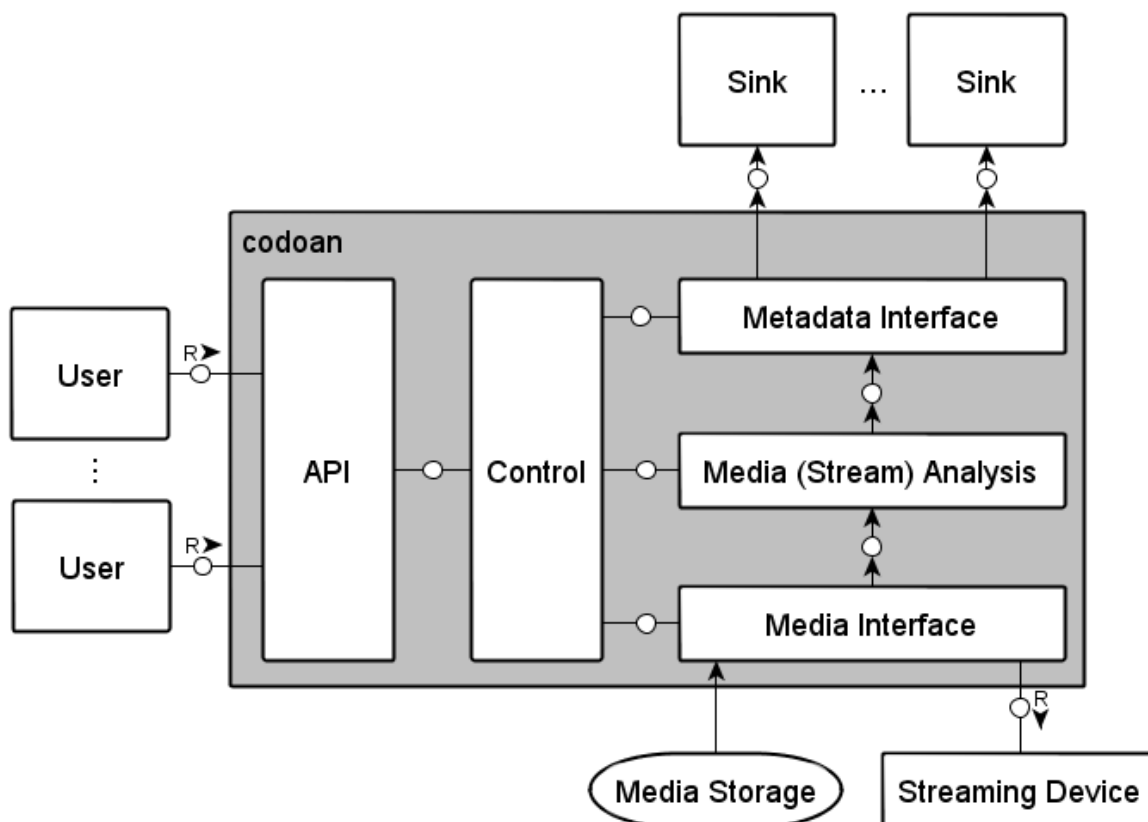


A simple compressed domain processing chain

Due to the omission of the preceding decoder it is possible to work directly with the received data. At the same time, the integrated syntax parser 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 does not always have to be passed through completely. Furthermore, this solution consumes fewer 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.

26.6 Architecture

The Compressed Domain Video Analysis GE consists of a 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 components of the Compressed Domain Video Analysis GE are **Media Interface**, **Media (Stream) Analysis**, **Metadata Interface**, **Control**, and the **API**. They are described in detail in the following subsections. A realization of a Compressed Domain Video Analysis GE consists of a composition of different types of realizations for the five 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 subcomponents). 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.

26.6.1 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 corresponding subcomponent 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 corresponding subcomponent 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 subcomponents of the **Media (Stream) Analysis** component are codec-independent. Subcomponents on a lower abstraction level are able to process H.264/AVC video streams.

26.6.2 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 analysis 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 the following subcomponents 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
- For more details see [\[ODA\]](#)
- **Person Detection**
 - Special case of Moving Object Detection
 - If previous knowledge about the actual objects exists, i.e., the objects are persons, the detection can be further enhanced
- **Object Tracking**
 - Detected objects will be tracked over several subsequent frames
 - Works with preceding Moving Object Detection and Person Detection subcomponents

In principle, the analysis 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 analysis. In this case, real-time processing is in principle not possible and also not intended.

26.6.3 Metadata Interface

The **Metadata Interface** uses a metadata format suitable for subsequent processing. The format could, for instance, be HTTP-based (e.g., RESTful APIs) or XML-based. An XML example is given in section [Main Interactions](#).

The **Media (Stream) Analysis** subcomponent either detects events or moving objects/persons. Therefore, the **Metadata Interface** provides information about detected global changes and moving objects/persons within the analyzed streams. This information is sent to previously registered **Sinks**. **Sinks** can be added by **Users** of the GE by sending corresponding requests to the **API**.

26.6.4 Control

The **Control** component is used to control the aforementioned components of the Compressed Domain Video Analysis GE. Furthermore, it processes requests received via the **API**. Thereby, it creates and handles a separate instance of the GE for each stream to be analyzed.

26.6.5 API

The RESTful **API** defines an interface that enables **Users** of the GE to request several operations using standard HTTP requests. These operations are described in detail in the following section.

26.7 Main Interactions

The API is a RESTful API that permits easy integration with web services or other components requiring analyses of compressed video streams. The following operations are defined:

getVersion

Returns the current version of the CDVA GE implementation.

listInstances

Lists all available instances.

createInstance

Creates a new instance. Thereby, the URI of the compressed video stream and whether events and/or moving objects/persons should be detected (and tracked) have to be provided with this request.

getInstanceInfo

Returns information about a created instance.

destroyInstance

Destroys a previously created and stopped instance.

startInstance

Starts the corresponding instance. This actually starts the analysis.

stopInstance

Stops a previously started instance.

getInstanceConfig

Returns the current configuration of an instance. This includes the configurations of all activated analysis algorithms.

configureInstance

Configures one or more analysis algorithms of the corresponding instance.

listSinks

Lists all registered sinks of the corresponding instance.

addSink

Adds a new sink to an instance. Thereby, an URI has to be provided that enables the GE to notify the sink in case of detections.

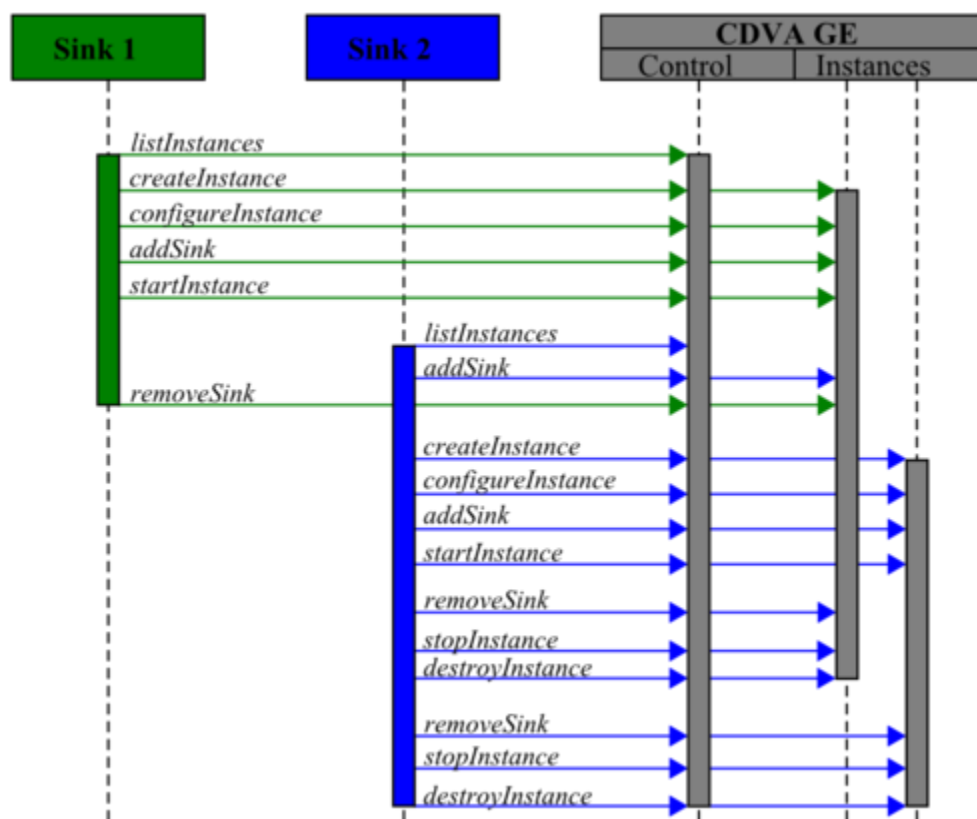
getSinkInfo

Returns information about a previously added sink.

removeSink

Removes a previously added sink from an instance. Once the sink is removed, it will not be notified in case of detections anymore.

The following figure shows an example of a typical usage scenario (two analyzer instances are attached to different media sources). Note that responses and notifications are not shown for reasons of clarity and comprehensibility.



CDVA GE – Usage scenario

First of all, Sink 1 requests a list of all created instances (*listInstances*). As no instance has been created so far, Sink 1 creates (*createInstance*) and configures (*configureInstance*) a new instance for analyzing a specific video stream. To get notified in case of events and/or moving objects/persons, Sink 1 adds itself as a sink to this instance (*addSink*). The actual analysis is finally started by sending a *startInstance* request. During the analysis of the video stream, a second sink, Sink 2, also requests a list of instances (*listInstances*). As Sink 2 is also interested in the results of the analysis Sink 1 previously started, it also adds itself to this instance (*addSink*), just before Sink 1 removes itself from the instance (*removeSink*) to not get notified anymore. Additionally, Sink 2 wants another video stream to be analyzed and therefore creates (*createInstance*) and configures (*configureInstance*) a new instance, adds itself to this instance (*addSink*) and starts the analysis (*startInstance*). While receiving the results of the second analysis, Sink 2 removes itself from the first instance (*removeSink*) and requests to stop the analysis (*stopInstance*)

and to destroy the instance (***destroyInstance***). Note that the instance will only be destroyed if all sinks have been removed. At the end of this scenario, Sink 2 finally removes itself from the second instance (***removeSink***) and also requests to stop the analysis of this instance (***stopInstance***) and to destroy this instance (***destroyInstance***).

Event and object metadata that are sent to registered sinks 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:wsn="http://docs.oasis-open.org/wsn/b-2"
                xmlns:codoan="http://www.fi-
ware.eu/data/cdva/codoan/schema"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.fi-
ware.eu/data/cdva/codoan/schema http://cdvideo.testbed.fi-
ware.eu/codoan/schema/codoan_onvif.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:StreamProperties>

```

```

<cdoan:FrameWidth>352</cdoan:FrameWidth>

<cdoan:FrameHeight>288</cdoan:FrameHeight>

</cdoan:StreamProperties>
</Extension>
</MetadataStream>

```

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

26.8 Basic Design Principles

- Critical product attributes for the Compressed Domain Video Analysis GE are especially high detection rates 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 subcomponents.

26.9 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.

[ODA]	M. Laumer, P. Amon, A. Hutter, and A. Kaup, " Compressed Domain Moving Object Detection Based on H.264/AVC Macroblock Types ", VISAPP 2013, Feb. 2013.
[ONVIF]	ONVIF Specifications
[rtpdump]	rtpdump format specified by RTP Tools

26.10 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.

26.10.1 Open API Specifications

- [Compressed Domain Video Analysis Open RESTful API Specification](#)

26.11 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](#)

26.12 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 **avalue**. 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.

27 Compressed Domain Video Analysis Open RESTful API Specification

27.1 Introduction to the Compressed Domain Video Analysis GE API

Please check the [Legal Notice](#) to understand the rights to use FI-WARE Open Specifications.

27.1.1 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.

27.1.2 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 these specifications, the reader should firstly have a general understanding of the appropriate Generic Enabler supporting the API ([Compressed Domain Video Analysis GE Product Vision](#)).

27.1.3 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	<ul style="list-style-type: none">Adapted to new templateAdded 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

April 26, 2013	Updated XML examples
May 23, 2013	Incorporated review comments
December 10, 2013	<ul style="list-style-type: none"> • Updated API operations for new feature object tracking • Minor corrections
February 25, 2014	<ul style="list-style-type: none"> • Updated XML examples • Added PBM notification

27.1.4 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 [/{sinkNotificationURI}](#)). The analyzed media is received by using RTP, as defined by [RFC3550](#), and RTSP, as defined by [RFC2326](#).

27.1.5 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.

27.2 General Compressed Domain Video Analysis GE API Information

27.2.1 Resources Summary

The resource summary is shown in the following overview.

```

Compressed Domain Video Analysis GE (server)
-----
//{server}/{assetName}
|
|-- /version                                GET -> getVersion
|
|-- /instances
|   |
|   |-- /{instanceID}
|       |
|       |-- ?action=start                  PUT -> startInstance
|       |-- ?action=stop                   PUT -> stopInstance
|       |-- /config                        GET -> getInstanceConfig
|       |-- /config                        PUT -> configureInstance
|       |
|       |-- /sinks
|           |
|           |-- /{sinkID}
|               |
|               |-- GET -> getSinkInfo
|               |-- DELETE -> removeSink

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

```

27.2.2 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.

27.2.3 Resource Identification

The resource identification for HTTP transport is made using the mechanisms described by HTTP protocol specification as defined by [RFC2616](https://tools.ietf.org/html/rfc2616).

27.2.4 Links and References

Request forwarding is not supported in the current version of the Compressed Domain Video Analysis GE.

27.2.5 Limits

Limits are not yet specified for the current version of the Compressed Domain Video Analysis GE.

27.2.6 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
```

27.2.7 Extensions

Querying extensions is not supported in the current version of the Compressed Domain Video Analysis GE.

27.2.8 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]

27.3 API Operations

27.3.1 /version

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

27.3.1.1 **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: 185
Content-Type: application/xml
Server: codoan REST server

<?xml version="1.0" encoding="UTF-8"?>
<Codoan>
  <Version>2.0.0</Version>
  <Copyright>(c) 2010-2014 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.

27.3.2 /instances

Verb	URI	Description
GET	://{server}/{assetName}/instances	listInstances : lists all active instances of the Compressed Domain Video Analysis GE
POST	://{server}/{assetName}/instances	createInstances : creates a new instance of the Compressed Domain Video Analysis GE

27.3.2.1 *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: 407
Content-Type: application/xml
Server: codoan REST server

<?xml version="1.0" encoding="UTF-8"?>
<Codoan>
  <Instances>
    <Instance activeSinks="0" detectEvents="true" detectObjects="true"
trackObjects="true" id="101" isRunning="false"
streamURI="rtsp://192.0.2.2/stream1"/>
    <Instance activeSinks="3" detectEvents="true"
detectObjects="false" trackObjects="false" id="102" isRunning="true"
streamURI="rtsp://192.0.2.8/camera7"/>
  </Instances>
```

```
</Codoan>
```

27.3.2.2 *createInstance*

Sample request:

```
POST //192.0.2.1/codoan/instances HTTP/1.1
Accept: application/xml
Content-Length: 205
Content-Type: application/xml

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

Sample response:

```
HTTP/1.1 201 Created
Content-Length: 248
Content-Type: application/xml
Server: codoan REST server

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

```

    <Instance activeSinks="0" detectEvents="true" detectObjects="true"
trackObjects="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.

27.3.3 `/instances/{instanceID}`

Verb	URI	Description
GET	<code>/{server}/{assetName}/instances/{instanceID}</code>	<i>getInstanceInfo</i> : returns information about an existing instance of the Compressed Domain Video Analysis GE
DELETE	<code>/{server}/{assetName}/instances/{instanceID}</code>	<i>destroyInstance</i> : destroys an existing instance of the Compressed Domain Video Analysis GE
PUT	<code>/{server}/{assetName}/instances/{instanceID}?action=start</code>	<i>startInstance</i> : starts the analysis of an existing instance of the Compressed Domain Video Analysis GE
PUT	<code>/{server}/{assetName}/instances/{instanceID}?action=stop</code>	<i>stopInstance</i> : stops the analysis of an existing instance of the Compressed Domain Video Analysis GE

27.3.3.1 ***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: 248
Content-Type: application/xml
Server: codoan REST server

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

27.3.3.2 *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: 248
Content-Type: application/xml
Server: codoan REST server

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

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

</Instances>

</Codoan>
```

27.3.3.3 *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: 247
Content-Type: application/xml
Server: codoan REST server

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

27.3.3.4 *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: 248
Content-Type: application/xml
Server: codoan REST server

<?xml version="1.0" encoding="UTF-8"?>
<Codoan>
  <Instances>
    <Instance activeSinks="0" detectEvents="true" detectObjects="true"
trackObjects="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.

27.3.4 /config

Verb	URI	Description
GET	<code>://{server}/{assetName}/instances/{instanceID}/config</code>	<i>getInstanceConfig</i> : returns the configuration of an existing instance of the Compressed Domain Video Analysis GE
PUT	<code>://{server}/{assetName}/instances/{instanceID}/config</code>	<i>configureInstance</i> : configures an existing instance of the Compressed Domain Video Analysis GE

The following parameters can be set to configure an instance:

- Event (Change) Detection
 - **NumberOfTrainingFrames**
 - The number of initial frames that should be used to train the algorithm
 - Default: 4 * SlidingWindowSize
 - Type: Positive integer
 - **SlidingWindowSize**
 - The size (in number of frames) of two sliding windows to calculate ANORP and ARPS factors
 - Default: 10
 - Type: Positive integer
 - **ThresholdANORPFactor**
 - Calculated ANORP factors are compared to this threshold
 - Default: 1.2
 - Type: Non-negative decimal
 - **ThresholdARPSFactor**
 - Calculated ARPS factors are compared to this threshold
 - Default: 1.75
 - Type: Non-negative decimal
 - **ThresholdIframe**
 - Threshold for detecting I-frames within a video stream
 - Default: 5
 - Type: Non-negative decimal
- Moving Object/Person Detection
 - **type**
 - The type of objects that will be detected
 - Default: "Other"
 - Type: "Other" or "Person"

- **BoxFilterSize**
 - Size of a post-processing box filter applied to blocks of pixels
 - Default: 3
 - Type: Odd positive integer
- **ThresholdH264MOC**
 - Threshold for detecting moving blocks within frames
 - Default: 6
 - Type: Positive integer
- **NumberOfPreviousFrames**
 - The number of previous frames to be considered during temporal filtering
 - Default: 3
 - Type: Non-negative integer
- **NumberOfSubsequentFrames**
 - The number of subsequent frames to be considered during temporal filtering
 - Default: 3
 - Type: Non-negative integer
- **DistanceWeight**
 - Weight for neighboring frames during temporal filtering
 - Default: 1.0
 - Type: Non-negative decimal
- **FilterStrength**
 - The strength of the temporal filtering process
 - Default: 0.5
 - Type: Non-negative decimal less than 1.0
- Object Tracking
 - **UseMovingObjectHistory**
 - If set to true, the analyzer stores detected objects in an object history, which is then used for tracking objects. If set to false, objects are tracked by considering previous frames only.

- Default: true
- Type: Boolean
- **BoundingBoxOverlapLimit**
 - The lower limit for a positive match of two bounding boxes
 - Default: 0.25
 - Type: Non-negative decimal greater than 0.0 and less than or equal 1.0 (indicates percent)

For a more detailed description, please refer to the respective reference in the [Compressed Domain Video Analysis Open Specification](#).

27.3.4.1 *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: 1344
Content-Type: application/xml
Server: codoan REST server

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

```
<EventDetection type="GlobalChange">
  <NumberOfTrainingFrames>40</NumberOfTrainingFrames>
  <SlidingWindowSize>10</SlidingWindowSize>
  <ThresholdANORPFactor>1.2</ThresholdANORPFactor>
  <ThresholdARPSFactor>1.75</ThresholdARPSFactor>
  <ThresholdIFrame>5</ThresholdIFrame>
</EventDetection>
<ObjectDetection type="Person">
  <ThresholdH264MOC>6</ThresholdH264MOC>
  <BoxFilterSize>3</BoxFilterSize>
  <TemporalFilter>
    <NumberOfPreviousFrames>3</NumberOfPreviousFrames>
    <NumberOfSubsequentFrames>3</NumberOfSubsequentFrames>
    <DistanceWeight>1.0</DistanceWeight>
    <FilterStrength>0.5</FilterStrength>
  </TemporalFilter>
</ObjectDetection>
<ObjectTracking>
  <UseMovingObjectHistory>true</UseMovingObjectHistory>
  <BoundingBoxOverlapLimit>0.25</BoundingBoxOverlapLimit>
</ObjectTracking>
</Configuration>
</Instance>
</Instances>
</Codoan>
```

27.3.4.2 *configureInstance*

Sample request:

```
PUT //192.0.2.1/codoan/instances/101/config HTTP/1.1
Accept: application/xml
Content-Length: 946
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<Codoan>
  <Instances>
    <Instance      detectEvents="false"      detectObjects="true"
trackObjects="true" streamURI="rtsp://192.0.2.2/stream1">
      <Configuration>
        <ObjectDetection type="Person">
          <ThresholdH264MOC>6</ThresholdH264MOC>
          <BoxFilterSize>3</BoxFilterSize>
          <TemporalFilter>
            <NumberOfPreviousFrames>3</NumberOfPreviousFrames>
            <NumberOfSubsequentFrames>3</NumberOfSubsequentFrames>
            <DistanceWeight>1.0</DistanceWeight>
            <FilterStrength>0.5</FilterStrength>
          </TemporalFilter>
        </ObjectDetection>
        <ObjectTracking>
          <UseMovingObjectHistory>true</UseMovingObjectHistory>
          <BoundingBoxOverlapLimit>0.25</BoundingBoxOverlapLimit>
        </ObjectTracking>
      </Configuration>
    </Instance>
  </Instances>
</Codoan>
```



```
        </ObjectTracking>

    </Configuration>

</Instance>

</Instances>

</Codoan>
```

Sample response:

```
HTTP/1.1 200 OK
Content-Length: 989
Content-Type: application/xml
Server: codoan REST server

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

    <Instances>

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

            <Configuration>

                <ObjectDetection type="Person">

                    <ThresholdH264MOC>6</ThresholdH264MOC>

                    <BoxFilterSize>3</BoxFilterSize>

                    <TemporalFilter>

                        <NumberOfPreviousFrames>3</NumberOfPreviousFrames>

                        <NumberOfSubsequentFrames>3</NumberOfSubsequentFrames>

                        <DistanceWeight>1.0</DistanceWeight>

                        <FilterStrength>0.5</FilterStrength>
```

```
</TemporalFilter>

</ObjectDetection>

<ObjectTracking>

  <UseMovingObjectHistory>true</UseMovingObjectHistory>

  <BoundingBoxOverlapLimit>0.25</BoundingBoxOverlapLimit>

</ObjectTracking>

</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.

27.3.5 /sinks

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

27.3.5.1 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: 280
Content-Type: application/xml
Server: codoan REST server

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

27.3.5.2 *addSink*

Sample request:

```
POST //192.0.2.1/codoan/instances/101/sinks HTTP/1.1
Accept: application/xml
Content-Length: 329
Content-Type: application/xml

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

```

    <Instance          detectEvents="true"          detectObjects="true"
trackObjects="true" streamURI="rtsp://192.0.2.2/stream1">

        <Sinks>

            <Sink
sinkNotificationURI="http://192.0.2.3/notification/stream1"
sendXml="true" sendBinaryMap="true"/>

        </Sinks>

    </Instance>

</Instances>

</Codoan>

```

Sample response:

```

HTTP/1.1 201 Created
Content-Length: 381
Content-Type: application/xml
Server: codoan REST server

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

    <Instances>

        <Instance activeSinks="1" detectEvents="true" detectObjects="true"
trackObjects="true"          id="101"          isRunning="false"
streamURI="rtsp://192.0.2.2/stream1">

            <Sinks>

                <Sink                                     id="201"
sinkNotificationURI="http://192.0.2.3/notification/stream1"
sendXml="true" sendBinaryMap="true"/>

            </Sinks>

        </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.

27.3.6 /{sinkID}

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

27.3.6.1 *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: 381
Content-Type: application/xml
Server: codoan REST server

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

  <Instances>
```

```

    <Instance activeSinks="1" detectEvents="true" detectObjects="true"
trackObjects="true"          id="101"          isRunning="false"
streamURI="rtsp://192.0.2.2/stream1">

        <Sinks>

            <Sink                                id="201"
sinkNotificationURI="http://192.0.2.3/notification/stream1"
sendXml="true" sendBinaryMap="true"/>

        </Sinks>

    </Instance>

</Instances>

</Codoan>

```

27.3.6.2 *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: 381
Content-Type: application/xml
Server: codoan REST server

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

    <Instances>

```

```

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

        <Sinks>

            <Sink                                id="201"
sinkNotificationURI="http://192.0.2.3/notification/stream1"
sendXml="true" sendBinaryMap="true"/>

        </Sinks>

    </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.

27.3.7 //{sinkNotificationURI}

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

27.3.7.1 **notifySink**

The notification is sent via a POST request. The data in its body either represents an XML structure or a Portable Bitmap (PBM). Thereby, the parameters **sendXml** and **sendBinaryMap**, which are set in the [addSink](#) operation, indicate if an XML and/or a PBM structure should be sent, respectively.

XML sample request:

```

POST //192.0.2.3/notification/stream1 HTTP/1.1

User-Agent: codoan REST client

Content-Length: 1795

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/cdva/codoan/schema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.fi-ware.eu/data/cdva/codoan/schema http://cdvideo.testbed.fi-ware.eu/codoan/schema/codoan_onvif.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>

```

PBM sample request:

```
POST //192.0.2.3/notification/stream1 HTTP/1.1
User-Agent: codoan REST client
Content-Length: 12761
Content-Type: image/x-portable-bitmap

P4
# created by Codoan
# analyzed media: rtsp://192.0.2.2/stream1
# frame number: 100
352 288
<binary data>
```

Assumed response:

```
HTTP/1.1 200 OK
```

28 FIWARE OpenSpecification Data QueryBroker

Name	FIWARE.OpenSpecification.Data.QueryBroker		
Chapter	Data/Context Management,		
Catalogue-Link Implementation	to QueryBroker	Owner	Siemens AG, Thomas Riegel

28.1 Preface

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

28.2 Copyright

- Copyright © 2012 by [Siemens AG](#)

28.3 Legal Notice

Please check the following [Legal Notice](#) to understand the rights to use these specifications.

28.4 Overview

28.4.1 Introduction to the Media-enhanced Query Broker GE

Today data - and especially in the media domain - 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 Query Format (MPQF)) [\[Döller 08a\]](#), 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 data is performed inside the connected data stores.

28.4.2 Target usage

The Media-enhanced Query Broker GE provides a smart, abstracting interface for retrieval of data from the FI-WARE data management layer. This is provided in addition to the publish/subscribe interface (e.g. [Context Broker \(Publish/Subscribe Broker\) GE](#)) 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 characteristics of the internal data storage and DB implementations and interfaces.

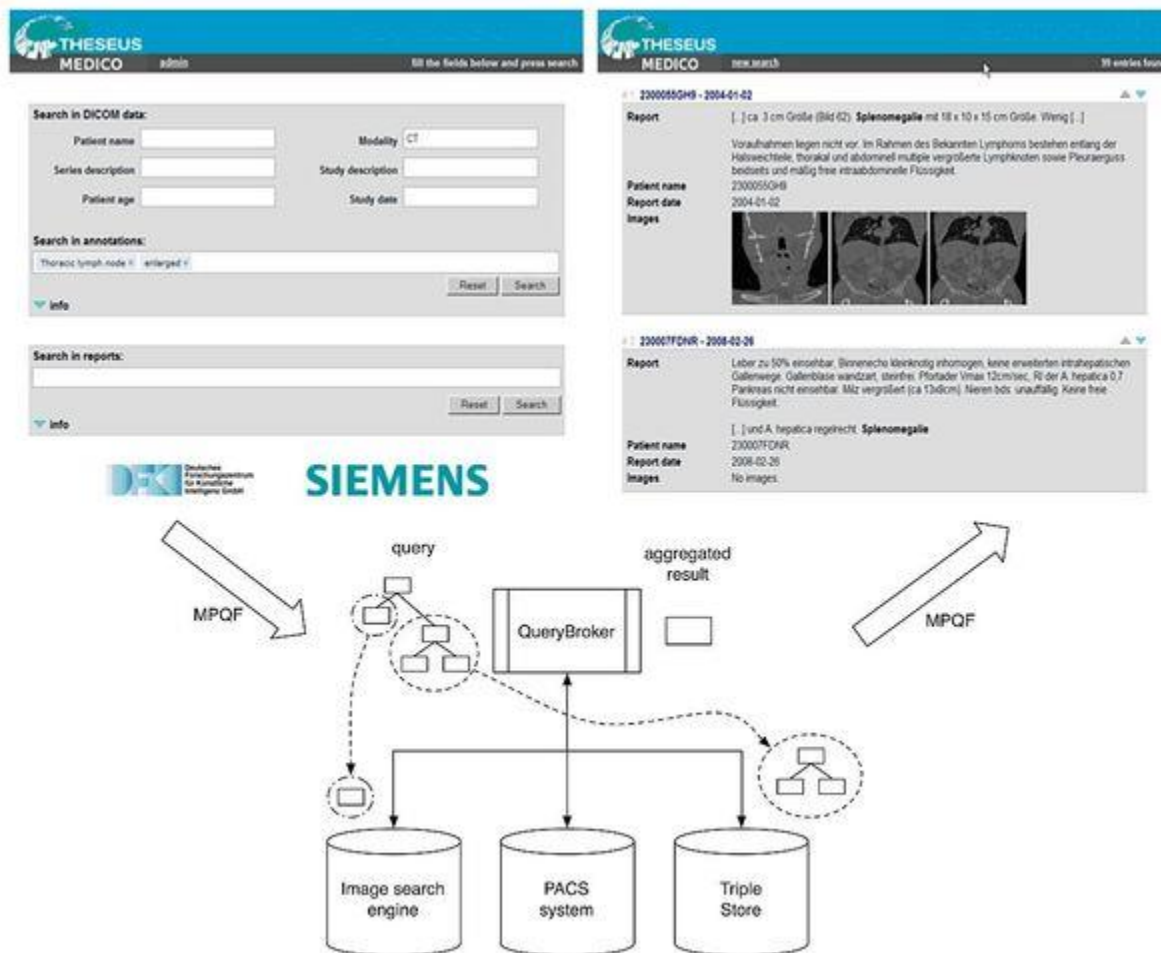
Therefore, this GE provides support for integration of query-functions into the users' applications by abstracting the access to databases and search engines available in the FI-WARE data management platform while also offering the option to simultaneously access outside data sources. At the same time 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, 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).

28.4.3 Example Scenario

To illustrate that the Media-enhanced Query Broker GE is not stuck to the media domain, but can contribute positively in other application fields too, an example from the medical domain is given:

Typically, in the current diagnostic process at hospitals the already identified issues of heterogeneity can be also found. 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

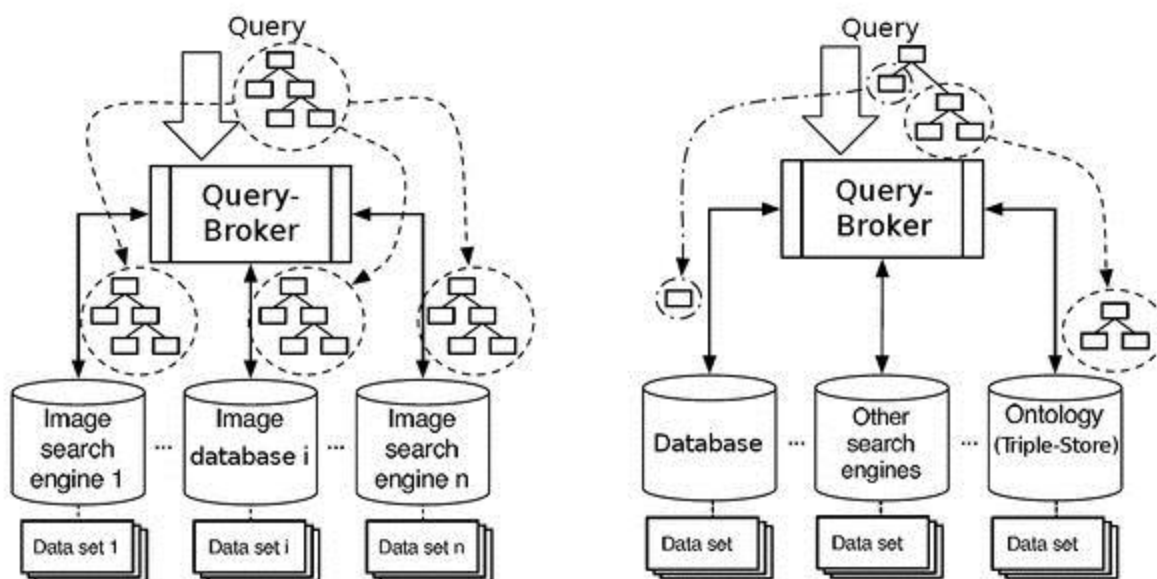
The figure above sketches an end-to-end workflow inside the MEDICO system. It provides the user with an easy-to-use web-based form to describe the desired search query. Currently, this user interface utilizes a semantically rich data set composed of [DICOM](#) tags, image annotations, text annotations 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 examinations 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. Obviously, 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 following the undermentioned design principles. An overview of the architecture can be found in [\[Stegmaier 10\]](#) and [\[Stegmaier 09b\]](#).

28.5 Basic Concepts

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

28.5.1 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.



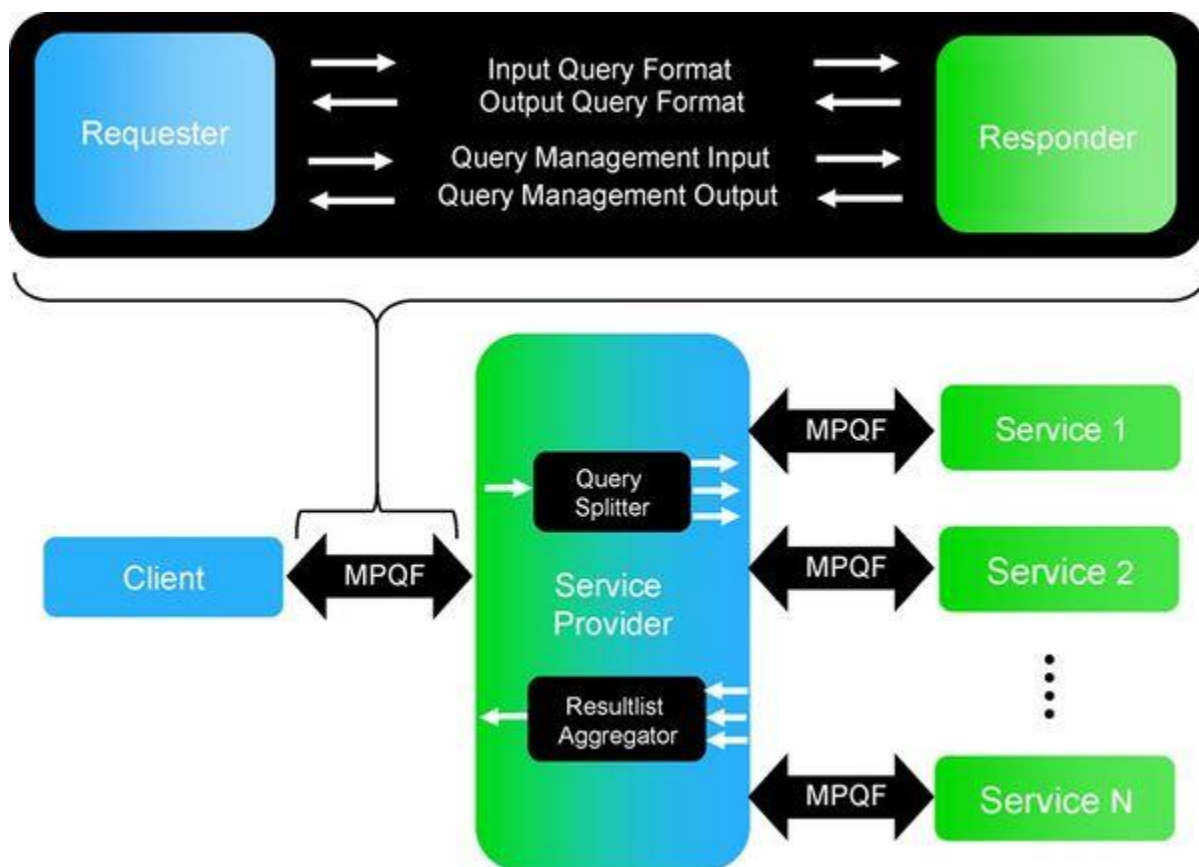
(a) Local/autonomous processing (b) Distributed processing
Query processing strategies

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.

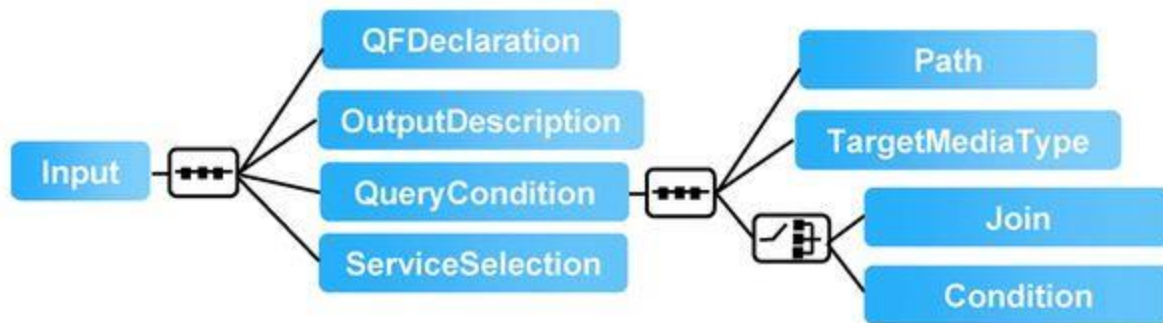
28.5.2 MPEG Query Format (MPQF)

Before discussing the design and the implementation of the Media-enhanced Query Broker in more detail, the main features of MPQF will be introduced as it is used for representing the queries. 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

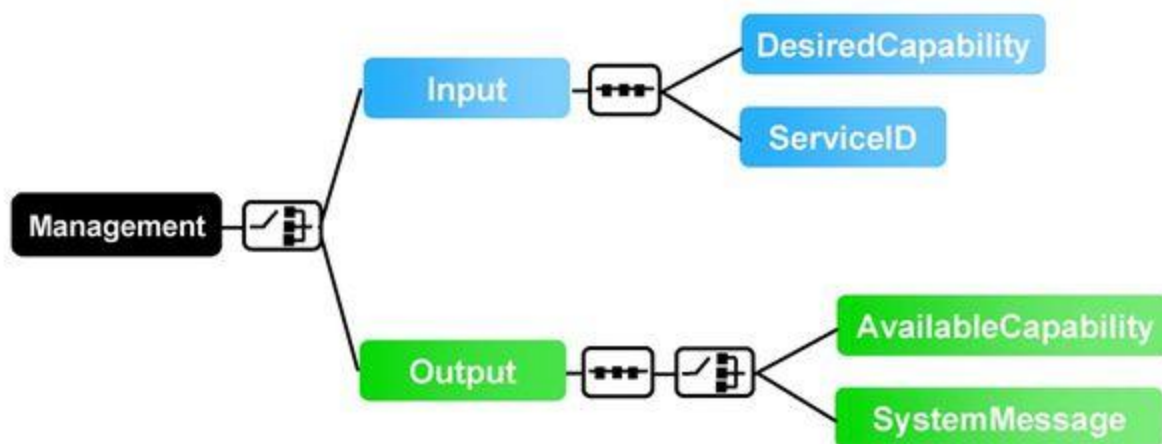
In detail, the IQF (see the figure above) can be composed of three different parts. The first is a declaration part pointing to resources (e.g., image file or its metadata description, etc.) that are used within the query condition or output description part. The output description part allows, by using 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 signaling paging and expiration dates.

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

<i>SpatialQuery</i>	Retrieval of spatial elements within media objects
<i>TemporalQuery</i>	Retrieval of temporal elements within media objects (e.g., a scene in a video)
<i>QueryByXQuery</i>	Container for limited XQuery expressions
<i>QueryByRelevanceFeedback</i>	Retrieval that takes result items of a previous search into account Free text retrieval
<i>QueryByROI</i>	Retrieval based on a certain region of interest
<i>QueryBySPARQL</i>	Container for limited SPARQL expressions (a SPARQL expression that operate on a single triple is used to filter information).

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 copes 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).

28.5.3 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.

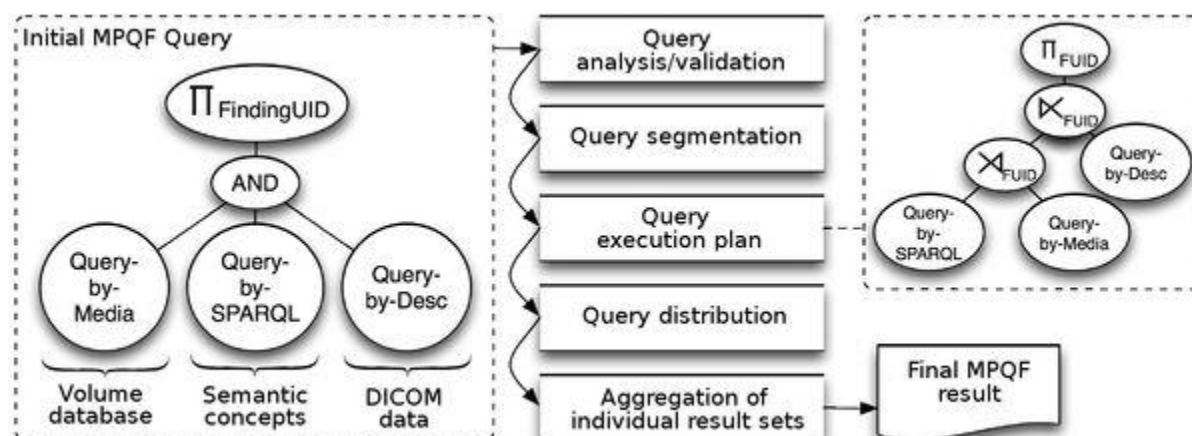
- Distribution of query

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.

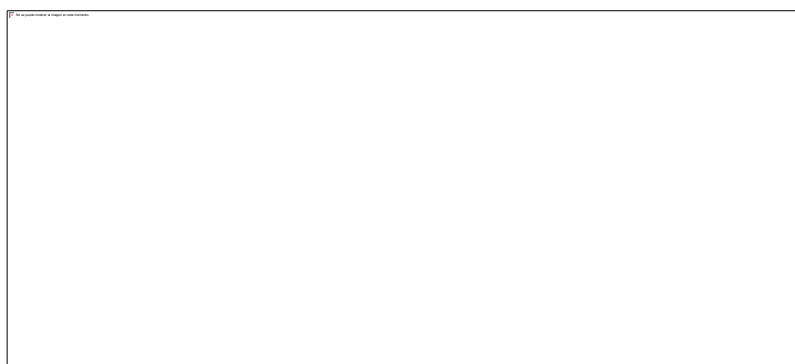
The described workflow of the federated query processing can best be illustrated using the example scenario, as depicted in the figure below.



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.

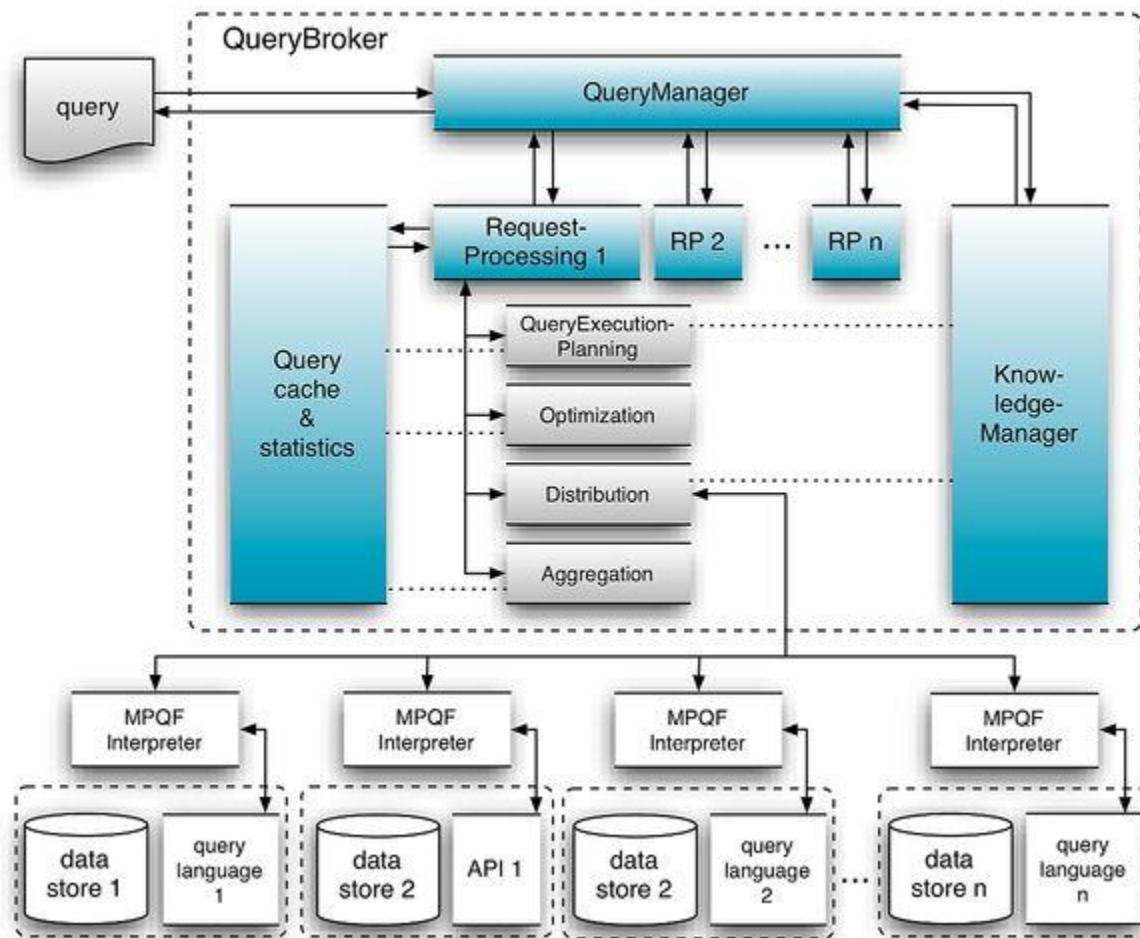


Semantic Link between knowledge bases

28.6 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 [thereafter](#) 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 built 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 for easing the concurrent query processing: 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.

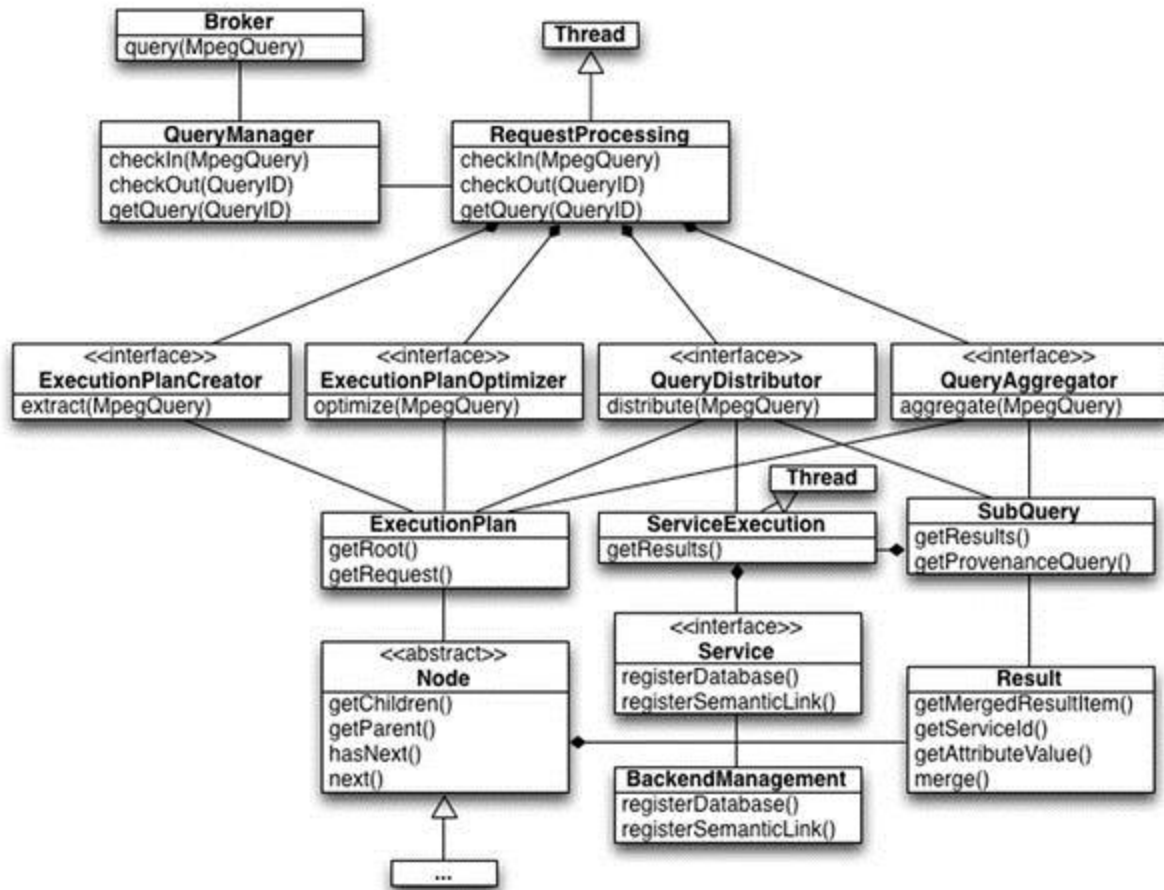
28.7 Main Interactions

28.7.1 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 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).

28.7.2 Architecture

The following figure shows an overview over the QueryBroker software architecture. Only the key elements are listed below for getting a quick impression how the elements are related.



QueryBroker software architecture

- **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 these two elements and process the queried results.
- **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.

28.7.3 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.

28.7.3.1 (De-)Register a Service

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.

28.7.3.2 Creating a Semantic Link

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.

28.7.4 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.

28.7.4.1 Querying

The QueryBroker uses the MPEQ Query Format ([MPQE](#)) 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.

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>
```

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.

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>

```

28.7.4.2 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.

There are blocking, half blocking and none blocking operators. A blocking operator needs all results from its children to decide which result will be 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.

A detailed description on how to access the software modules and interfaces of the QueryBroker is provided in the [User and Programmer Guide](#) of the Query Broker. It explains the necessary steps to integrate the QueryBroker into another application and how to access its actual backend and frontend functionalities. Additionally a code example is given, which shows an example implementation of all required steps to initialize and run the QueryBroker.

28.8 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 loosely coupled components in the overall system supporting an easy extension of the provided functionality of components, or even the replacement of these by new implementations.

28.9 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=34228
[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 Semantic 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/

28.10 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.

28.10.1 Open API Specifications

- [Query Broker Open RESTful API Specification](#)

28.11 Re-utilised Technologies/Specifications

- At its core, the QueryBroker utilizes the MPEG Query Format (MPQF) [ISO/IEC 15938-12:2012] 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 15938-12:2012] "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, http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=61195.

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, <http://doi.ieeecomputersociety.org/10.1109/MMUL.2008.96>.

[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.

28.12 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 be 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 two 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.

29 Query Broker Open RESTful API Specification

29.1 Introduction to the REST-Interface of the QueryBroker

Please check the [Legal Notice](#) to understand the rights to use FI-WARE Open Specifications.

29.1.1 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.

29.1.2 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]).

29.1.3 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	<ul style="list-style-type: none">• initial version
July 4, 2012	<ul style="list-style-type: none">• version 0.5
Oct 4, 2012	<ul style="list-style-type: none">• version 0.6

Jan 25, 2013	<ul style="list-style-type: none">• version 0.7
Apr 16, 2013	<ul style="list-style-type: none">• version 0.8
June 26, 2013	<ul style="list-style-type: none">• version 0.9
Sept 13, 2013	<ul style="list-style-type: none">• version 1.0
...	<ul style="list-style-type: none">• ...

29.1.4 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](#).

29.1.5 Additional Resources

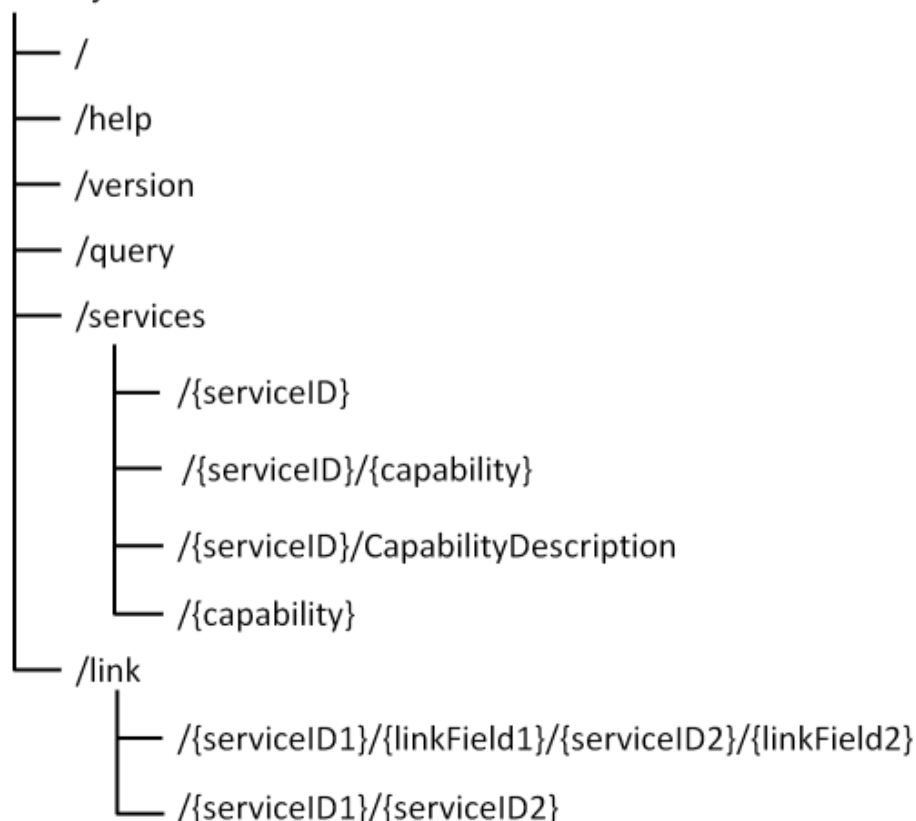
You can download the most current version of this document from the FIWARE API specification website at the [Summary of FI-WARE Open Specifications](#). For more details about the Media-enhanced Query Broker GE that this API is based upon, please refer to "[Architecture Description of Media-enhanced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

29.2 General QueryBroker REST API Information

29.2.1 Resources Summary

QueryBroker Component

//{serverRoot}/QueryBrokerServer



29.2.2 Authentication

Authentication is currently NOT supported - this feature may be incorporated in the future employing the [Access Control GE](#).

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).

Apart from that specific authentication credentials may be required for accessing the registered services (data repositories). These authentication data need to be handled by the service interface, which has to be implemented by any service endpoint and installed at the QueryBroker.

29.2.3 Representation Format

The QueryBroker REST API supports XML based representation formats for both requests and responses, namely MPQF. This is specified by setting the Content-Type header to *multipart/form-data*, and is required for operations that have a request body. The response format is in the current version MPQF.

29.2.4 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 has to be used to specify the MIME type of the wrapped representation.

29.2.5 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.

29.2.6 Links and References

None

29.2.7 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.

29.2.8 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 by the registered data repositories may exhaust the available RAM (min. 1 GB, but 4 GB preferred) causing an fault.

29.2.9 Versions

The current version of the used implementation of the Query Broker GE can be requested by the following HTTP request:

```
GET http://{ServerRoot}/QueryBrokerServer/version HTTP/1.1
```

29.2.10 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]

29.3 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:

In order to be able to register and access data repositories "data base connectors" or service interfaces need to be implemented. The fully qualified class name (e.g., de.uop.dimis.test.Service) of the implemented service is used as serviceID. Hence, all services which will be registered at the QueryBroker have to be in the classpath of the QueryBroker-Server WAR-file.

A description on how to realize such a service interface is given in [QueryBroker GE Installation and Administration Guide](#).

29.3.1 QueryBroker operations

29.3.1.1 *Welcome page of QueryBroker*

Verb	URI	Description
GET	<code>://{serverRoot}/QueryBrokerServer/</code>	Returns a welcome page in html format.

Request example:

```
GET //localhost/QueryBrokerServer/ HTTP/1.1
Host: localhost:8080
Accept: */*
```

Response example:

```
HTTP/1.1 200 OK
```

Content-Length: 8798

Content-Type: text/xml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"><html ... </html>
```

29.3.1.2 *Submit MPQF query*

Verb	URI	Description
POST	<code>://{serverRoot}/QueryBrokerServer/query</code>	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
Host: localhost:8080
Accept: */*
Content-Type: multipart/form-data
Content-Length: 864

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

<ns4:mpegQueryType xmlns:ns2="JPSearch:schema:coremetadata"
xmlns:ns4="urn:mpeg:mpqf:schema:2008"
xmlns:ns3="urn:medico:dicom:schema:2011" mpqfID="quasia_2">

  <ns4:Query>

    <ns4:Input immediateResponse="true">

      <ns4:OutputDescription maxItemCount="32"/>

      <ns4:QueryCondition>

        <ns4:TargetMediaType>image/jpeg</ns4:TargetMediaType>

        <ns4:Condition
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```

xsi:type="ns4:QueryByDescription"                                matchType="similar"
preferenceValue="1.0">
    <ns4:DescriptionResource resourceID="resource_4">
        <ns4:AnyDescription>
            <ns2:ImageDescription>
                <ns2:Keyword>water</ns2:Keyword>
            </ns2:ImageDescription>
        </ns4:AnyDescription>
    </ns4:DescriptionResource>
</ns4:Condition>
</ns4:QueryCondition>
</ns4:Input>
</ns4:Query>
</ns4:mpegQueryType>

```

Response example:

```

HTTP/1.1 200 OK
Content-Length: 17089
Content-Type: text/plain;charset=ISO-8859-1

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<mpegQueryType mpqfID="10a38241-3f7b-4540-9152-669669267013"
xmlns:ns2="JPSearch:schema:coremetadata"
xmlns="urn:mpeg:mpqf:schema:2008"
xmlns:ns3="urn:medico:dicom:schema:2011">
    <Query>
        <Input immediateResponse="true">
            <OutputDescription maxItemCount="32"/>

```

```

        <QueryCondition>

            <TargetMediaType>image/jpeg</TargetMediaType>

            <Condition                                xsi:type="QueryByDescription"
matchType="similar"                                preferenceValue="1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

                <DescriptionResource resourceID="resource_4">

                    <AnyDescription>

                        <ns2:ImageDescription>

                            <ns2:Keyword>water</ns2:Keyword>

                        </ns2:ImageDescription>
                    </AnyDescription>

                </DescriptionResource>

            </Condition>

        </QueryCondition>

    </Input>

    <Output>

        <ResultItem    confidence="1.0"    originID="www.flickr.com"
recordNumber="1">

            <Thumbnail
fromREF="www.flickr.com">http://farm9.static.flickr.com/8406/861857702
5_6f2f98b87f_t.jpg</Thumbnail>

            <MediaResource
fromREF="www.flickr.com">http://farm9.static.flickr.com/8406/861857702
5_6f2f98b87f.jpg</MediaResource>

            <Description>

                <ns2:jpSearchCoreType>

<ns2:Identifier>http://farm9.static.flickr.com/8406/8618577025_6f2f98b
87f.jpg</ns2:Identifier>

                <ns2:Creators>

```

```

<ns2:GivenName>hannabergstrom</ns2:GivenName>

        </ns2:Creators>

        <ns2:Publisher>

                <ns2:OrganizationInformation>

                        <ns2:Name>www.flickr.com</ns2:Name>

                </ns2:OrganizationInformation>

        </ns2:Publisher>

        <ns2:CreationDate>2013-03-
06T23:36:16.000+01:00</ns2:CreationDate>

        <ns2:Description></ns2:Description>

        <ns2:RightsDescription>

<ns2:RightsDescriptionInformation>unknown</ns2:RightsDescriptionInform
ation>

                <ns2:Description>All                                Rights
Reserved</ns2:Description>

<ns2:ActualRightsDescriptionReference>unknown</ns2:ActualRightsDescrip
tionReference>

                </ns2:RightsDescription>

                <ns2:Source>

<ns2:SourceElementType>unknown</ns2:SourceElementType>

                <ns2:SourceElement>

<ns2:SourceElementTitle>unknown</ns2:SourceElementTitle>

<ns2:SourceElementDescription>unknown</ns2:SourceElementDescription>

<ns2:SourceElementIdentifier>unknown</ns2:SourceElementIdentifier>

```

```

</ns2:SourceElement>

<ns2:CreationMethod>unknown</ns2:CreationMethod>

<ns2:CreationDescription>www.flickr.com/65051422@N08</ns2:CreationDescription>

    </ns2:Source>

    <ns2:Keyword>photograpgy</ns2:Keyword>

    <ns2:Keyword>blackandwhite</ns2:Keyword>

    <ns2:Keyword>summer</ns2:Keyword>

    <ns2:Keyword>water</ns2:Keyword>

    <ns2:Keyword>analog</ns2:Keyword>

    <ns2:Title>Family, Gotland</ns2:Title>

    <ns2:Width>100</ns2:Width>

    <ns2:Height>100</ns2:Height>

    </ns2:jpSearchCoreType>

</Description>

</ResultItem>

:

:

<SystemMessage>

    <Status>

        <Code>1</Code>

        <Description>Query was successful</Description>

    </Status>

</SystemMessage>

</Output>

```



```
</Query>
</mpegQueryType>
```

29.3.1.3 (De-)Register Database

Verb	URI	Description
POST	://{serverRoot}/QueryBrokerServer/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.)
POST	://{serverRoot}/QueryBrokerServer/services/{serviceID}/CapabilityDescription	Registers a 'SQL'-service at the QueryBroker endpoint. Its capability description has to be provided as valid MPQF-query in the body and a configuration file containing credential data for accessing the 'SQL'-service with the name <i>{serviceID}.properties</i> is necessary for that class (available with Release 2).
DELETE	://{serverRoot}/QueryBrokerServer/services/{serviceID}/	Deregister the service at the QueryBroker e.g., /services/de.uop.dimis.FlickrService/
GET	://{serverRoot}/QueryBrokerServer/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.services.FlickrService/QueryByMedia HTTP/1.1

Host: localhost:8080

Content-Type:multipart/form-data
```

```
Accept: */*
```

Response example:

```
HTTP/1.1 200 OK

Content-Length: 129

Content-Type: text/plain;charset=ISO-8859-1

Service "de.uop.dimis.services.FlickrService" registered for capability
QueryByMedia ("urn:mpeg:mpqf:2008:CS:full:100.3.6.1 ").
```

29.3.1.4 *Creating a Semantic Link*

Verb	URI	Description
POST	<code>//{serverRoot}/QueryBrokerServer/link/{serviceID1}/{linkField1}/{serviceID2}/{linkField2}</code>	Registers a semantic link between two registered endpoints.
GET	<code>//{serverRoot}/QueryBrokerServer/link/{serviceID1}/{serviceID2}</code>	Returns information about the registered semantic link between the given services.

Request example:

```
POST
//localhost/QueryBrokerServer/link/de.uop.dimis.FlickrService/identifier/de.uop.dimis.GoogleService/description/ HTTP/1.1

Host: localhost:8080

Content-Type: multipart/form-data

Accept: */*
```

Response example:

```
HTTP/1.1 200 OK

Content-Length: 136

Content-Type: text/plain;charset=ISO-8859-1
```

de.uop.dimis.FlickrService:identifier and
de.uop.dimis.GoogleService:description are now semantically connected

29.3.1.5 *Report version of QueryBroker*

Verb	URI	Description
GET	://{serverRoot}/QueryBrokerServer/version	Returns the version number and in brackets the version info of the underlying components as a string.

Request example:

```
GET //localhost/QueryBrokerServer/version HTTP/1.1
Host: localhost:8080
Accept: */*
```

Response example:

```
HTTP/1.1 200 OK
Content-Length: 49
Content-Type: text/plain; charset=ISO-8859-1

Release 2.3 (QB: 2.5.8, QB-A: 0.7.1, QB-S: 1.0.0)
```

29.3.1.6 *Return a concise listing of the RESTful commands*

Verb	URI	Description
GET	://{serverRoot}/QueryBrokerServer/help	Returns a concise listing of the RESTful commands for controlling the QueryBroker.

Request example:

```
GET //localhost/QueryBrokerServer/help HTTP/1.1
Host: localhost:8080
Accept: */*
```

Response example:

```
HTTP/1.1 200 OK
Content-Length: 2093
Content-Type: text/plain;charset=ISO-8859-1

Supported commands (always use "Content-Type: multipart/form-data" as
header information):

-- registering and deregistering services:

GET:  http://localhost:8080/QueryBrokerServer/services/{capability}  --
Returns all service adapters that support the given capability.

POST:
http://localhost:8080/QueryBrokerServer/services/{service}/{capability
} -- Registers the given service with the given capability.

POST:
http://localhost:8080/QueryBrokerServer/services/{service}/CapabilityD
escription -- Registers the given service with all capabilities that
are supplied in the MPQF XML string inside the request body.

DELETE:  http://localhost:8080/QueryBrokerServer/services/{service}  --
Removes the given service from the list of registered services with
all his capabilities.

-- managing semantic links between services:

POST:
http://localhost:8080/QueryBrokerServer/link/{serviceid1}/{link1}/{ser
```

```
viceid2}/{link2} -- Creates a semantic link between the two services
(which both must be registered beforehand).

GET:
http://localhost:8080/QueryBrokerServer/link/{serviceid1}/{serviceid2}
-- Displays the semantic link between the two services (which both
must be registered beforehand).

-- querying services:

POST: http://localhost:8080/QueryBrokerServer/query -- Sends the query
that is supplied in the MPQF XML string inside the request body.

-- miscellaneous information:

GET: http://localhost:8080/QueryBrokerServer/ -- Displays a welcome
message.

GET: http://localhost:8080/QueryBrokerServer/help -- Displays this
help message.

GET: http://localhost:8080/QueryBrokerServer/version -- Returns the
version numbers of the Querybroker, Querybroker-Adapters and
Querybroker-Server.

-- further reading:

A complete overview for this service can be found at: http://forge.fiware.eu/plugins/mediawiki/wiki/fiware/index.php/Query\_Broker\_Open\_RESTful\_API\_Specification
```

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

30.1.1 General Information

["FI-WARE Project Partners"](#) refer to Parties of the FI-WARE Project in accordance with the terms of the FI-WARE Consortium Agreement.

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

30.1.2 Use Of Specification - Terms, Conditions & Notices

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

30.1.3 Licenses

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

30.1.4 Patent Information

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

30.1.5 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 Holders.

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

30.1.6 Disclaimer Of Warranty

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

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

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

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

30.1.7 Trademarks

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

30.1.8 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.org>.

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

31 FI-WARE Open Specification Legal Notice (essential patents license)

31.1.1 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.

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

31.1.2 Use Of Specification - Terms, Conditions & Notices

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

31.1.3 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 royalty free license (without the right to sublicense) under its respective copyrights incorporated in the Specification, to copy and modify this Specification and to distribute copies of the modified version, and to use this Specification, to create and distribute special purpose specifications and software that is an implementation of this Specification.

31.1.4 Patent License

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

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

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

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

31.1.5 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 Holders.

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

31.1.6 Disclaimer Of Warranty

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

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

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

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

31.1.7 Trademarks

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

31.1.8 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.org>.

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