Private Public Partnership Project (PPP)  
Large-scale Integrated Project (IP)

D.6.4.2: FI-WARE User and Programmers Guide

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.6.4.2  
Project Document Date: 2012-04-30  
Deliverable Type and Security: Public  
Author: FI-WARE Consortium  
Contributors: FI-WARE Consortium
1.1 Executive Summary

This document contains the User and Programmers guide of the Data/Context Management technical chapter in FI-WARE. The purpose of this manual is twofold:

- Provide the users with insights in the functionality of each GE and how to use it
- Provide the developers with the right pointers to develop using the Open Specifications

1.2 About This Document

This document comes along with the Software implementation of components, each release of the document being referred to the corresponding Software release (as per D.x.2), to provide documentation of the features offered by the components and interfaces to users/adopters. Moreover, it explains the way they can be exploited in their developments.

1.3 Intended Audience

The document targets users as well as programmers of FI-WARE Generic Enablers.

1.4 Chapter Context

FI-WARE will enable smarter, more customized/personalized and context-aware applications and services by the means of a set of assets able to gather, exchange, process and analyze massive data in a fast and efficient way. Nowadays, several well-known free Internet services are based on business models that exploit massive data provided by end users. This data is exploited in advertising or offered to 3rd parties so that they can build innovative applications. Twitter, Facebook, Amazon, Google and many others are examples of this.

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

FI-WARE Data/Context Management GEs enables to:

- Record, subscribe for being notified about and query for context information coming from different sources.
- Model changes in context as events that can be processed to detect complex situations that will lead to generation of actions or the generation of new context information (therefore, also treatable as events).
• Processing large amounts of context information in an aggregated way, using map&reduce techniques, in order to generate knowledge that may also lead to execution of actions and/or creation of new context information.

• Process data streams (particularly, multimedia video streams) coming from different sources in order to generate new data streams as well as context information that can be further exploited.

• Process metadata that may be linked to context information, using standard semantic support technologies.

• Manage some context information, such Location information, in a standardized way.

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

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

The following diagram shows the main components (Generic Enablers) that comprise the first release of Fi-WARE Data/Context chapter architecture.
More information about the Data Chapter and FI-WARE in general can be found within the following pages:

http://wiki.fi-ware.eu

Data/Context Management Architecture

Materializing Data/Context_Management_in_FI-WARE

1.5 Structure of this Document

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

The following resources were used to generate this document:

D.6.4.2 User and Programmers Guide front page

CEP GE - IBM Proactive Technology Online User and Programmer Guide

Location Platform - User and Programmers Guide

Compressed Domain Video Analysis - User and Programmers Guide
1.6 Typographical Conventions

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

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

1.6.1 Links within this document

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

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

1.6.2 Figures

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

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

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

1.6.3 Sample software code

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

```
http://[SERVER_URL]?filter=name:Simth*&index=20&limit=10
```
1.7 Acknowledgements
The current document has been elaborated using a number of collaborative tools, with the participation of Working Package Leaders and Architects as well as those partners in their teams they have decided to involve.

1.8 Keyword list

1.9 Changes History

<table>
<thead>
<tr>
<th>Release</th>
<th>Major changes description</th>
<th>Date</th>
<th>Editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>First Version</td>
<td>2013-04-22</td>
<td>TID</td>
</tr>
<tr>
<td>v2</td>
<td>To Review</td>
<td>2013-05-17</td>
<td>TID</td>
</tr>
<tr>
<td>v3</td>
<td>Final version for submission</td>
<td>2013-05-17</td>
<td>TID</td>
</tr>
</tbody>
</table>

1.10 Table of Contents

1.1 Executive Summary ........................................................................................................ 2
1.2 About This Document ...................................................................................................... 2
1.3 Intended Audience .......................................................................................................... 2
1.4 Chapter Context ............................................................................................................... 2
1.5 Structure of this Document ............................................................................................. 4
1.6 Typographical Conventions .............................................................................................. 5
  1.6.1 Links within this document ....................................................................................... 5
  1.6.2 Figures ......................................................................................................................... 5
  1.6.3 Sample software code ................................................................................................. 5
1.7 Acknowledgements ........................................................................................................... 6
1.8 Keyword list ..................................................................................................................... 6
1.9 Changes History ................................................................................................................. 6
1.10 Table of Contents ........................................................................................................... 6
2 CEP GE - IBM Proactive Technology Online User and Programmer Guide ....................... 10
2.1 Introduction .................................................................................................................................................. 10
2.2 User Guide.................................................................................................................................................. 10
2.3 Programmer Guide ...................................................................................................................................... 10

3 Location Platform - User and Programmers Guide ....................................................................................... 11
3.1 Introduction ................................................................................................................................................ 11
3.2 User Guide ................................................................................................................................................ 11
3.2.1 Location Platform Start-up / Shutdown ................................................................................................. 11
3.2.2 Handset fleet simulation ......................................................................................................................... 11
3.2.3 Location Platform Agents Detailed Configuration .................................................................................. 11
3.2.4 Location Platform Database Configuration .......................................................................................... 15
3.3 Programmer guide ...................................................................................................................................... 16

4 Compressed Domain Video Analysis - User and Programmers Guide .......................................................... 19
4.1 Introduction ................................................................................................................................................ 19
4.2 User Guide ................................................................................................................................................ 19
4.2.1 Functionality .......................................................................................................................................... 19
4.2.2 Example Usage Scenario ....................................................................................................................... 20
4.3 Programmers Guide ................................................................................................................................... 37
4.3.1 RESTful API .......................................................................................................................................... 37
4.3.2 Application Programming Interface (API) .............................................................................................. 40

5 Query Broker - User and Programmers Guide ............................................................................................. 41
5.1 Introduction ................................................................................................................................................ 41
5.2 User guide .................................................................................................................................................. 41
5.3 Programmer guide ..................................................................................................................................... 42
5.4 References ............................................................................................................................................... 57

6 Metadata Preprocessing - User and Programmers Guide ............................................................................. 59
6.1 Introduction ................................................................................................................................................ 59
6.2 User Guide ................................................................................................................................................ 59
6.2.1 Functionality .......................................................................................................................................... 59
6.2.2 Example usage ....................................................................................................................................... 60
6.3 Programmers Guide .................................................................................................................................. 63
6.3.1 API overview ......................................................................................................................................... 64
6.3.2 API reference ......................................................................................................................................... 65

7 Semantic Application Support - Users and Programmers Guide ................................................................. 67
7.1 Introduction ................................................................................................................................................ 67
2 CEP GE - IBM Proactive Technology Online User and Programmer Guide

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

2.1 Introduction

The CEP GE is implemented by IBM Proactive Technology Online (a.k.a Proton). IBM Proactive Technology Online is a scalable integrated platform to support the development, deployment, and maintenance of event-driven applications. While standard reactive applications are based on reactions to single events, Proton reacts to situations rather than to single events. A situation is a condition that is based on a series of events that have occurred within a dynamic time window called a context. Situations include composite events (e.g., sequence), counting operators on events (e.g., aggregation) and absence operators. The Proton engine receives information on the occurrence of events from event producers, detects situations, and reports the detected situations to external consumers.

2.2 User Guide

The IBM Proactive Technology Online User Guide explains how to create a complex event processing (CEP) application, also known as an Event Processing Network. It describes the language constructs that are used to build CEP applications and includes a tutorial that explains how this can be done using the IBM Proactive Technology Online authoring tool. The User Guide can be found in the following link:

IBM Proactive Technology Online User Guide

2.3 Programmer Guide

The Programmer Guide describes the processing engine of IBM Proactive Technology Online. The programmer guide includes the high level architecture of the engine, the various supported adapters that can be used by the event producers to send events to the engine and the various supported adapters that can be used to send derived events from the engine to the consumers. It describes how a built-in adapter type can be used and how a new adapter type can be added. Configuration information and running instructions are also included. The Programmer Guide can be found in the following link:

IBM Proactive Technology Online Programmer Guide
3 Location Platform - User and Programmers Guide

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

3.1 Introduction

This document aggregates Programming Guide and User Guide of Location GE.

LOCS implements the FI-WARE GE Open Specifications associated to the Location GE available at FIWARE.ArchitectureDescription.Data.Location

Whenever the term "Location GE" is used, you may assume that we are indeed referring to LOCS that implements the Location GE Open Specifications or an instance of LOCS.

3.2 User Guide

3.2.1 Location Platform Start-up / Shutdown

Location Platform Server start-up/shutdown is described in chapter 3/4 of of Location GE Installation and Administration Guide.

Refer to the following procedures in Location GE Installation and Administration Guide.

3.2.2 Handset fleet simulation

A Handset simulator is also provided to simulate a fleet of mobile that can move along configurable geographic paths and which positions can be transparently injected through Location GE services.

Refer the the following Handset Simulator REST API Information for details in Location GE Unit Test Plan.

3.2.3 Location Platform Agents Detailed Configuration

This chapter describes in details the configuration items for :

- MLP Agent handling Terminal Location REST API.
- SUPL Agent managing SUPL messages exchanges with terminal handset.

3.2.3.1 MLP Agent detailed configuration

Only parameters used in association with Terminal Location API are indicated. Others parameters shall be kept unchanged. Configuration parameters which could have a need to be modified in scope of Location Platform GE deployment are in green.
### Section 3.2.3.2 SUPL Agent detailed configuration

Configuration parameters which could have a need to be modified in scope of Location Platform GE deployment are in green.

<table>
<thead>
<tr>
<th>Section</th>
<th>Parameter Name</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operator</td>
<td>Operator Identifier</td>
<td>String</td>
<td>Sets the name of deployed operator, as configured in dbMonitor.</td>
</tr>
<tr>
<td>Configuration</td>
<td>HTTP Rest Port</td>
<td>String</td>
<td>Http port through which are received the Terminal Location REST requests.</td>
</tr>
<tr>
<td>Configuration</td>
<td>HTTP Port</td>
<td>Integer</td>
<td>Http port through which are received the MLP requests.</td>
</tr>
<tr>
<td></td>
<td>Current System Max Location Age</td>
<td>Integer</td>
<td>Sets the location age for defining the age of a current location if not defined in location request.</td>
</tr>
<tr>
<td></td>
<td>Default coordinate reference system</td>
<td>String</td>
<td>Sets the default coordinate reference system to be used. Values allowed: EPSG_6_1_4326, EPSG_6_1_4327, EPSG_6_1_4124, EPSG_6_1_2400, EPSG_6_1_27572.</td>
</tr>
<tr>
<td></td>
<td>Location timeout in seconds</td>
<td>Integer</td>
<td>Sets the time interval after which a location request is considered as non-responsive (timeout).</td>
</tr>
</tbody>
</table>

**3.2.3.2 SUPL Agent detailed configuration**

Configuration parameters which could have a need to be modified in scope of Location Platform GE deployment are in green.

<table>
<thead>
<tr>
<th>Section</th>
<th>Parameter Name</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operator</td>
<td>Operator Identifier</td>
<td>String</td>
<td>Sets the name of deployed operator, as configured in dbMonitor.</td>
</tr>
<tr>
<td>Configuration</td>
<td>Tcp port</td>
<td>Integer</td>
<td>SUPL TCP port, default is 14050.</td>
</tr>
<tr>
<td>SUPL Parameters</td>
<td>Assistance delivery Method</td>
<td>Enum</td>
<td>Assistance data message delivery: ADM / MPR.</td>
</tr>
<tr>
<td>Security parameters</td>
<td>Preferred Trigger Report Capability Mode</td>
<td>Enum</td>
<td>Preferred Trigger Report Capability set for the server: RealTime / QuasiRealTime / Batch.</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------</td>
<td>------</td>
<td>------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Ver field (HMAC) Verification</td>
<td>Boolean</td>
<td>Indicates if activate or not the verification of the HMAC ver field.</td>
</tr>
<tr>
<td></td>
<td>Current Based Verification</td>
<td>Boolean</td>
<td>Indicates if activate or not the Current Bases Verification.</td>
</tr>
<tr>
<td>SUPL Time outs</td>
<td>SETINIT: SUPL_START timeout</td>
<td>Integer (sec)</td>
<td>SUPL_START maximum session duration.</td>
</tr>
<tr>
<td></td>
<td>SETINIT: SUPL_POS_INIT timeout</td>
<td>Integer (sec)</td>
<td>SUPL_POS maximum session duration.</td>
</tr>
<tr>
<td></td>
<td>NETINIT: SUPL_POS_INIT timeout</td>
<td>Integer (sec)</td>
<td>SUPL_POS_INIT maximum session duration.</td>
</tr>
<tr>
<td></td>
<td>NETINIT: SUPL_POS timeout</td>
<td>Integer (sec)</td>
<td>SUPL_POS maximum session duration.</td>
</tr>
<tr>
<td></td>
<td>NETINIT: SUPL_NOTIF timeout</td>
<td>Integer (sec)</td>
<td>SUPL_NOTIF maximum session duration.</td>
</tr>
<tr>
<td></td>
<td>NETINIT: SUPL_INIT timeout</td>
<td>Integer (sec)</td>
<td>SUPL_INIT maximum session duration.</td>
</tr>
<tr>
<td></td>
<td>NETINIT: SUPL_TRIGGER_STOP timeout</td>
<td>Integer (sec)</td>
<td>SUPL_TRIGGER_STOP maximum session duration.</td>
</tr>
<tr>
<td></td>
<td>NETINIT: SUPL_TRIGGER_STOP tolerance</td>
<td>Integer (sec)</td>
<td>SUPL_TRIGGER_STOP maximum session tolerance delta time.</td>
</tr>
<tr>
<td>Payload Parameters</td>
<td>Horizontal accuracy</td>
<td>Integer (m)</td>
<td>Expected horizontal accuracy in meters.</td>
</tr>
<tr>
<td></td>
<td>Response time</td>
<td>Integer (sec)</td>
<td>Response time value.</td>
</tr>
<tr>
<td></td>
<td>Trace payload data</td>
<td>Boolean</td>
<td>Indicates if trace decoded payload data in agent log.</td>
</tr>
<tr>
<td>Assistance Data General</td>
<td>Maximum number of assistance data delivery</td>
<td>Integer (sec)</td>
<td>Maximum number of assistance data delivery retries to the SET.</td>
</tr>
<tr>
<td>Parameters</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>retries to the SET</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SET to GPS satellites elevation cut-off angle</td>
<td>Degrees</td>
<td>Enter SET to GPS satellites elevation cut-off angle</td>
<td></td>
</tr>
<tr>
<td>Minimum Code Search Window</td>
<td>N/A</td>
<td>Select minimum code search window (chips)</td>
<td></td>
</tr>
<tr>
<td>Minimum Doppler Uncertainty</td>
<td>Hz</td>
<td>Select minimum Doppler uncertainty</td>
<td></td>
</tr>
<tr>
<td>Assistance data specific parameters</td>
<td>SET to LRX maximum distance for local DGPS corrections</td>
<td>K meters</td>
<td>Enter LRX-SET maximum distance for local differential corrections.</td>
</tr>
<tr>
<td>Maximum PRC for local DGPS corrections</td>
<td>Meters</td>
<td>Enter maximum PRC for local differential corrections.</td>
<td></td>
</tr>
<tr>
<td>Maximum PRRC for local DGPS corrections</td>
<td>Meters</td>
<td>Enter maximum PRRC for local differential corrections.</td>
<td></td>
</tr>
<tr>
<td>Assistance Data Set-Assisted only Parameters</td>
<td>Fix refining search window</td>
<td>m seconds</td>
<td>Enter fix refining search window.</td>
</tr>
<tr>
<td>Assistance Data Set-Assisted only Parameters</td>
<td>Fix refining search window</td>
<td>m seconds</td>
<td>Enter fix refining search window.</td>
</tr>
<tr>
<td>Fix refining tolerance</td>
<td>m seconds</td>
<td>Enter fix refining tolerance.</td>
<td></td>
</tr>
<tr>
<td>Fix refining min C/N0 threshold</td>
<td>DB-Hz</td>
<td>Enter fix refining min C/N0 threshold.</td>
<td></td>
</tr>
<tr>
<td>Fix refining min C/N0 threshold</td>
<td>DB-Hz</td>
<td>Enter fix refining min C/N0 threshold.</td>
<td></td>
</tr>
<tr>
<td>Horizontal Uncertainty Factor (alpha)</td>
<td>N/A</td>
<td>Enter Horizontal Uncertainty Factor * 10.</td>
<td></td>
</tr>
<tr>
<td>Vertical Uncertainty Factor (beta)</td>
<td>N/A</td>
<td>Enter Vertical Uncertainty Factor * 10.</td>
<td></td>
</tr>
<tr>
<td>Confidence Factor (gamma)</td>
<td>N/A</td>
<td>Enter Confidence Factor * 10.</td>
<td></td>
</tr>
<tr>
<td>Maximum latitude error allowed on SET computed</td>
<td>Integer</td>
<td>Maximum latitude error allowed on SET computed position in</td>
<td></td>
</tr>
</tbody>
</table>
3.2.4 Location Platform Database Configuration

3.2.4.1 **CELLS Database**

This database handles description of known network 2G/3G/WLAN cells. The database schema is described here below:

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1, id2, id3, id4, idType</td>
<td>INT, ENUM(...)</td>
</tr>
<tr>
<td>latitude</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>longitude</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>altitude</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>uncertaintySemiMajor</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>uncertaintySemiMinor</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>uncertainityAltitude</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>majorAxesOrientation</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>antennaType</td>
<td>ENUM(...)</td>
</tr>
<tr>
<td>description</td>
<td>VARCHAR(256)</td>
</tr>
<tr>
<td>beginAngle</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>endAngle</td>
<td>DOUBLE</td>
</tr>
</tbody>
</table>

As given in example in validation test data file `$LOCS_HOME/data/fiware_validation.sql`, the cell database can be easily populated using the MySQL stored procedure:

- `addCell()` : to create network cell descriptions

3.2.4.2 **OPE Database**

This database handles description of known end-user terminals, service provider, service and associated privacy policies for each end-user.

The database schema is described here below:
As given in example in validation test data file $LOCS_HOME/data/fiware_validation.sql, the cell database can be easily populated using the MySQL stored procedure:

- `addEndUser()`, `addHandset()`, `associateHandset()`: to describe known terminals and their main characteristics.
- `addServiceProvider()`, `addService()`, `associateService()`: to create Service Provider and associated Services.
- `addPrivacyPolicy()`: to create privacy policies for each terminal end-user/service.

3.3 Programmer guide

The Network API for Terminal Location is a RESTful, resource-oriented API that allows to access to the following localization services:
- Obtain the current terminal location
- Manage client-specific subscriptions to periodic notifications
- Manage client-specific subscriptions to area (circle) notifications

A simple example of Location Query request for device telephone URI 33611223344 is presented here below:

```
GET
/location/v1/queries/location?requester=test:test&address=tel:33611223344&requestedAccuracy=1000&acceptableAccuracy=1000
&maximumAge=1000&tolerance=DelayTolerant HTTP/1.1
Accept: application/xml
Host: example.com
```

The following response is received, including handset position.

```
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>150.0</tl:altitude>
      <tl:accuracy>110</tl:accuracy>
      <tl:timestamp>2012-04-17T09:21:32.893+02:00</tl:timestamp>
    </tl:currentLocation>
  </tl:terminalLocation>
</tl:terminalLocationList>
```
For more complete usage of Location Platform API and description of XML structures, please refer also to the REST API documentation provided here [RESTful API documentation](link), that provides examples of requests/responses for each exposed services.
4 Compressed Domain Video Analysis - User and Programmers Guide

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

4.1 Introduction

In the following, a detailed description on how to use the functionality and interfaces of the Compressed Domain Video Analysis GE as realized by Codoan is provided. First, a comprehensive overview of the functionality of Codoan is given. This is further illustrated by an example usage scenario with two separate sinks/users. The respective API calls for this example usage are further detailed, which completes the User Guide. The Programmers Guide mainly consists of an overview of the RESTful API calls and an API reference for this GE.

4.2 User Guide

Codoan provides 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.

4.2.1 Functionality

Codoan follows the following basic design principles:

- Critical product attributes for Codoan are especially high detection rates containing only few false positives and low-complexity operation.

- Partitioning to independent functional blocks enables Codoan to support a variety of analysis methods and to get easily extended by new features. Even several operations can be combined.

- Low-complexity algorithms and implementations enable Codoan to perform very fast analyses and to be highly scalable.

- Codoan supports performing parallel analyses using different sub-components.

The following diagram depicts the generic functional blocks of Codoan.
4.2.2 Example Usage Scenario

Employing Codoan usually requires several common steps to be performed. The following figure shows an example of a typical usage scenario (two analyzer instances (event/object recognition) attached to a media source). Note that responses and notifications are not shown for reasons of clarity and comprehensibility.

Sample usage scenario of Codoan

This scenario contains the following requests (in chronological order):
- `listInstances` (Sink 1)
  - lists all existing analyzer instances
  - a single instance receives and analyzes a single video stream
  - Request
    - GET `{serverRoot}/codoan/instances` HTTP/1.1
    - Accept: application/xml
  - Response
    - HTTP/1.1 200 OK
    - Content-Length: 100
    - Content-Type: application/xml
    - Server: codoan RESTful web server (Mongoose web server)
    - 
      ```xml
      <?xml version="1.0" encoding="UTF-8"?>
      <Codoan>
      <Instances/>
      </Codoan>
      ```

- `createInstance` (Sink 1)
  - creates a new analyzer instance
  - an instance with the same parameters may not already exist
  - Request
    - POST `{serverRoot}/codoan/instances` HTTP/1.1
    - Accept: application/xml
    - Content-Length: 185
    - Content-Type: application/xml
    - 
      ```xml
      <?xml version="1.0" encoding="UTF-8"?>
      <Codoan>
      <Instances>
      <Instance detectEvents="true" detectObjects="true"
streamURI="rtsp://192.0.2.2/stream1"/>
      </Instances>
      </Codoan>
      ```
o Response

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

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

- `configureInstance` (Sink 1)
  - configures the parameters of the detection modules
  - whether event or object detection is activated depends on the `createInstance` request

o Request

o PUT //{serverRoot}/codoan/instances/101/config
  HTTP/1.1
  Accept: application/xml
  Content-Length: 380
  Content-Type: application/xml

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

Response

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

<?xml version="1.0" encoding="UTF-8"?>
<Codoan>
<Instances>
<Instance activeSinks="0" detectEvents="false" detectObjects="true" id="101" isRunning="false" streamURI="rtsp://192.0.2.2/stream1">
<Configuration>
<Event type="GlobalChange">
<NumberOfTrainingFrames>40</NumberOfTrainingFrames>
<SlidingWindowSize>10</SlidingWindowSize>
<ThresholdANORPFactor>1.2</ThresholdANORPFactor>
<ThresholdARPSFactor>1.75</ThresholdARPSFactor>
<ThresholdIFrame>5</ThresholdIFrame>
</Event>
<Object type="Person">
<BoxFilterSize>3</BoxFilterSize>
<ThresholdH264MOC>6</ThresholdH264MOC>
</Object>
</Configuration>
</Instance>
</Instances>
- **addSink (Sink 1)**
  - adds a sink (or observer) to a specific instance
  - once a sink has been registered it gets notified in case of an event or a moving object
  - Request

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

- Response

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

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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Codoan>
  <Instances>
    </Instances>
  </Codoan>
```
- **startInstance** (Sink 1)
  - starts an existing instance
  - a single instance receives and analyzes a single video stream
  - Request
    - PUT //{serverRoot}/codoan/instances/101?action=start HTTP/1.1
    - Accept: application/xml
  - Response
    - HTTP/1.1 200 OK
    - Content-Length: 227
    - Content-Type: application/xml
    - Server: codoan RESTful web server (Mongoose web server)
    - <?xml version="1.0" encoding="UTF-8"?>
    - <Codoan>
    - <Instances>
    - <Instance activeSinks="1" detectEvents="true" detectObjects="true" id="101" isRunning="true" streamURI="rtsp://192.0.2.2/stream1"/>
    - </Instances>
    - </Codoan>

- **listInstances** (Sink 2)
  - lists all existing analyzer instances
- a single instance receives and analyzes a single video stream

- Request
  - GET //{serverRoot}/codoan/instances HTTP/1.1
  - Accept: application/xml

- Response
  - HTTP/1.1 200 OK
  - Content-Length: 366
  - Content-Type: application/xml
  - Server: codoan RESTful web server (Mongoose web server)

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

- **addSink** (Sink 2)
  - adds a sink (or observer) to a specific instance
  - once a sink has been registered it gets notified in case of an event or a moving object

- Request
  - POST //{serverRoot}/codoan/instances/101/sinks HTTP/1.1
  - Accept: application/xml
  - Content-Length: 293
  - Content-Type: application/xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Codoan>
  <Instances>
    ...
  </Instances>
</Codoan>
```
**removeSink** (Sink 1)

- removes a sink (or observer) from the given instance
- no notifications will be sent anymore when the sink has been removed

**Request**

```
DELETE //{serverRoot}/codoan/instances/101/sinks/201
HTTP/1.1
Accept: application/xml
```
Response

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

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

- `createInstance` (Sink 2)
  - creates a new analyzer instance
  - an instance with the same parameters may not already exist

Request

POST //{serverRoot}/codoan/instances HTTP/1.1
Accept: application/xml
Content-Length: 185
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<Codoan>
<Instances>
<Instance detectEvents="true" detectObjects="true" streamURI="rtsp://192.0.2.4/stream1"/>
</Instances>
</Codoan>
configureInstance (Sink 2)

- configures the parameters of the detection modules
- whether event or object detection is activated depends on the createInstance request

Request

- PUT //{serverRoot}/codoan/instances/102/config
- HTTP/1.1
- Accept: application/xml
- Content-Length: 380
- Content-Type: application/xml

- <?xml version="1.0" encoding="UTF-8"?>
- <Codoan>
- <Instances>
-   <Instance detectEvents="false" detectObjects="true" id="102" isRunning="false" streamURI="rtsp://192.0.2.4/stream1">
-     <Configuration>
-       <Object type="Person">
-     </Object>
-   </Instance>
- </Instances>
- </Codoan>
<Response>

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

<?xml version="1.0" encoding="UTF-8"?>
<Codoan>
<Instances>
<Instance activeSinks="0" detectEvents="false" detectObjects="true" id="102" isRunning="false" streamURI="rtsp://192.0.2.4/stream1">
<Configuration>
<Event type="GlobalChange">
<NumberOfTrainingFrames>40</NumberOfTrainingFrames>
<SlidingWindowSize>10</SlidingWindowSize>
<ThresholdANORPFactor>1.2</ThresholdANORPFactor>
<ThresholdARPSFactor>1.75</ThresholdARPSFactor>
<ThresholdIFrame>5</ThresholdIFrame>
</Event>
<Object type="Person">
<BoxFilterSize>5</BoxFilterSize>
<ThresholdH264MOC>7</ThresholdH264MOC>
</Object>
</Configuration>
</Instance>
</Instances>
</Codoan>
• **addSink** (Sink 2)
  - adds a sink (or observer) to a specific instance
  - once a sink has been registered it gets notified in case of an event or a moving object

  **Request**

  ```
  POST //{serverRoot}/codoan/instances/102/sinks
  HTTP/1.1
  Accept: application/xml
  Content-Length: 293
  Content-Type: application/xml

  <?xml version="1.0" encoding="UTF-8"?>
  <Codoan>
    <Instances>
      <Instance detectEvents="true" detectObjects="true"
        streamURI="rtsp://192.0.2.4/stream1">
        <Sinks>
          <Sink
            sinkNotificationURI="http://192.0.2.6/notification/stream1"/>
        </Sinks>
      </Instance>
    </Instances>
  </Codoan>
  ```

  **Response**

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

  <?xml version="1.0" encoding="UTF-8"?>
  <Codoan>
    <Instances>
  ```
- **startInstance** (Sink 2)
  - starts an existing instance
  - a single instance receives and analyzes a single video stream
  - Request
    - PUT \/{serverRoot}\/{codoan\}/instances/102?action=start HTTP/1.1
    - Accept: application/xml
  - Response
    - HTTP/1.1 200 OK
    - Content-Length: 227
    - Content-Type: application/xml
    - Server: codoan RESTful web server (Mongoose web server)
    - 
      <?xml version="1.0" encoding="UTF-8"?>
      <Codoan>
        <Instances>
          <Instance activeSinks="1" detectEvents="true" detectObjects="true" id="102" isRunning="true" streamURI="rtsp://192.0.2.4/stream1"/>
        </Instances>
      </Codoan>

- **removeSink** (Sink 2)
  - removes a sink (or observer) from the given instance
- no notifications will be sent anymore when the sink has been removed

**Request**

```plaintext
DELETE //{serverRoot}/codoan/instances/101/sinks/202
HTTP/1.1
Accept: application/xml
```

**Response**

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

<?xml version="1.0" encoding="UTF-8"?>
<Codoan>
  <Instances>
    <Instance activeSinks="1" detectEvents="true" detectObjects="true" id="101" isRunning="false" streamURI="rtsp://192.0.2.2/stream1">
      <Sinks>
        <Sink id="202" sinkNotificationURI="http://192.0.2.5/notification/stream1"/>
      </Sinks>
    </Instance>
  </Instances>
</Codoan>
```

- **stopInstance** (Sink 2)

  - stops an existing instance
  - a single instance receives and analyzes a single video stream

  **Request**

  ```plaintext
  PUT //{serverRoot}/codoan/instances/101?action=stop
  HTTP/1.1
  Accept: application/xml
  ```

  **Response**
- `destroyInstance` (Sink 2)

  - destroys a previously created instance
  - an instance can not be destroyed if it has active sinks connected

  **Request**

  ```
  DELETE //{{serverRoot}}/codoan/instances/101 HTTP/1.1
  Accept: application/xml
  ```

  **Response**

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

  ```
  <?xml version="1.0" encoding="UTF-8"?>
  <Codoan>
  <Instances>
  <Instance activeSinks="0" detectEvents="true" detectObjects="true" id="101" isRunning="false" streamURI="rtsp://192.0.2.2/stream1"/>
  </Instances>
  </Codoan>
  ```
• **removeSink** (Sink 2)
  - removes a sink (or observer) from the given instance
  - no notifications will be sent anymore when the sink has been removed
  - **Request**
    
    ```
    DELETE //{serverRoot}/codoan/instances/102/sinks/201
    HTTP/1.1
    Accept: application/xml
    ```
  - **Response**
    
    ```
    HTTP/1.1 200 OK
    Content-Length: 328
    Content-Type: application/xml
    Server: codoan RESTful web server (Mongoose web server)
    ```
    ```xml
    <?xml version="1.0" encoding="UTF-8"?>
    <Codoan>
    <Instances>
    <Instance activeSinks="1" detectEvents="true" detectObjects="true" id="102" isRunning="false" streamURI="rtsp://192.0.2.4/stream1">
    <Sinks>
    <Sink id="201" sinkNotificationURI="http://192.0.2.5/notification/stream1"/>
    </Sinks>
    </Instance>
    </Instances>
    </Codoan>
    ```

• **stopInstance** (Sink 2)
  - stops an existing instance
  - a single instance receives and analyzes a single video stream
  - **Request**
- **PUT** //{serverRoot}/codoan/instances/102?action=stop HTTP/1.1
- Accept: application/xml

- Response

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

  ```xml
  <?xml version="1.0" encoding="UTF-8"?>
  <Codoan>
  <Instances>
    <Instance activeSinks="1" detectEvents="true" detectObjects="true" id="102" isRunning="false" streamURI="rtsp://192.0.2.4/stream1"/>
  </Instances>
  </Codoan>
  ```

- **destroyInstance** (Sink 2)

  - destroys a previously created instance
  - an instance can not be destroyed if it has active sinks connected

- Request

  - DELETE //{serverRoot}/codoan/instances/102 HTTP/1.1
  - Accept: application/xml

- Response

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

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

4.3.1 RESTful API

For a convenient usage of Codoan a RESTful, resource-oriented API accessed via HTTP, that uses XML-based representations for information interchange, has been provided.

The following graphical diagram summarizes the available resources:

```
Codoan (server)  
----------------------  
//{serverRoot}/codoan  
  |  
  |-- /version  
  GET -> getVersion  
  |  
  |-- /instances  
  GET -> listInstances  
  |  
  |-- /{instanceID}  
  POST -> createInstance  
  |  
  |-- /{instanceID}  
  GET -> getInstanceInfo  
  |  
  DELETE -> destroyInstance  
  |  
  |-- ?action=start  
  PUT -> startInstance  
  |  
  |-- ?action=start  
  PUT -> stopInstance  
  |  
  |-- /config  
  GET -> getInstanceConfig  
  |  
  PUT -> configureInstance  
```
GET -> listSinks
|  |<-- /sinks
POST -> addSink
GET -> getSinkInfo
DELETE -> removeSink

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

The corresponding API operations cover management, execution, and information purposes:

Get the version of Codoan

<table>
<thead>
<tr>
<th>Verb</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>{//serverRoot}/codoan/version</td>
<td>getVersion: get the current version of Codoan</td>
</tr>
</tbody>
</table>

Listing active instances of Codoan

<table>
<thead>
<tr>
<th>Verb</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>{//serverRoot}/codoan/instances</td>
<td>listInstances: lists all active instances of Codoan</td>
</tr>
</tbody>
</table>

Creating a new instance of Codoan

<table>
<thead>
<tr>
<th>Verb</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>{//serverRoot}/codoan/instances</td>
<td>createInstance: creates a new instance of Codoan</td>
</tr>
</tbody>
</table>

Get info about the instance of Codoan with ID instanceID

<table>
<thead>
<tr>
<th>Verb</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>{//serverRoot}/codoan/instances/{instanceID}</td>
<td>getInstanceInfo: get info about the instance with ID instanceID</td>
</tr>
</tbody>
</table>

Destroying an instance of Codoan with ID instanceID
### Verb | URI | Description
---|---|---
DELETE | `{serverRoot}/codoan/instances/{instanceID}` | `destroyInstance`: destroys the instance with ID `instanceID`

**Start an instance of Codoan with ID `instanceID`**

<table>
<thead>
<tr>
<th>Verb</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUT</td>
<td><code>{serverRoot}/codoan/instances/{instanceID}</code>?action=start</td>
<td><code>startInstance</code>: starts the instance with ID <code>instanceID</code></td>
</tr>
</tbody>
</table>

**Stop an instance of Codoan with ID `instanceID`**

<table>
<thead>
<tr>
<th>Verb</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUT</td>
<td><code>{serverRoot}/codoan/instances/{instanceID}</code>?action=stop</td>
<td><code>stopInstance</code>: stops the instance with ID <code>instanceID</code></td>
</tr>
</tbody>
</table>

**Get config of the instance of Codoan with ID `instanceID`**

<table>
<thead>
<tr>
<th>Verb</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td><code>{serverRoot}/codoan/instances/{instanceID}/config</code></td>
<td><code>getInstanceConfig</code>: get the config of the instance with ID <code>instanceID</code></td>
</tr>
</tbody>
</table>

**Configuring the instance of Codoan with ID `instanceID`**

<table>
<thead>
<tr>
<th>Verb</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUT</td>
<td><code>{serverRoot}/codoan/instances/{instanceID}/config</code></td>
<td><code>configureInstance</code>: configures the instance with ID <code>instanceID</code></td>
</tr>
</tbody>
</table>

**List all sinks (observers) of an instance of Codoan with ID `instanceID`**

<table>
<thead>
<tr>
<th>Verb</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td><code>{serverRoot}/codoan/instances/{instanceID}/sinks</code></td>
<td><code>listSinks</code>: lists all sinks (observers) of the instance with ID <code>instanceID</code></td>
</tr>
</tbody>
</table>

**Adding a new sink (observer) to an instance of Codoan with ID `instanceID`**
<table>
<thead>
<tr>
<th>Verb</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td><code>{serverRoot}/codoan/instances/{instanceID}/sinks</code></td>
<td><em>addSink</em>: adds a new sink (observer) to the instance with ID <code>instanceID</code></td>
</tr>
</tbody>
</table>

Get info about the sink (observer) `sinkID` from the Codoan instance with ID `instanceID`

<table>
<thead>
<tr>
<th>Verb</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td><code>{serverRoot}/codoan/instances/{instanceID}/sinks/{sinkID}</code></td>
<td><em>getSinkInfo</em>: gets info about the sink (observer) with ID <code>sinkID</code> from instance with ID <code>instanceID</code></td>
</tr>
</tbody>
</table>

Removing the sink (observer) `sinkID` from the Codoan instance with ID `instanceID`

<table>
<thead>
<tr>
<th>Verb</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE</td>
<td><code>{serverRoot}/codoan/instances/{instanceID}/sinks/{sinkID}</code></td>
<td><em>removeSink</em>: removes the sink (observer) with ID <code>sinkID</code> from instance with ID <code>instanceID</code></td>
</tr>
</tbody>
</table>

Notifying the sink (observer) in case of events or detected objects

<table>
<thead>
<tr>
<th>Verb</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td><code>{sinkNotificationURI}</code></td>
<td><em>notifySink</em>: notifies the sink (observer) in case of events or detected objects</td>
</tr>
</tbody>
</table>

For a more detailed description on how to use Codoan’s RESTful API, please refer also to the RESTful API documentation provided here [RESTful API Specification](#), that provides examples of requests/responses for each exposed service.

### 4.3.2 Application Programming Interface (API)

It is envisaged that Codoan shall be used as a service employing the RESTful API above. In the case that this (REST) interface is not applicable and Codoan needs to be accessed directly from other applications (e.g., via linking the library), function calls, which correspond to the above listed REST resources, can/will be supported (foreseen in Release 2).
5 Query Broker - User and Programmer Guide

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

5.1 Introduction

In the following a detailed description on how to access the software modules and interfaces of the Media-enhanced QueryBroker is provided. Beside a comprehensive overview on the RESTful interface for accessing the Query Broker the necessary steps to integrate the QueryBroker into another application is explained. Additionally a code example is given, which shows an example implementation of all required steps to initialize and run the QueryBroker.

5.2 User guide

The QueryBroker provides a REST interface - 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 data repositories by sending according MPQF expressions.

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 according MPQF expressions can be lengthy this is done by POST allowing to transmit the data in the body of the http request.

Resources summary:

```
QueryBroker Component
://{serverRoot}/QueryBrokerServer

 /version
 /query
 /services
   /{serviceID}
   /{serviceID}/{capability}
   /{serviceID}/CapabilityDescription
     /{capability}
 /link
   /{serviceID1}/{linkField1}/{serviceID2}/{linkField2}
   /{serviceID1}/{serviceID2}
```

For a more detailed description on how to use the QueryBroker REST-API, please refer also to the REST API documentation provided on [Query Broker Open RESTful API Specification (PRELIMINARY)](http://example.com), that provides examples of requests/responses for each exposed service.

### 5.3 Programmer guide

The QueryBroker is a J2SE (Java 2 Platform, Standard Edition) application. It requires Java JRE 6 or later installed. There exist two different programming APIs for invoking the QueryBroker, namely a JAVA-based one provisioning according methods and a RESTful, resource-oriented API accessed via HTTP that uses XML-based representations for information interchange. The REST interface is realized as a wrapper and provided as WAR file (Web application ARchive) together with the QueryBroker core.
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).

5.3.1.1 Architecture

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

The functionalities of the key elements are as follows:

- **BackendManagement** provides the functionality to register and remove service endpoints.
  - **Service** interface has to be implemented by any service endpoint. A service endpoint connects a database or another dataset to the multimedia query framework.
- **Broker** represents the central access point to the federated query framework. It provides the functionality to query distributed and heterogeneous multimedia databases using MPQF as query format. The main task is to receive MPQF-queries and control the following request processing (synchronous / asynchronous mode of...
Future Internet Core Platform

operation or result fetching). See Chapter Frontend Functionality for more information.

- **QueryManager** handles all received and active queries. So, new queries can be checked-in and corresponding result sets can be checked-out by the Broker.
- **RequestProcessing** controls a 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 to 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. That 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. So the aggregator can combine these two elements and process the queried results.

5.3.1.2 **RESTful Interface**

The REST API 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.

Running the QueryBroker requires at least registering one or more available data stores before submitting according queries.

In order to register a certain data repository a "data base connector", a.k.a. service interface has to be implemented by any service endpoint and installed at the QueryBroker. A service endpoint connects a database or another dataset to the multimedia query framework (see **QueryBroker GE Installation and Administration Guide** how to do that).

Assuming that according service interfaces are already implemented and installed, a service endpoint can be registered by the following POST method:

Request example:

```
POST //localhost/QueryBrokerServer/services/de.uop.dimis.air.adapters .LireAdapter/QueryByMedia HTTP/1.1
Accept: */*
```
This registers the service with the serviceID "de.uop.dimis.air.adapters.LireAdapter" at the QueryBroker endpoint. The ServiceID must be equal to the qualified name of the implementation class. The DesiredCapabilities declare which query types the service can handle. In this example the "de.uop.dimis.air.adapters.LireAdapter" can handle Query-By-Media.

Now a simple query can be submitted by calling the following POST method (The request body must contain a valid MPQF query in XML serialization.):

Request example:

```xml
POST //localhost/QueryBrokerServer/query HTTP/1.1
Host: localhost:8080
Accept: */*
Content-Type: application/xml
Content-Length: 425

<?xml version="1.0" encoding="UTF-8"?>
<mpqf:MpegQuery mpqfID="" ...
  <mpqf:Query>
    <mpqf:Input immediateResponse="true">
      <mpqf:QueryCondition>
        <mpqf:Condition xsi:type="mpqf:QueryByMedia" matchType="similar">
          <mpqf:MediaResource resourceID="de.uop.dimis.air.adapters.LireAdapter">
            <mpqf:MediaResource>
              <mpqf:MediaUri>http://any.uri.com</mpqf:MediaUri>
            </mpqf:MediaResource>
          </mpqf:MediaResource>
        </mpqf:Condition>
      </mpqf:QueryCondition>
    </mpqf:Input>
  </mpqf:Query>
</mpqf:MpegQuery>
```
A typical response may be:

```
HTTP/1.1 200 OK
Content-Length: 738
Content-Type: text/plain;charset=UTF-8

<?xml version="1.0" encoding="UTF-8"?>
<mpqf:MpegQuery>
  <mpqf:Query>
    <mpqf:Output>
      <mpqf:GlobalComment>This is the message from the server</mpqf:GlobalComment>
      <mpqf:ResultItem recordNumber="001" rank="1" confidence="1.0">
        <mpqf:MediaResource>http://www.mpeg7qf/db/image/19701221.jpg</mpqf:MediaResource>
      </mpqf:ResultItem>
      <mpqf:ResultItem recordNumber="002" rank="2" confidence="0.99">
      </mpqf:ResultItem>
    </mpqf:Output>
  </mpqf:Query>
</mpqf:MpegQuery>
```

5.3.1.3 **JAVA Interface**

In the following a detailed description on how to access the software modules and interfaces of the QueryBroker is provided. 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.

As already mentioned "data base connectors" or service interfaces need to be implemented in order to be able to register and access data repositories. A description on how to realize such a service interface is given in [QueryBroker GE Installation and Administration Guide](#).
**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 \( \text{de.uop.dimis.air.backend.BackendManagement} \). 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.

**(De-)Register a Service**

Executable by BackendManagement.getInstance().registerDatabase(mqt);

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

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <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 \textit{ServiceID}, which is equal to the qualified name of the implementation class. The \textit{DesiredCapabilities} declare which query types the service can handle. In this example the \textit{ExampleService} can handle Query-By-Media. See the MPQF-Standard Annex B.2 for more Query URNs. In order to deregister a service endpoint a MPQF-Register-Message must be sent with an empty list of desired capabilities.
**Java example:**

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

**Creating a Semantic Link**

Executable by `BackendManagement.getInstance().registerSemanticLink(sl)`;

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

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

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

Java example:

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

Frontend Functionalities

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

Querying

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

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

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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<mpqf:MpegQuery mpqfID="" ...>
  <mpqf:Query>
    <mpqf:Input immediateResponse="true">
      <mpqf:QueryCondition>
        <mpqf:Condition xsi:type="mpqf:QueryByMedia" matchType="similar">
          <mpqf:MediaResource resourceID="res01">
            <mpqf:MediaResource>
              <mpqf:MediaUri>http://any.uri.com</mpqf:MediaUri>
            </mpqf:MediaResource>
          </mpqf:MediaResource>
        </mpqf:Condition>
      </mpqf:QueryCondition>
    </mpqf:Input>
  </mpqf:Query>
</mpqf:MpegQuery>
```

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

```
Java example:

// get the query xml file as stream
InputStream stream = ...
// unmarshal the xml file
Unmarshaller u = NamespaceHelper.getInstance().getJAXBContextMpqfJPSearch().createUnmarshaller();
MpegQueryType mpqfQuery = (MpegQueryType) u.unmarshal(stream);
// query the databases
MpqfQueryType response = Broker.getInstance().query(mpqfQuery);
// get the results from the response object
List<ResultItemType> results = response.getQuery().getOutput().getResultItem();
```
Alternatively the same query can be described and forwarded to the QueryBroker via pure Java Code:

*Java example:*

```java
// create mpqf objects with the mpqf object factory
ObjectFactory factory = new ObjectFactory();
MpegQueryType mpqfQuery = factory.createMpegQueryType();
Query query = factory.createMpegQueryTypeQuery();
mpqfQuery.setQuery(query);
InputQueryType input = factory.createInputQueryType();
// activate the synchronous query
input.setImmediateResponse(true);
query.setInput(input);
QueryConditionType conditions = factory.createQueryConditionType();
input.setQueryCondition(conditions);
QueryByMedia qbm = factory.createQueryByMedia();
conditions.setCondition(qbm);
MediaResourceType resource = factory.createMediaResourceType();
qbm.setMediaResource(resource);
MediaLocatorType locator = factory.createMediaLocatorType();
resource.setMediaResource(locator);
locator.setMediaUri("http://any.uri.com");
// query the databases
MpegQueryType response = Broker.getInstance().query(mpqfQuery);
// get the results from the response object
List<ResultItemType> results =
    response.getQuery().getOutput().getResultItem();
```

**Asynchronous Query**

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

*Java example:*

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

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

Java example:

Java example:

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

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

Complex Query Example

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

<?xml version="1.0" encoding="UTF-8"?>
<mpqf:MpegQuery mpqfID="101"
xmlns:mpqf="urn:mpeg:mpqf:schema:2008"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mpeg:mpqf:schema:2008
mpqf_semantic_enhancement.xsd">
    <mpqf:Query>
      <mpqf:Input>

D.6.4.2: FIWARE User and Programmers Guide
<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>
Query Execution Tree Evaluation
The query aggregator evaluates the query execution plan (QEP). The result of this evaluation is a number of results that will later be returned to the querying client. Every leaf of the QEP has a reference to a sub-query, which includes one or many service executions, so the results from the sub-query are accessible. For the evaluation an iterator driven model is used. Every node has a next and hasNext method to get the next result item from its children. Since the QEP can be a DAG, hasNext and next need a parameter to decide on which path in the DAG they are called, to be able to choose the correct iterator. HasNext checks if there is a next result item in the local result list. If there is one, next can call this item. If not, hasNext tries to query its child nodes for a next valid result item. If this is possible the new item will be appended in the local result list. Only if no successor can be computed hasNext will return false. So make always sure to call hasNext before next.

There are blocking, half blocking and none blocking operators. A blocking operator needs all results from its children to decide which result will returned next. The SortBy operator is a blocking operator. An operator is half blocking, if it doesn’t need all results from every child. The AND operator is implemented in such a way. None blocking operators like LIMIT can forward results without knowing every other possible result. Some operators have to merge results. If two results are equal (according to the specific semantic link) they must be merged. Merging operators are for example AND and OR. Merging two results means that one result is augmented with additional information from the second result. No information is overwritten.

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

Java example:

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

        // ... do your program logic with the query conditions ...
    }
}
```
// create a result container for the computed results
ObjectFactory mpqfObjFac = new ObjectFactory();
MpegQueryType result = mpqfObjFac.createMpegQueryType();
Query qry = mpqfObjFac.createMpegQueryTypeQuery();
result.setQuery(qry);
OutputQueryType oqt =
mpqfObjFac.createOutputQueryType();
qry.setOutput(oqt);
List<ResultItemType> resultItems = oqt.getResultItem();

// for each result of the service endpoint create a
// result item and add it
// to the results list
for (...) {
    ResultItemType resultItem =
mpqfObjFac.createResultItemType();
    resultItems.add(resultItem);
    Description description =
mpqfObjFac.createResultItemTypeDescription();
    resultItem.getDescription().add(description);
    JPSearchCoreType coreType =
jpsearchObjFac.createJPSearchCoreType();
    description.getContent().add(coreType);

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

return result;
We assume that a second service with the identifier ‘de.uop.dimis.air.SecondService’ gets registered, too. The two databases have the semantic link between the fields ‘identifier’ (ExampleService) and ‘title’ (SecondService).
Now the xml files are loaded and transferred to the QueryBroker. This has to be done only once for initialization. After these steps the QueryBroker is available for query requests.

**Java example:**

```java
// Initialize the QueryBroker and register the service (e.g. in a main-method)
// get the register query xml file as stream
InputStream stream =
    ExampleService.class.getResourceAsStream("RegisterExampleService.xml");
// unmarshal the xml file
Unmarshaller u =
    NamespaceHelper.getInstance().getJAXBContextMpqfJPSearch().createUnmarshaller();
MpegQueryType mpqfQuery = (MpegQueryType) u.unmarshal(stream);
// register the database
BackendManagement.getInstance().registerDatabase(mpqfQuery);
// get the semantic link xml file as stream
InputStream stream =
    ExampleService.class.getResourceAsStream("SemanticLinks.xml");
// unmarshal the xml file
Unmarshaller u =
    NamespaceHelper.getInstance().getJAXBContextsSemanticLinks().createUnmarshaller();
KeyMatchesType link = (KeyMatchesType) ((JAXBElement<?>) u.unmarshal(stream)).getValue();
// register the semantic link
BackendManagement.getInstance().registerSemanticLink(link);
// the querybroker is now ready for queries
```

5.4 References

<table>
<thead>
<tr>
<th>[JAXB]</th>
<th>Java Architecture for XML Binding (JAXB), Metro Projekt, <a href="http://jaxb.dev.java.net/">http://jaxb.dev.java.net/</a></th>
</tr>
</thead>
</table>
6 Metadata Preprocessing - User and Programmers Guide

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

6.1 Introduction

In the following, a detailed description on how to use the functionality and interfaces of the Metadata Preprocessing GE as realized by the MetadataProcessor is provided. First, a comprehensive overview of the functionality of the Metadata Processor is given. This is further illustrated by a quite generic example use case, which should be relevant for many quite different applications areas. The respective API calls for this example usage are further detailed, which completes the User Guide. The Programmers Guide mainly consists of an overview of the RESTful API calls and an API reference for this GE.

6.2 User Guide

6.2.1 Functionality

The functionality of the MetadataProcessor is illustrated in the following figure.

![Figure. Functionality of MetadataProcessor.](image)

The functionality is realized by a filter&pipe architecture. Metadata streams are received through the inbound metadata interface. This interface also performs the depacketization of the payload data. The depacketized metadata is transformed and/or filtered. This processing is the core of the MetadataProcessor. Note that transformation and filtering can be performed jointly. The processed metadata is packetized and the resulting stream is sent to one or more receivers. Note that each processing unit can only connect one source (i.e., a single input.
stream) but multiple sinks (i.e., multiple output streams). However, the MetadataProcessor can instantiate and manage multiple processing units.

The MetadataProcessor can be configured via an API. Input and output streams can be connected and disconnected. Furthermore, the processing engine (i.e., metadata transformation and metadata filtering) can be configured.

6.2.2 Example usage

The most important functionality of the MetadataProcessor is illustrated by an example, which can be applied quite generically to a wide range of usage scenarios with only minor changes. The following figure (i.e., UML sequence diagram) depicts the message flow of this example.

Figure. Example usage of MetadataProcessor.

The following steps are performed in the illustrated example.
1. An application (or a user of the MetadataProcessor) first creates an processing unit ('mddp_instance7') by sending a `createInstance` command to the MetadataProcessor ('metadata_preprocessing_ge').

```
POST //198.51.100.24/mdp/instances HTTP/1.1
Accept: application/xml
```

If the creation of the processing unit is successful, the following response is received.

```
HTTP/1.1 201 Created
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<instanceID>7</instanceID>
```

2. The processing unit is configured by sending a `configureInstance` command. Note that a specific processing unit is referenced in the resource path of the HTTP request.

```
PUT //198.51.100.24/mdpp/instances/7/config HTTP/1.1
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<configurationInstance>
  <source>
    <sourceURI>rtsp://203.0.113.1/stream1</sourceURI>
  </source>
  <processing>
    <plugin position="1" type="xslt">
      <xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:template match="/person_list">
          <object_list>
            <xsl:apply-templates />
          </object_list>
        </xsl:template>
      </xsl:stylesheet>
    </plugin>
  </processing>
</configurationInstance>
```
With this command, an XSLT is configured as a plugin by specifying the XSLT stylesheet. In addition, also the source is configured, i.e., an RSTP server is specified by its URL. If the configuration is successful, the following response is received.

HTTP/1.1 200 OK

3. In order to connect a sink, i.e., an RTSP receiver, the application sends an `addSink` command to the MetadataProcessor. Note that the application could also connect itself as an RTSP receiver (not illustrated in this example).

POST //198.51.100.24/mdp/instances/7/sinks HTTP/1.1
Content-Type: application/xml
Accept: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<configurationSink>
  <sinkURI>http://192.0.2.11/metadata1</sinkURI>
</configurationSink>

If connecting the sink is successful, the following response is received.

HTTP/1.1 201 Created
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<sinkID>102</sinkID>

The sink specified in the `addSink` command receives the URL of the RTSP server component of the MetadataProcessor by the following message.
The sink connects to this server in order to receive the processed metadata. Note that more than one sinks can be connected (not illustrated in this example).

4. The application starts the processing of the processing unit of the MetadataProcessor by sending a `startInstance` command.

```
PUT //192.0.2.11/metadata1 HTTP/1.1
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<configurationListener>
  <streamURI>rtsp://198.51.100.24/mdp/7/stream1</streamURI>
</configurationListener>
```

If starting the transformation process is successful, the following response is received.

```
HTTP/1.1 200 OK
```

5. After metadata processing is finished, the processing unit is stopped by sending a `stopInstance` command.

```
PUT //198.51.100.24/mdp/instances/7?action=start HTTP/1.1
```

If stopping the transformation process is successful, the following response is received.

```
HTTP/1.1 200 OK
```

6. The application destroys the processing unit (e.g., because it is not needed for later transformation purposes) by sending a `destroyInstance` command.

```
DELETE //198.51.100.24/mdp/instances/7 HTTP/1.1
```

Note that disconnecting of source and sink(s) does not have to be done separately since this is handled automatically in case a processing unit is destroyed. If the deletion of the processing unit is successful, the following response is received.

```
HTTP/1.1 200 OK
```

### 6.3 Programmers Guide

The Metadata Preprocessing GE, i.e., the MetadataProcessor as of Release 2.2, is meant to be used as a web service. Therefore, programming against this GE involves using its RESTful API.
An overview of the API is given and the respective HTTP requests of the API are further detailed.

### 6.3.1 API overview

The following figure gives an overview of the RESTful API of the Metadata Preprocessing GE, i.e., the MetadataProcessor.
### 6.3.2 API reference

The following table gives a complete reference of the Metadata Preprocessing GE API. Concrete examples for using the RESTful API are given in the User Guide above and in the Metadata Preprocessing Open RESTful API Specification (PRELIMINARY).

<table>
<thead>
<tr>
<th>Verb</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>//{serverRoot}/{assetName}/version</td>
<td>getVersion: returns the current version of the Metadata Preprocessing GE realization/asset (e.g., MetadataProcessor)</td>
</tr>
<tr>
<td>GET</td>
<td>//{serverRoot}/{assetName}/instances</td>
<td>listInstances: lists all instances (i.e., processing units) of the Metadata Preprocessing GE</td>
</tr>
<tr>
<td>POST</td>
<td>//{serverRoot}/{assetName}/instances</td>
<td>createInstance: creates an instance (i.e., a processing unit) of the Metadata Preprocessing GE</td>
</tr>
<tr>
<td>DELETE</td>
<td>//{serverRoot}/{assetName}/instance/{instanceID}</td>
<td>destroyInstance: destroys a specific instance (i.e., processing unit)</td>
</tr>
<tr>
<td>PUT</td>
<td>//{serverRoot}/{assetName}/instance/{instanceID}?action=start</td>
<td>startInstance: starts the processing of the processing unit</td>
</tr>
<tr>
<td>PUT</td>
<td>//{serverRoot}/{assetName}/instance/{instanceID}?action=stop</td>
<td>stopInstance: stops the processing of the processing unit</td>
</tr>
<tr>
<td>GET</td>
<td>//{serverRoot}/{assetName}/instances/{instanceID}/config</td>
<td>getConfig: returns the configuration of an existing processing unit</td>
</tr>
<tr>
<td>PUT</td>
<td>//{serverRoot}/{assetName}/instances/{instanceID}/config</td>
<td>configureInstance: configures an existing processing unit</td>
</tr>
<tr>
<td>GET</td>
<td>//{serverRoot}/{assetName}/instances/{instanceID}/listSinks</td>
<td>listSinks: lists all connected sinks</td>
</tr>
<tr>
<td>Method</td>
<td>URI</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>POST</td>
<td><code>//{serverRoot}/{assetName}/instances/{instanceID}/sinks</code></td>
<td><strong>addSink</strong>: adds a sink for receiving the transformed/filtered metadata (e.g., another GE)</td>
</tr>
<tr>
<td>GET</td>
<td><code>//{serverRoot}/{assetName}/instances/{instanceID}/sinks/{sinkID}</code></td>
<td><strong>getSinkInfo</strong>: returns the information about a specific sink</td>
</tr>
<tr>
<td>DELETE</td>
<td><code>//{serverRoot}/{assetName}/instances/{instanceID}/sinks/{sinkID}</code></td>
<td><strong>removeSink</strong>: removes a specific sink</td>
</tr>
</tbody>
</table>
7 Semantic Application Support - Users and Programmers Guide

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

7.1 Introduction

Welcome the User and Programmer Guide for the Semantic Application Support Generic Enabler. The current version of this generic enabler has two main parts: An open source isolated client for ontology engineering [1] and server-side component and services for ontology storage, versioning and management. In future versions the client and the server functionality will be linked together by the development of a NeOn Toolkit plugin.

7.2 User guide

7.2.1 Overview of the GE

The Semantic Web Application Support GE 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). Following this idea, Internet will evolve into a machine understandable Web, therefore increasing its exploitation capabilities. The Semantic Web has focused the efforts of many researchers, institutions and IT practitioners, and received a fair amount of investment from European and other governmental bodies. As a result of these efforts, a large amount of mark-up languages, techniques and applications, ranging from semantic search engines to query answering system, have been developed. Nevertheless the adoption of Semantic Web from the IT industry is still following a slow and painful process.

In recent years, several discussions had taken place to find out the reasons preventing Semantic Web paradigm adoption. There is a general agreement that those reasons range from technical (lack of infrastructure to meet industry requirements in terms of scalability, performance, distribution, security, etc.) to engineering (not general uptake of methodologies, lack of best practices and supporting tools), and finally commercial aspects (difficulties to penetrate in the market, lack of understanding of the main strengths and weaknesses of the semantic technologies by company managers, no good sales strategies, etc.).

The Semantic Application Support enabler addresses part of the abovementioned problems (engineering and technical) from a data management point of view, by providing:

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

### 7.2.2 Semantic Application Support GE Architecture

Semantic Application Support GE can be split in a client-side Engineering Environment and a server-side Infrastructure. Figure SWAS-1 presents the SWAS Infrastructure architecture.

![SWAS Infrastructure architecture](image)

**Figure SWAS-1: SWAS Infrastructure architecture**

As it is shown in the diagram, it follows a typical three layer Java Enterprise Architecture. Components included in business and presentation layers are JEE based. In the data layer, two components can be found:

- A relational database, that will be used by Ontology Registry to store ontology documents loaded into the GE.
- A Knowledge Base providing OWL-2RL support. This Knowledge Base will be used by ontology and workspace registries to store ontology and workspace related metadata and by managing, querying and publishing modules to provide their functionality.

Business components will interact with data layer components by means of two different mechanisms. To interact with the relational database, business components will use JPA (Java Persistence API) that make business components database system independent. Unfortunately such an abstraction mechanism is not available for knowledge base required interaction so business components interacting with the knowledge base will be knowledge base implementation dependant. In Semantic Web Application Support reference implementation,
the combination of Sesame and OWLIM has been chosen as knowledge base implementation. Knowledge base independence feature will be study for future releases.

Business Layer contains following components:

- Ontology registry that manages ontologies loaded into the system and its related metadata. Operations such as retrieving / uploading ontology, retrieving / uploading metadata, etc would be provided by this component. A description of methods provided for FI-WARE first release can be found in Backend Functionality section.

- Workspace registry that manages workspaces and their related metadata created by users to be used by their semantic enable applications. Operations such as creating / deleting a workspace, listing ontologies loaded into the workspace, etc would be provided by this component. Description of methods belonging to this component will be described in future FI-WARE releases.

- Publishing module that allow user to publish data into the GE. Data can be either ontologies or RDF serialized content. In case of ontologies, publishing module will rely on ontology registry functionality. In case of RDF serialized content, publishing module will store the content in proper knowledge base workspace in collaboration with workspace registry. In both cases publishing module will update subscription module if needed. Description of methods belonging to this component will be described in future FI-WARE releases.

- Managing module that allow users to monitor the status of the GE. Operations such as retrieving a list of available ontologies, retrieving a list of subscriptions, etc will be provided by this module. Managing module will rely on the rest of business components to provide its functionality. Description of methods belonging to this component will be described in future FI-WARE releases.

- Subscription module that allows users to subscribe to events produced in the GE. Operations such as subscribing to ontology updates or workspace modifications will be provided by this module. Description of methods belonging to this component will be described in future FI-WARE releases.

- Querying module that allows users to query their workspace following SPARQL Query Protocol. Description of methods belonging to this component will be described in future FI-WARE releases.

In order to provide GE functionality in a platform independent way, several Rest APIs will be developed. In this first FI-WARE release, a subset of methods belonging to publish and managing APIs will be provided. Therefore, clients or presentation layer applications will interact with business components by means of HTTP requests / responses.

SWAS Engineering Environment provides comprehensive support for the ontology engineering life-cycle. Concrete details about SWAS Engineering Environment functionality will be provided in Frontend Functionality section. SWAS architecture is based on Eclipse architecture, a leading development environment providing a technical layer for easy creation of new features and
supported for a huge development community. Figure SWAS-2 shows the SWAS Engineering Environment architecture.

**Figure SWAS-2: SWAS Engineering Environment architecture**

As it shown in the diagram SWAS Engineering Environment is divided into two layers, the SWAS Engineering Environment core and the contributed plug-ins. The SWAS Engineering Core provides the core ontology editing functionality. The contributed plug-ins are extensions that provide extra functionality supporting different phases of the NeOn Methodology.

### 7.2.3 Frontend Functionality

Semantic Web Application Support Engineering Environment functionality is based on the functionality provided by the baseline asset NeOn Toolkit. The NeOn Toolkit is a state-of-the-art, eclipse based, open source multi-platform ontology engineering environment, which provides comprehensive support for the ontology engineering life-cycle. Due to its nature, it wouldn’t be possible to describe all SWAS Engineering Environment provided functionality in a service like manner. Therefore this user guide provides a short overview of the functionality required for the SWAS Engineering Environment. For more information about the NeOn Toolkit consult [NTK_Documentation](#).

Figure SWAS-3 presents an overview of NeOn Toolkit GUI.
The NeOn Toolkit, and therefore the SWAS GE Engineering Environment takes advantage of the paradigm introduced by Eclipse, one of the leading development environments, including:

- Using workspaces, projects, folders and files as containers to organize and store development artifacts.
- Using workbench, editors, views and perspectives to provide functionality to the user by means of GUI.

Therefore, most of the functionality provided by SWAS Engineering Environment is provided as editors, views and perspectives. Figure SWAS-4 presents the Ontology navigation perspective.
Under this perspective users are able to manage their projects and ontologies, creating or removing projects, loading or creating new ontologies, etc. In the scope of a given ontology, users are able to manage (adding, removing, etc) main ontology contents such as classes, object properties and data properties. Once selected, ontology contents can be edited by means of a proper editor. Figure SWAS-5 presents the class editor.
Class editor is composed of four tabs:

- Class restrictions tab that allow the user to modify restrictions applicable to the class.
- Taxonomy tabs that allow the user to modify the class ancestors, successors or siblings.
- Annotation tab that allows the user to annotate the class with textual descriptions.
- Source tab that presents the user the OWL code generated for the described class.

Data property and object property editor provide similar functionality for data and object properties. Finally, views present additional information about the items selected in the ontology navigation perspective. Figure SWAS-6 presents the range view.
Range view presents for each class, the set of object properties that has the selected class as range. As mentioned in Semantic Application Support GE Architecture section, Engineering Environment functionality can be extended by means of plug-ins. Nowadays there are more than 30 active plug-ins for NeOn Toolkit covering a wide range of functionality covering several steps of the NeOn Methodology. Some of this plug-ins functionality may inspire Engineering Environment functionality in the future if needed.

7.2.4 Backend Functionality

This section describes the functionality provided by the SWAS GE enabler as service invocation methods for both human or computer agents. As described in Architecture of the GE FiWARE.ArchitectureDescription.Data.SemanticSupport, this functionality is accessible by means of Web API (also known as RESTful services). In this first FIWARE release the following subset of methods belonging to publishing and managing rest APIs has been provided:

- **Publishing Rest API.**
  - *Get ontology version:* Retrieves from the GE the ontology document identified by a given ontology IRI 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. 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 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. 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 and version IRI. To invoke the operation, a PUT http request should be sent to `http://<ge url location>/ontology-registry/ontologies/<ontology IRI>/<version IRI>` with an file 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 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. 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 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 comply 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. To invoke the operation, a PUT http request should be sent to http://<ge url location>/ontology-registry/metadata/<ontology IRI> with an file attachment including the metadata RDF/XML serialization. Metadata uploaded must comply to OMV (Ontology metadata vocabulary).

### Managing Rest API

- **List ontologies**: Retrieves an XML document containing the list of ontology documents and their versions loaded into the GE. To invoke the operation, a GET http request should be sent to http://<ge url location>/ontology-registry/mgm/list. As a result, an xml encoding the requested information will be sent as response.

- **List ontology versions**: Similar to List ontologies. Retrieves an XML document containing the versions of an ontology document identified by a given ontology IRI loaded into the GE. To invoke the operation, a GET http request should be sent to http://<ge url location>/ontology-registry/mgm/<ontology IRI>. As a result, an xml encoding the requested information will be sent as response.

### Workspaces Management Rest API

- **List Workspaces**: Retrieves an XML document which contains a list of all workspaces included in the server, a GET http request should be sent to
http://<ge url location>/semantic-workspaces-service/rest/workspaces/mgm/list. As result, a 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]. As a result, an xml 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]. As a result, 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. As result, a xml 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/. As result, a xml 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]. As result, 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. As result, 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]. As result, 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. As result, 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. As result, 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]. As result, a XML 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. As result, a xml 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. As result, a xml 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]. As result, 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. As result, 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. As result, 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. As result, a XML encoding the result of the operation.

### 7.3 Programmers Guide

#### 7.3.1 Development of NeOn Toolkit plugings

The NeOn Toolkit web site provides information for developers of NeOn Toolkit plugins. For more information check the following link [NTK_Developers_Guide](#).
7.3.2 Invocation of services

All methods described in the users guide above are RESTful services and therefore can be invoked by means of regular HTTP requests or using a REST API.

As a matter of example, the following Java code shows how to invoke Upload ontology version by using Jersey API.

```java
File file = new File("Directory");
if(!file.isDirectory())
    throw new Exception("Invalid starting point");
File[] contexts = file.listFiles();
// Iterating over folders
for(int i = 0; contexts != null && i < contexts.length; i++){
    // Iterating over files
    File[] ontologies = contexts[i].listFiles();
    for(int j = 0; ontologies != null && j < ontologies.length; j++){
        if(ontologies[j].isFile()){
            System.out.println(ontologies[j].getAbsolutePath());
            String ontologyIRI = ontologies[j].getAbsolutePath().substring(ontologies[j].getAbsolutePath().lastIndexOf('/') + 1);
            String versionIRI = "";
            versionIRI = "1.0";
            FormDataMultiPart form = new FormDataMultiPart();
            ContentDisposition cd = form.getContentDisposition();
            FileDataBodyPart fdp = new FileDataBodyPart(ontologyIRI, ontologies[j], MediaType.APPLICATION_OCTET_STREAM_TYPE);
            form.bodyPart(fdp);
            Client c = Client.create();
            String resourceName = "http://localhost:8080/ontology-registry-service/webresources/ontology-registry/" + ontologyIRI;
            if(versionIRI != null && versionIRI.compareTo("") != 0)
                resourceName = resourceName + "/" + versionIRI;
            System.out.println("resourceName: " + resourceName);
            WebResource r = c.resource(resourceName);
            String response = 
```
Some other methods (those ones who rely on GET requests) can just by invoked by introducing a simple URL into a web browser. Figure SWAS-7 shows Get ontology version invocation using this mechanism.

As a matter of example of the method GetOntologyVersion see below the request and expected response of the service:

<table>
<thead>
<tr>
<th>Verb</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/ontologies/{ontologyIRI}/{versionIRI}</td>
<td>Retrieves the ontology field identified by a given ontology IRI and version IRI</td>
</tr>
</tbody>
</table>

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

Figure SWAS-7: Invoking Get ontology version using web browser.
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/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/"
<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:subPropertyOf rdf:resource="http://www.tatoo-fp7.eu/tatooweb/merm#dateProvided"/>
8 Middleware - User and Programmers Guide

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

8.1 Introduction

FI-WARE Middleware GE (code named KIARA) is a new middleware based on the Data Distribution Service (DDS) specifications, an OMG Standard defining the API and Protocol for high performance publish-subscribe middleware, and eProsima RPC over DDS, an Remote Procedure Call framework using DDS as the transport and based on the ongoing OMG RPC for DDS standard. A quick DDS introduction is provided here.

In contrast to other GEs, the FI-WARE Middleware GE is not a standalone service running in the network, but a set of compile-/runtime tools and a communication library to be delivered with the application.

In this first release of the middleware for FI-WARE (R2.2), we are providing the basic assets, DDS and eProsima RPC for DDS.

eProsima RPC for DDS has been updated to include several planned features for KIARA, such as asynchronous calls and a high performance dispatching agent.

In subsequent releases we will add the following features:

- Improved IDL / Declarative API
- Direct usage of application native types (dynamic mapping to IDL)
- Run time optimization with embedded compiler/interpreter (LLVM)
- RESTful Web Service compatibility (XML/JSON marshalling)
- Transport protocol / mechanism negotiation
- Extended transport protocols and mechanisms
- Advanced security features (Security policies)
- Support for modern Languages (C & C++ in first release)

8.1.1 Software Options

Two different implementations of DDS can be selected:

1. RTI DDS 5.0 or later (Free, Open Community Source License)
2. OpenDDS 3.4.1 or later (Free, Open Source License)

We recommend RTI DDS because its performance and ease of use.
For the remote procedure calls framework use

- eProsima RPC for DDS 1.0 (free and Open Source License).

8.2 User Guide

These products are programmer tools, therefore there is no different user and programmers guides.

See programmer's guide section to browse the available documentaiton.

8.3 Programmer Guide

All the available documentation for this release is published online.

8.3.1 DDS

8.3.1.1 RTI DDS

RTI DDS documentation is published online here, including User Manual, API Reference, Best Practices, Examples...

A “community portal” is available, with Forums, Knowledge base and more.

8.3.1.2 OpenDDS

Open DDS documentation is published online here, including a Developer’s Guide, API Reference, examples...

The OpenDDS Web contains also articles, faqs, developer tools, etc.

8.3.2 eProsima RPC for DDS

eProsima RPC for DDS is published online in the Product Page, including a User Manual and the API Reference:

- User Manual (PDF)
- API Reference (Doxygen)

If you need assistance, please contact eProsima Support