



SEVENTH FRAMEWORK PROGRAMME

Specific Targeted Research Project

Project Number:	FP7-SMARTCITIES-2013(ICT)
Project Acronym:	VITAL
Project Number:	608682
Project Title:	Virtualized programmable InTerfAces for innovative cost-effective IoT depLoyments in smart cities

D6.1.2 Integrated VITAL Framework

Document Id:	VITAL-D612-100316-Draft
File Name:	VITAL-D612-100316-Draft.pdf
Document reference:	Deliverable 6.1.2
Version:	Draft
Editors:	Miguel Angel Mateo. Elisa Hermann
Organisation:	Atos
Date:	10 / 03 / 2016
Document type:	Deliverable
Security:	PU (Public)

Copyright © 2016 VITAL Consortium

PROPRIETARY RIGHTS STATEMENT

This document contains information, which is proprietary to the VITAL Consortium.
Neither this document nor the information contained herein shall be used, duplicated or communicated by any means to any third party, in whole or in parts, except with prior written consent of the consortium

DOCUMENT HISTORY

Rev.	Author(s)	Organisation(s)	Date	Comments
V01	Elisa Herrmann	ATOS	2015/11/26	Initial version of the document
V02	Zeeshan Jan	NUIG	2016/01/07	DMS updated in all related sections
V03	Lorenzo Bracco, Paola Dal Zovo	REPLY	2016/01/15	Updates to security sections
V04	Miguel Angel Mateo	ATOS	2016/02/11	Section 1 updated, figures 12 and 16 redrawn.
V05	Miguel Angel Mateo	ATOS	2016/02/18	Section 3 updated
V06	Lorenzo Bracco, Paola Dal Zovo	REPLY	2016/02/18	Update HI Reply and security related sections
V07	Lorenzo Bracco, Paola Dal Zovo	REPLY	2016/02/25	Update security related sections to current status
V08	Angelos Lenis	SILO	2016/01/03	Technical Review
V09	Miguel Angel Mateo	ATOS	2016/09/03	Comments addressed
V10	Zeeshan Jan	NUIG	08/03/2016	DMS section updated.
V11	Aqeel Kazmi	NUIG	08/03/2016	Updates to DMS section
V12	Martin Serrano	NUIG	10/03/2016	Circulated for Approval
Draft	Martin Serrano	NUIG	10/03/2016	EC Submitted

OVERVIEW of UPDATES/ENHANCEMENTS OVER D6.1.1

Section	Description
Section 1	Introductory section. Updates to the description of the summary and scope of the second version of the deliverable.
Section 2	Alignment of the section to the status of the VITAL architecture at the time of writing the document.
Section 3	Updated the whole section
Section 4	Updated DMS and Security, figures 12 and 14 format unification
Section 5	Updated DMS and Security
Section 6	Updated DMS and Security
Section 7	Updated DMS and Security

TABLE OF CONTENTS

1	INTRODUCTION	8
1.1	SCOPE	8
1.2	AUDIENCE.....	8
1.3	SUMMARY.....	8
1.4	STRUCTURE.....	9
2	ARCHITECTURE.....	10
2.1	OVERVIEW OF VITAL ARCHITECTURE.....	10
2.2	IOT DATA ADAPTER.....	11
2.3	DATA MANAGEMENT SERVICES (DMS).....	12
2.4	DISCOVERY SERVICE	12
2.5	FILTERING SERVICE.....	13
2.6	ORCHESTRATION SERVICE.....	13
2.7	COMPLEX EVENT PROCESSING (VITALCEP)	13
2.8	DEVELOPMENT TOOLS.....	15
2.9	MANAGEMENT PLATFORM.....	15
2.10	SECURITY	15
3	DEPLOYMENT	17
3.1	IOT DATA ADAPTER.....	19
	• IoT Data Adapter	19
	• PPI for IoT Data Adapter	20
3.2	SECURITY GATEWAY FOR PPIs	20
3.3	DATA MANAGEMENT SERVICES (DMS).....	21
3.4	DISCOVERY SERVICE	21
3.5	FILTERING SERVICE.....	22
3.6	ORCHESTRATION SERVICE.....	22
3.7	COMPLEX EVENT PROCESSING (VITALCEP)	23
	• Complex Event Processing Engine	23
	• CEP manager plus interfaces.....	23
3.8	DEVELOPMENT TOOLS	23
3.9	MANAGEMENT PLATFORM.....	24
3.10	SECURITY	25
3.11	PPI FOR CAMDEN FOOTFALL	26
3.12	PPI FOR HI REPLY ISTANBUL TRAFFIC.....	27
4	FUNTIONAL MODULES INTERACTIONS	27
4.1	IOT DATA ADAPTER.....	27
4.2	DATA MANAGEMENT SERVICES (DMS).....	30
4.3	DISCOVERY SERVICE	31
4.4	FILTERING SERVICE.....	31
4.5	ORCHESTRATION SERVICE.....	32
4.6	COMPLEX EVENT PROCESSING (VITALCEP)	33
	4.6.1 VITALCEP instance creation	33
	4.6.2 VITALCEP instance update	33
	4.6.3 VITALCEP instance deletion.....	34
4.7	DEVELOPMENT TOOLS.....	34
4.8	MANAGEMENT TOOLS.....	35
4.9	SECURITY	35

5	LICENSING	37
5.1	IOT DATA ADAPTER.....	37
5.2	DATA MANAGEMENT SERVICES (DMS).....	37
5.3	DISCOVERY SERVICE	37
5.4	FILTERING SERVICE.....	38
5.5	ORCHESTRATION SERVICE.....	38
5.6	COMPLEX EVENT PROCESSING (VITALCEP)	38
5.7	DEVELOPMENT TOOLS.....	39
5.8	MANAGEMENT TOOLS.....	39
5.9	SECURITY	40
6	USE.....	41
6.1	IOT DATA ADAPTER.....	41
6.2	DATA MANAGEMENT SERVICES (DMS).....	42
6.3	DISCOVERY SERVICE	48
6.4	FILTERING SERVICE.....	51
6.5	ORCHESTRATION SERVICE.....	53
6.6	COMPLEX EVENT PROCESSING (VITALCEP)	62
6.7	DEVELOPMENT TOOLS.....	64
6.8	MANAGEMENT TOOLS.....	64
6.9	SECURITY	69
7	NEXT STEPS.....	73
7.1	IOT DATA ADAPTER.....	74
7.2	DATA MANAGEMENT SERVICE (DMS).....	74
7.3	DISCOVERY SERVICE	74
7.4	FILTERING SERVICE.....	74
7.5	ORCHESTRATION SERVICE.....	74
7.6	COMPLEX EVENT PROCESSING (VITALCEP)	75
7.7	DEVELOPMENT TOOLS.....	75
7.8	MANAGEMENT TOOLS.....	75
7.9	SECURITY	76
8	CONCLUSION.....	76

LIST OF FIGURES

FIGURE 1	VITAL FUNCTIONAL ARCHITECTURE	11
FIGURE 2	VITAL CEP FUNCTIONAL ARCHITECTURE.....	14
FIGURE 3	VITAL SECURITY LAYER	17
FIGURE 4	IoT SYSTEM PROVIDER REQUESTS FROM IoT DATA ADAPTER TO REFRESH ALL RELATED METADATA.	28
FIGURE 5	IoT DATA ADAPTER PULLS PERIODICALLY OBSERVATIONS MADE BY A SENSOR.	29
FIGURE 6	IoT SYSTEM PUSHES NEW OBSERVATIONS MADE BY A SENSOR TO IoT DATA ADAPTER.	30
FIGURE 7	IoT DATA ADAPTER PUSHES DATA INTO DMS.	30
FIGURE 8	DISCOVERY SERVICE.	31
FIGURE 9	FILTERING SERVICE.	31
FIGURE 10	ORCHESTRATION SERVICE.....	32
FIGURE 11	VITALCEP INSTANCE CREATION.....	33
FIGURE 12	VITALCEP INSTANCE UPDATE.	33
FIGURE 13	VITACEP INSTANCE DELETION.	34
FIGURE 14	MANAGEMENT TOOL INTERACTIONS.	35
FIGURE 15	AN APPLICATION OR VITAL MODULE LOGS IN A USER	36
FIGURE 16	ENDPOINTS PROTECTION	36
FIGURE 17	ON DEMAND PERMISSIONS EVALUATION TO AUTHORIZE ACCESS TO DATA	36

FIGURE 18 IOT DATA ADAPTER WEB INTERFACE.....	42
FIGURE 19 INSERTSYSTEM TO DMS.....	43
FIGURE 20 INSERTSERVICE TO DMS.....	43
FIGURE 21 INSERTSENSOR TO DMS.....	44
FIGURE 22 INSERTOBSERVATION TO DMS.....	44
FIGURE 23 QUERYSYSTEM FROM DMS.....	45
FIGURE 24 QUERYSERVICE FROM DMS.....	46
FIGURE 25 QUERYSENSOR FROM DMS.....	47
FIGURE 26 QUERYOBSERVATION FROM DMS.....	48
FIGURE 27 RESULT FOR CONNDMS FUNCTION.....	49
FIGURE 28 DISCOVERY SERVICE DESCRIPTION.....	50
FIGURE 30 RESULT FOR GETICOS FUNCTION.....	51
FIGURE 31 FILTERING SERVICE DESCRIPTION.....	52
FIGURE 32 RESULT OF FILTERING ACCORDING TO THRESHOLD.....	53
FIGURE 33 LIST OF OPERATIONS IN THE VITAL ORCHESTRATOR UI.....	54
FIGURE 34 GUI FOR UPDATING OPERATIONS IN THE VITAL ORCHESTRATOR UI.....	55
FIGURE 35 RESULT OF EXECUTING AN OPERATION IN THE VITAL ORCHESTRATOR UI.....	56
FIGURE 36 GUI FOR UPDATING WORKFLOWS IN THE VITAL ORCHESTRATOR UI (1).....	57
FIGURE 37 GUI FOR UPDATING WORKFLOWS IN THE VITAL ORCHESTRATOR UI (2).....	58
FIGURE 38 WORKFLOW EXECUTION RESULT IN THE VITAL ORCHESTRATOR UI.....	59
FIGURE 39 LIST OF WORKFLOWS IN THE VITAL ORCHESTRATOR UI.....	60
FIGURE 40 LIST OF DEPLOYED META SERVICE IN THE VITAL ORCHESTRATOR UI.....	61
FIGURE 41 VIEW OF A SINGLE META SERVICE IN THE VITAL ORCHESTRATOR UI.....	61
FIGURE 42 RETRIEVE VITALCEPs INSTANCES.....	62
FIGURE 43 RETRIEVE A VITALCEP INSTANCE.....	62
FIGURE 44 UPDATE VITALCEP INSTANCE.....	63
FIGURE 45 DELETE VITALCEP INSTANCE.....	63
FIGURE 46 VITAL DEVELOPMENT AND DEPLOYMENT ENVIRONMENT.....	64
FIGURE 47 SERVICES DASHBOARD.....	65
FIGURE 48 IOT SYSTEM DASHBOARD.....	66
FIGURE 49 LIST OF ALL SENSORS OF THE INFRASTRUCTURE.....	67
FIGURE 50 LIST OF SENSORS IN A MAP, FOR SENSORS WITH LOCATION DATA AVAILABLE.....	68
FIGURE 51 OVERVIEW OF A SINGLE SENSOR.....	69
FIGURE 52 LOG IN A USER AND OBTAIN THE SSO TOKEN.....	70
FIGURE 53 REQUEST FOR A POLICY EVALUATION.....	72
FIGURE 54 SECURITY MANAGEMENT UI.....	73

LIST OF TABLES

TABLE 1 IOT DATA ADAPTER LICENSING.....	37
TABLE 2 DATA MANAGEMENT SYSTEM LICENSING.....	37
TABLE 3 DISCOVERY SERVICE LICENSING.....	37
TABLE 4 IOT FILTERING SERVICE LICENSING.....	38
TABLE 5 ORCHESTRATION SERVICE LICENSING.....	38
TABLE 6 COMPLEX EVENT PROCESSING LICENSING.....	38
TABLE 7 IOT DEVELOPMENT TOOLS LICENSING.....	39
TABLE 8 MANAGEMENT TOOLS LICENSING.....	39
TABLE 9 SECURITY LICENSING.....	40

TERMS AND ACRONYMS

ACL	Access Control List
API	Application programming interface
CEP	Complex Event Processing
COMANTO	Context Management Ontology
COMPOSE	Collaborative Open Market to Place Objects at your Service
CQELS	Continuous Query Evaluation over Linked Stream
CRUD	Create Read Update Delete
DCMI	Dublin Core Metadata Initiative
DCO	Delivery Context Ontology
DOLCE-Lite	Descriptive Ontology for Linguistic and Cognitive Engineering Lite
GEO	Basic Geo Ontology
GPS	Global Positioning System
GSN	Global Sensor Network
hRest	HTML for REST
HTTP	Hypertext Transfer Protocol
ICO	Internet Connected Object
ICT	Information and communication technology
IoT	Internet of Things
IRI	Internationalised Resource Identifier
JSON	JavaScript Object Notation
JSON-LD	JSON for Linked Data
LODE	Ontology for Linking Open Descriptions of Events
MSM	Minimal Service Model
MUO	Measurements Unit Ontology
N3	Notion 3
OAE	Ontology of Adverse Events
OM	Ontology of Units of Measure
OnTraJaCS	Ontology Based Traffic Control Systems
OWL	Web Ontology Language
OWL-S	Web Ontology Language for Web Services
PPI	Platform Provider Interface
PPO	Privacy Preference Ontology
QUDT	Quantities, Units, Dimensions and Data Types Ontologies

RDF	Resource Description Framework
REST	Representational State Transfer
S4AC	Social Semantic SPARQL Security for Access Control
SA-REST	Semantic Annotations for REST
SAWSDL	Semantically Annotated WSDL
SOAP	Simple Object Access Protocol
SOUPA	Standard Ontology for Ubiquitous and Pervasive Applications
SPARQL	SPARQL Protocol and RDF Query Language
SSN	Semantic Sensor Network
SSO	Stimulus-Sensor-Observation pattern
TAC	Triple Access Control
TAO	Trust Assertion Ontology
UO	Units of Measurement Ontology
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTO	Urban Traffic Ontology
VUAI	Virtual Unified Access Interface
W3C	World Wide Web Consortium
WAC	Web Access Control
WADL	Web API Definition Language
WSDL	Web Service Definition Language
WSMO	Web Service Modelling Ontology

1 INTRODUCTION

1.1 Scope

The sixth work package of the VITAL project (WP6) deals with the integration and validation of the VITAL Framework, which is the result of the researching work done in work packages two, three, four and five (WP2, WP3, WP4 and WP5).

D6.1.2 is the second release of the prototype implementation of the integrated VITAL framework, described in this accompanying report. The purpose of the present document is to showcase technical requirements, installation process and use of each VITAL functional module and the interfaces they provide for interworking and also for developing applications. It also offers a view of the overall platform by grouping information written in other deliverables, descriptions and Data Flows diagrams, in order to serve as main reference document for application developers.

The present version, D6.1.2, describes the evolution of VITAL platform towards a system which offers a common way to access data and services of Vital and the underlying IoT platforms under a common umbrella that abstracts application developers from implementing different interfaces to interact with several platforms. This umbrella is composed of interfaces and applications for managing and for implementing new applications.

1.2 Audience

This deliverable is addressed to the following audiences:

- **IoT applications developers and solution providers**, notably solution providers emphasizing on smart city applications using the IoT paradigm. Application developers and solution providers are expected to be interested into the project's general-purpose interfaces for accessing IoT systems.
- **IoT researchers**, notably researchers working on abstract data models and service models for IoT applications. To these researchers, VITAL Framework may serve as a source of information and experimentation for their research.
- **VITAL developers**, notably individuals engaging in the development of the VITAL added-value functionalities and of the VITAL applications. The former will need to understand each functionality and its interfaces in order to make sure that their components/modules support them.

As already outlined, the present deliverable will be consulted by VITAL developers working in WP4, WP5 and WP6 activities, which will be using PPIs and VUALs for accessing IoT data and services in a platform-agnostic manner.

1.3 Summary

This report represents the result of the integration process carried out in VITAL Project, by installing the functional modules in a testbed environment according to their requirements as expressed in this document. These functional components are the result of the work done in the work packages 2, 3, 4, and 5. All these functional

modules form the VITAL Framework which will be evaluated through two use cases, one in Camden Town (London) and the other in Istanbul.

The VITAL Framework has been designed to serve as platform to integrate many different and heterogeneous IoT platforms, while providing their capabilities to new users in a platform agnostic way, allowing VITAL users to use resources and services exposed by VITAL as if were VITAL resources, independently of the source platform.

1.4 Structure

The structure of the document is as follows:

- Section 2 offers an overview of the architecture of VITAL and its main functional modules.
- Section 3 describes the required software components, its dependencies and the procedure to install them.
- Section 4 describes, through data flows, the main interactions between the functional modules.
- Section 5 lists the licensing of software modules used in the VITAL prototype and the final license level of each functional module.
- Section 6 describes how to use each VITAL functional module.
- Section 7 describes the next steps to be performed on each functional module.
- Section 8 is the Conclusion Section.

2 ARCHITECTURE

2.1 Overview of VITAL Architecture

The main objective of the VITAL platform is to support the development, integration, deployment and operation of applications leveraging data from multiple IoT platforms and applications. In order to achieve the integration of different IoT platforms (technology, context, etc.), VITAL platform offers a common interface to access data and services of the relying platforms to provide a unified view to users and application developers.

The process to develop the VITAL platform, aka VITAL Framework, is depicted through the definition of each work package of the project and their timing.

- The first step consists in the study of the reference scenarios that will serve as evaluation of the resulting platform and their requirements. The outcome of this work, performed in the Work Package Two (WP2) **Deliverable [D2.3]**, is a collection of technical requirements and a functional view of the VITAL architecture.
- The second step is to develop a common vocabulary that can be applied to all information (objects, data, services, etc.) participating in the VITAL platform. The adoption of these data models will allow to have a homogeneous view of the entities in the system. These data models have been defined in the VITAL Ontology which is part of the outcome of Work Package three (WP3), **Deliverables [D3.1.1, D3.1.2]**.
- The third step is to specify and develop a set of interfaces to provide access to the VITAL capabilities (ICOs, data, services), and also to the capabilities of the relying platforms, in a platform agnostic way. This is also part of the outcome of Work Package 3 (WP3) **Deliverables [D3.2.1, D3.2.2 and D3.2.3]**.
- The fourth step consist in specifying and developing a set of platform agnostic functionalities which leverage the integration of the relying IoT platforms, enabling their horizontal integration in order to build more complex applications. These added value services are the outcome of Work Package four (WP4), **Deliverables [D4.1.1, D4.1.2, D4.2.1, D4.2.2, D4.3.1 and D4.4.1]**.
- The fifth step aims to provide a set of tools for the management and governance of the VITAL platform and applications built on top (WP5), **Deliverables [D5.1.2, D5.2.1 and D5.3.1]**

The next figure depicts the functional architectural view of VITAL Framework:

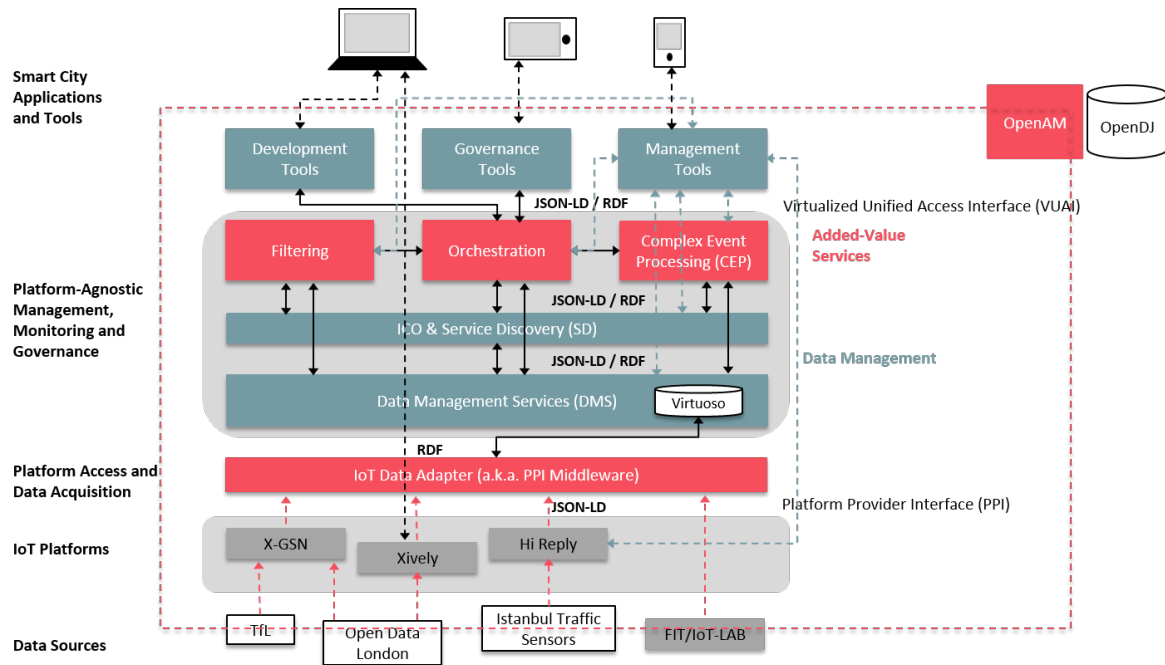


Figure 1 VITAL Functional Architecture

The communication between all functional modules is based on RESTful web services; this mechanism, which is resource-oriented, allows the integration of modules developed with different technological solutions, high reliability and easy scalability.

JSON-LD has been chosen as the data communication language to be used in the VITAL Framework. It is a light data format that represents Linked Data in JSON, which facilitates the representation of each element of VITAL (sensors, services, etc), by describing them according to the VITAL ontology. It is used also for communicating sensor and IoT systems data and metadata.

This communication mechanism enables to develop each functional component as an independent resource and avoiding other technological constraints than the ones expressed in the functional requirements.

2.2 IoT Data Adapter

The IoT Data Adapter is the component that acts as the intermediary between the VITAL platform and the PPI-compliant IoT systems that are connected to VITAL, as depicted in Figure 1 VITAL Functional Architecture. The IoT Data Adapter aims to (1) provide a web-accessible user interface through which external (or internal) IoT systems can be managed (e.g. registered and deregistered, enabled and disabled), and (2) to collect information from the registered IoT systems, and (3) push them to the Data Management Services (DMS).

More information about this component and the services it provides can be found in [D3.2.2].

2.3 Data Management Services (DMS)

The Data Management Service (DMS) is responsible for storing and allowing access to all kinds of information (i.e. metadata about IoT systems, IoT services and ICOs), as well as to historical measurement data. The data is modelled using the VITAL ontology. The development of DMS is completed using the JAVA EE technologies and provides eight RESTful interfaces for storing and retrieving all types of metadata, as well as measurement data acquired from ICOs. Four interfaces for IoT Data Adapter module are provided in order to store metadata and measurement data into DMS. Similarly, four other interfaces are for the retrieval of metadata and measurement data. The interfaces that provide access to the metadata and observations use query language based on MongoDB API, which allows easy and convenient querying. More specifically, using RESTful interfaces, the DMS exposes the following functionalities:

Storage: Allows the IoT Data Adapter module to store metadata and ICO observations received from PPIs. IoT Data Adapter transfers this information in JSON-LD format. This functionality is exposed through four RESTful interfaces to push data into DMS.

Retrieval: DMS provides access to the stored metadata and ICO observations. Retrieval of metadata is allowed through four RESTful interfaces. The ICO observations (historical measurements) can be accessed using MongoDB query language.

2.4 Discovery service

The Service Discovery (SD) module provides the means for discovering ICOs, IoT services and IoT systems that are virtualized in the VITAL platform. The Service Discovery is accessible through a RESTful web service, which exposes information like the context, a description, its status and the operations it offers, as described in deliverable [D3.2.2].

All operations provided by the module are triggered by POST requests, receiving as input a JSON object. Keys defined in such object are used as parameters to specify discovery options. The produced response is a list of objects that satisfies discovery criteria, returned in JSON-LD format.

An example of service provided is the discovery of ICOs residing in a certain spatial region. Based on latitude and longitude coordinates, and a radius of interest, the produced result contains identifiers for ICOs registered in the specified area.

The Discovery Service module has interactions with the modules above and below in the architecture. More precisely it provides functions that can be requested by all the modules above which are, namely:

- Complex Event Processing (CEP)
- Orchestrator
- Management Services

The module below is Data Management Service (DMS) whose functions can be exploited by the Discovery in order to accomplish its operations.

The main functions defined to discover ICOs are:

- **getICOs**: using this function the Discovery queries the DMS in order to find all the ICOs available in a defined area
- **getICO**: this functions is used to query the DMS in order to retrieve one single device characterized by is URI

Further details on the Discovery module are available in the deliverable [D4.1.1].

2.5 Filtering Service

Filtering is a component that enables filtering of data and events to be available for all modules inside VITAL system. It is accessible through a RESTful web service, developed using Java EE technology, which exposes information like the context, a description, its status and the exposed operations, as described in deliverable [D3.2.2]. All operations provided by the Filtering are triggered by POST requests, receiving as input a JSON object. Filtered data are returned in JSON-LD format.

An example of service provided by Filtering module is the selection of measured values that result to be above or below a specified threshold.

2.6 Orchestration Service

The VITAL orchestrator is one of the core modules of the VITAL architecture (i.e. the Workflow Manager module) as listed in deliverable [D2.3]. It is designed to offer cross-platform & cross-business-context process integration in smart cities by providing an abstract digital layer over the application silos of modern smart cities, including (visual) tools and techniques for the composition of VITAL services by selecting and combining data sources and data streams, and by actuating services in certain conditions.

The VITAL Orchestrator has a hybrid role in the VITAL architecture (

Figure 1). On the one hand, it can be viewed as a high level component of the VITAL architecture. The orchestrator does not generate data on its own; rather, it is a consumer and coordinator of data and services provided by lower level systems. On the other hand, through the combination of lower lever functionalities, it produces complex services and/or data streams, which can be then integrated in the VITAL toolset.

Design choices, internal architecture and implementation of the VITAL Orchestrator module are described in [D4.4.1].

2.7 Complex Event Processing (VITALCEP)

As described in deliverable [D4.3.1], the main functionality of the VITAL Complex Event Processing engine is to analysing streams of data in real-time according to applications specific logics. It is a tool versatile as far as its behaviour can be adapted to several and different data processing needs by means of the DOLCE language.

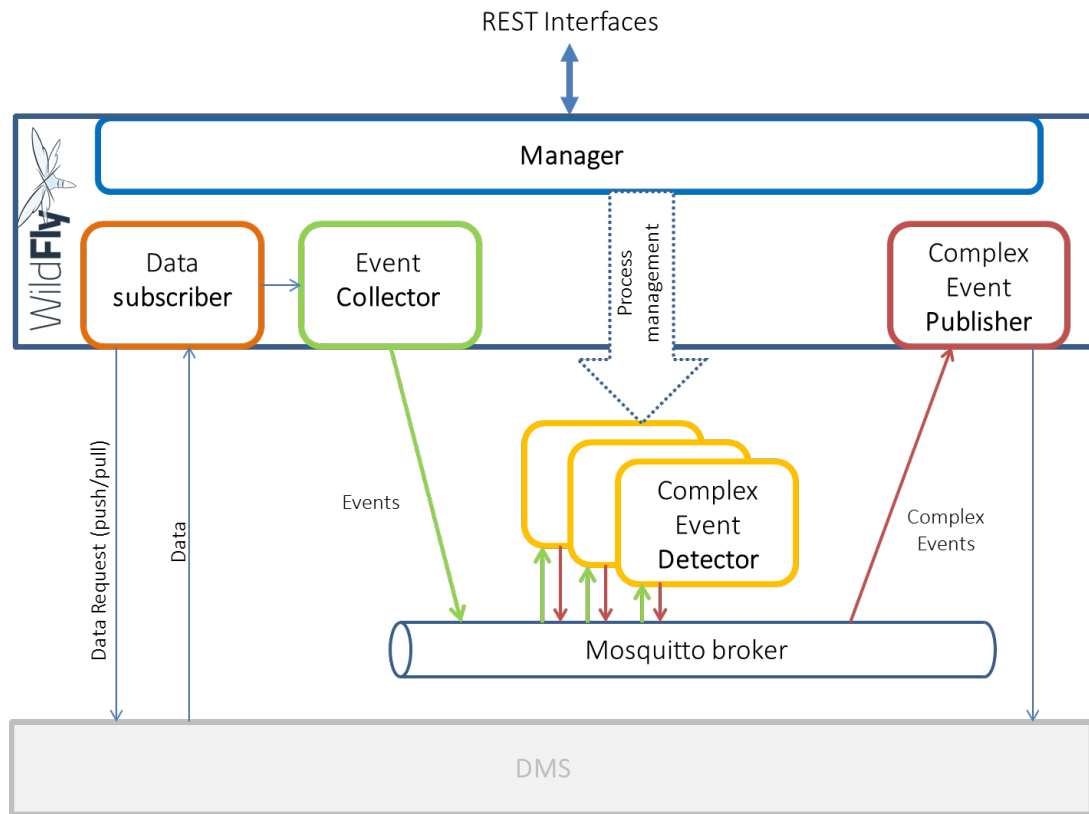


Figure 2 VITAL CEP Functional Architecture

The VITALCEP consists of five logical modules:

- **Complex Event Detector (CEP engine):** it is the core of the VITALCEP, this module processes incoming data to detect complex events according to the DOLCE rules specifications.
- **Event Collector:** this module receives or gathers data from the DMS and translates it into simple events. These simple events are then sent to CEP engine in order to be processed.
- **Complex Event Publisher:** this module receives the complex event detected by the CEP engine, translates them into the VITAL data format and sends these complex events to the DMS.
- **Data subscriber:** this module manages the connection to DMS in order to receive (subscription) or to retrieve (querying) data to feed the Event Collector module.
- **Manager:** this module manages the management, filtering and PPI interfaces of VITALCEP. It is also responsible of managing the creation and deletion of VITALCEP instances to fulfil the needs of the processing.

The VITALCEP module participates in different services of VITAL Framework. According to the main aim of a CEP engine it will offer the Event Driven capability of the Framework, but the capabilities of this module make it a valuable tool to provide some functions to the Filtering service (error values detection, deviation of a value from an specific point, value trends, etc) and also to the monitoring functionality (by

detecting specific status of the sensors). VITALCEP offers three main REST APIs which can be found in deliverable [D4.3.1]

2.8 Development Tools

One of the key goals of the VITAL project is to facilitate the development, deployment and operation of IoT applications and services. The VITAL development and deployment environment contributes to the achievement of this goal by allowing developers to access and compose the various capabilities that the VITAL framework offers (e.g. filtering, complex event processing, resource discovery), in order to implement smart city applications. All these capabilities are exposed through Virtualized Unified Access Interfaces (VUAI), and are made available to the developers through the main component of the environment, which is the development tool.

More information about the design and implementation of the development tools can be found in [D5.2.1].

2.9 Management Platform

The VITAL Management Platform is a higher-level component that integrates with multiple and heterogeneous IoT systems and services. It provides a unified view of the underlying topology by collecting, storing and displaying monitoring and configuration data associated with each component of the VITAL architecture, whether it is a system, a sensor or a service.

More specifically, the VITAL management layer provides the following functionalities (described also in [D5.1.2-2015]):

Topology: The management platform, integrates with the **Service Discovery** module, to dynamically find and collect metadata information of the entities in the managed ecosystem and produce a map of connected resources in Vital. Alternatively, the management platform can be configured to utilize a static list of URLs, and collect metadata from the exposed VUAI interface of each system / service behind the URL.

Monitoring: The management platform collects performance-associated data from each **VUAI** enabled component in VITAL. The metrics are stored and are displayed in gauge or time-series charts in the management dashboard. Additionally, another module of the platform collects SLA related data to monitor.

Configuration: The management platform connects to **VUAI**s to retrieve the set of supported configuration parameters they support and enables administrators to edit them as desired.

Security: The management platform integrates with the **Security** to provide administrators with the ability to manage users and groups of the infrastructure (see also Section 2.10 below).

2.10 Security

As specified in deliverable [D3.2.3], the security layer controls access to resources handling both authentication and authorization using the following open source components:

- OpenAM Identity Provider, that performs authentication
- Policy Agent Service Provider, that performs authorization to control access to HTTP system services
- OpenDJ LDAPv3 compliant directory service
- Reverse Proxy that masks the underlying services (e.g. PPIs)

An additional custom module developed for VITAL (described in detail in section 5.7 of D5.1.2) acts as a central component allowing in one place easy access to all the security functions of the system adapting the OpenAM based solution to the needs of VITAL. The module can also, in coordination with the other modules of the system, perform a fine grained control over access to data when the Policy Agent cannot (because data is not identified by the URI of the HTTP request); this situation is typically that of a query to retrieve data from the DMS or the PPIs (for the latter an experimental gateway has been developed to filter results).

A setup has been configured for testing. The identity provider and the service provider are exposed to these public endpoints:

- Identity Provider: <https://vitalgateway.cloud.reply.eu/idp>
- Service Provider: <https://vitalsp.cloud.reply.eu>

The custom security module is available at:

- <https://vitalgateway.cloud.reply.eu/securitywrapper>

The exposed services are on SSL, temporarily with a self-signed certificate.

VITAL services, including the PPIs, are exposed at this base URL: <https://vitalsp.cloud.reply.eu/vital>.

The authentication and authorization functionalities are illustrated in the Figure 3.

Part of the security system is also a management web application which allows to configure users, groups and policies (which define rules to allow access to resources). It is being integrated at the UI level with the main VITAL Management module.

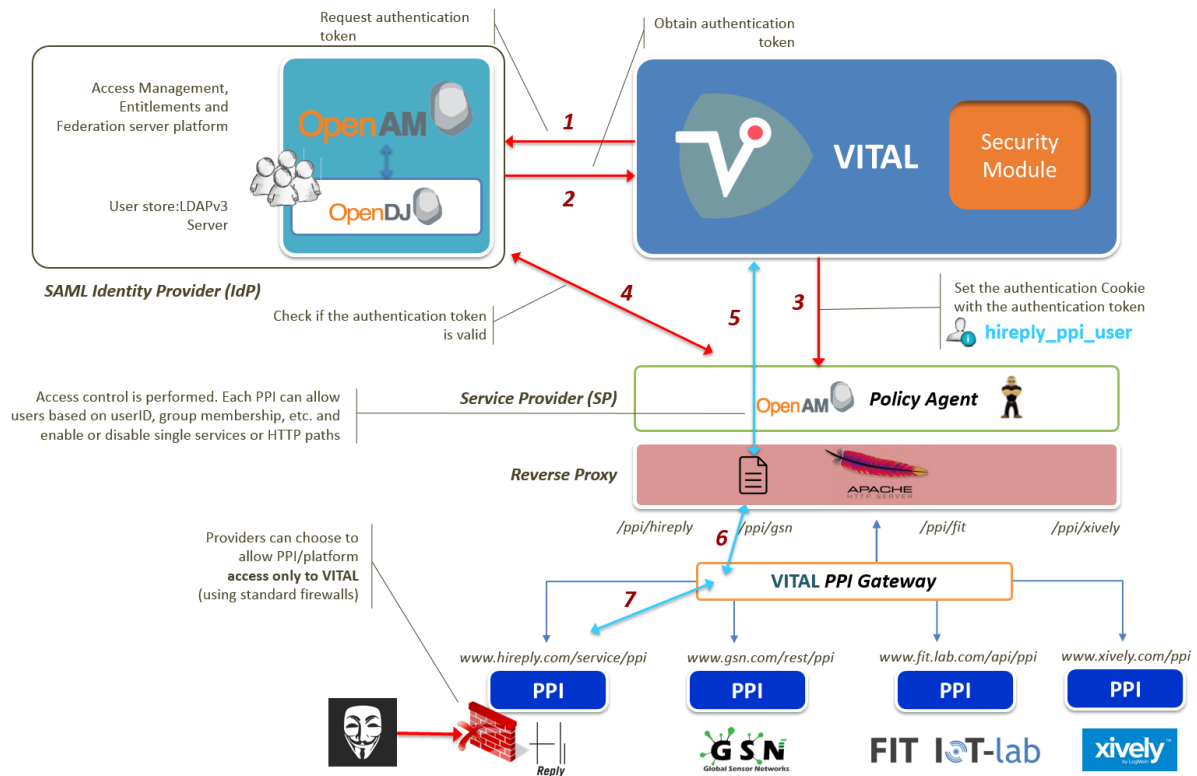


Figure 3 VITAL Security layer

3 DEPLOYMENT

On the way of providing an open source platform the project has chosen a set of technologies and tools to serve as base for VITAL developments, with the objective of provide homogenization between modules and have a common deployment procedure for all modules. Other characteristics taken into account are scalability and reliability.

The first step was to select the technologic base for all modules, to this aim **Java** platform has been adopted, among others, as it provides an open source language, able to run in any server and supported by a great scientific community. Together with Java we use **Maven** as software managing and building tool.

The second step was to provide environment to host applications and data. For this purpose **Wildfly** application server and **MongoDB** were chosen, the first as it is a well-known open source application server with high reliability and performance. For the second it was chosen at a first step VirtuosoDB, but as it is explained in D3.2.3 finally we opted for MongoDB.

Other deployment tools involved in VITAL are only required by some modules such as:

Node.js: it is a runtime server for server side applications.

Bower: it is a package manager to optimize dependencies of applications.

Grunts: JavaScript task runner.

Mosquitto: MQTT message broker.

Java Spark: it is a lightweight web framework for Java applications.

OpenDJ: it is a light directory access protocol LDAP.

OpenAM: it is a platform for access management and federation.

The table below these lines provides information regarding the needs of each functionality:

	Discovery	Filtering	DMS	IoT data adapter	Management	Orchestrator	Security	CEP
Java 8.0								
Maven								
WildFly								
MongoDB								
Node.js								
Bower								
Grunt								
Mosquitto								
Java Spark								
OpenAM								
OpenDJ								

In order to prepare the environment, the best choice is to follow instructions from providers:

- Java installation:
 - <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- Maven installation:
 - <https://maven.apache.org/>
- Wildfly installation:
 - <http://wildfly.org>
- MongoDB
 - <https://www.mongodb.org>
- Node.js
 - <https://nodejs.org>
- Grunts:
 - <http://gruntjs.com>
- Bower:

- <http://bower.io>
- Mosquitto:
 - <http://mosquitto.org/>
- JavaSpark:
 - <http://sparkjava.com>
- OpenAM:
 - <http://docs.forgerock.org/en/openam/latest/install-guide>
- OpenDJ:
 - <https://backstage.forgerock.com/#!/docs/opendj>

Next sections provide instructions to build and deploy each module of VITAL. More detailed information can be found in the README files of modules in the GITHUB project of VITAL (At the time of writing this document the GitHub project is under construction, the URL to the projects repositories are pointing to GitLab repository provided in order to progress in deployment task until new GitHub project is available).

3.1 IoT Data Adapter

The IoT Data Adapter comprises two Maven-based projects written in Java: one that implements the component functionalities (IoT-data-adapter), and one that serves as the PPI implementation that the component exposes (IoT-data-adapter-ppi).

- **IoT Data Adapter**

Build IoT Data Adapter package:

- Clone git repository:

```
>git clone http://gitlab.atosresearch.eu/vital-iot/vital-core-iot-data-adapter-ppi.git
```
- Build package

```
>cd {project_source_dir}
>mvn clean package
>{project_source_dir}/bin/jboss-cli.sh -connect
>/system-property=iot-data-adapter.properties:add(value=${jboss.server.config.dir}/iot-data-adapter.properties)
>curl -X PUT http://\[es-host\]:\[es-port\]/iot-data-adapter
>cp {project_source}/main/resource/iot-data-adapter.properties $WILDFLY_HOME/standalone/configuration
```

- Deployment

```
>mvn wildfly:deploy
```

- **PPI for IoT Data Adapter**

- Clone git repository:

```
>git clone http://gitlab.atosresearch.eu/vital-iot/vital-  
core-iot-data-adapter-ppi.git
```

- Build package

```
>cd {project_source_dir}  
>mvn clean package  
>{project_source_dir}/bin/jboss-cli.sh -connect  
>/system-property=iot-data-adapter-  
ppi.properties:add(value=${jboss.server.config.dir}/iot-data-  
adapter-ppi.properties)>curl -X  
>PUT http://\[es-host\]:\[es-port\]/iot-data-adapter  
>cp {project_source_dir}/main/resource/iot-data-adapter-  
ppi.properties $WILDFLY_HOME/standalone/configuration
```

- Deployment

```
>mvn wildfly:deploy
```

3.2 Security Gateway for PPIs

- Clone git repository:

```
>git clone http://gitlab.atosresearch.eu/vital-iot/vital-  
core-security-gateway.git
```

- Build package

```
>cd {project_source_dir}  
>vi src/main/resources/config.properties  
(set the values defined there to match you deployment scenario)  
>mvn clean package
```

- Configure Wildfly

```
>vi $WILDFLY_HOME/standalone/configuration/standalone.xml
```

Be sure to have into section “management→security-realms”:

```
<security-realm name="UndertowRealm">
  <server-identities>
    <ssl>
      <keystore path="my.jks" relative-to="jboss.server.config.dir" keystore-
password="password" alias="mycert" key-password="password"/>
    </ssl>
  </server-identities>
</security-realm>
```

Be sure to have into section “profile→subsystem (the undertow one) →server”:

```
<http-listener name="default" redirect-socket="https" socket-binding="http"/>
<https-listener name="https" security-realm="UndertowRealm" socket-binding="https"/>
```

- Deployment

```
>mvn wildfly:deploy
```

3.3 Data Management Services (DMS)

- Clone git repository:

```
>git clone http://gitlab.atosresearch.eu/vital-iot/vital-
core-dms.git
```

- Build package and deploy

```
>cd {project_source_dir}
>mvn clean package exec:java
```

3.4 Discovery service

- Clone git repository:

```
>git clone http://gitlab.atosresearch.eu/vital-iot/vital-
core-discovery.git
```

- Build package

```
>cd {project_source_dir}
>mvn clean package
```

- Deployment

```
>mvn wildfly:deploy
```

3.5 Filtering Service

- Clone git repository:

```
>git clone git@gitlab.atosresearch.eu:vital-iot/vital-core-filtering.git
```

- Build package

```
>cd {project_source_dir}
>mvn clean package
```

- Deployment

```
>mvn wildfly:deploy
```

3.6 Orchestration Service

- Clone git repository:

```
>git clone git@gitlab.atosresearch.eu:vital-iot/vital-core-orchestrator.git
```

- Build package

```
>cd {project_source_dir}
>mvn clean package
```

- Configure MongoDB

```
>vi $MONGODB_HOME/etc/configuration.json
```

Be sure to have following properties into the file:

```
{
  "dms_url":
  {
    "host": "vmvital03.der1.ie",
    "port": "8011"
  },
  "discovery_url": "http://vital-integration.atosresearch.eu:8180/discoverer/ppi", "system_urls":
  [
    "http://vital-integration.atosresearch.eu:8180/vital-orchestrator-web/rest/ppi",
    "http://vital-integration.atosresearch.eu:8180/discoverer/ppi",
    "http://vital-integration.atosresearch.eu:8180/filtering/ppi",
    "http://vital-integration.atosresearch.eu:8180/iot-data-adapter-ppi/ppi/system",
    "http://vital-integration.atosresearch.eu:8180/camden-footfall-ppi/ppi/system",
    "http://vital-integration.atosresearch.eu:8180/cep",
    "http://vital-integration.atosresearch.eu:8180/hireplyppi"
  ]
}
```

```
>mongo vital-orchestrator --eval "db.CONFIGURATION.remove({})"
>mongoimport --db vital-orchestrator --collection CONFIGURATION --
file ./configuration.json
```

- Deployment

```
>mvn wildfly:deploy
```

3.7 Complex Event Processing (VITALCEP)

The VITALCEP module comprises two different projects; one project is a C++ project that implements the Complex Event Processing engine, the second is a maven-based project written in java that implements the cep manager and all interfaces of the module.

- **Complex Event Processing Engine**

- Clone git repository:

```
>git clone http://gitlab.atosresearch.eu/ari/vital-core-cep.git
```

- Build package

```
>cd {project_source_dir}/ucep
>rake
```

- **CEP manager plus interfaces**

- Clone git repository:

```
>git clone git@gitlab.atosresearch.eu:vital-iot/vital-core-cep.git
```

- Set configuration

```
>cd {project_source_dir}/src/main/resources
>cp cep.properties $WILDFLY_HOME/standalone/configuration
```

- Build package

```
> cd {project_source_dir}/src/main
>mvn clean package
```

- Deployment

```
>mvn wildfly:deploy
```

3.8 Development Tools

The development tool is a Node.js web application that was built using the Express framework. The dependencies of the development tools are:

- Express >= 4.12.4
- Grunt >= 0.1.13
- Node.js >= 0.10.34
- rstats >= 0.3.1
- sqlite3 >= 3.0.8
- In order to build the code, move into the root directory of the project, and execute:
 - >npm install**
- to make sure all required dependencies are available, and then execute:
 - >grunt build**
- Run the development and deployment environment using the command:
 - >node red.js**
- Once the environment is running, go to:
[http://\[host\]:\[port\]](http://[host]:[port])

To access it, where **host** is the host where the environment has been deployed, and **port** is the port where the web server listens (the default port is **1880**, and can be changed in **settings.js**).

3.9 Management Platform

- Clone git repository:

```
>git clone http://gitlab.atosresearch.eu/vital-iot/vital-management.git
```
- Build package

```
>cd {project_source_dir}
>mvn clean package
```
- Configure MongoDB

```
>vi $MONGODB_HOME/etc/configuration.json
```

Be sure to have following properties into the file:

```
{
  "dms_url":
  {
    "host": "vmvital03.der1.ie",
    "port": "8011"
  },
  "discovery_url": "http://vital-integration.atosresearch.eu:8180/discoverer/ppi", "system_urls":
  [
    "http://vital-integration.atosresearch.eu:8180/vital-orchestrator-web/rest/ppi",
    "http://vital-integration.atosresearch.eu:8180/discoverer/ppi",
    "http://vital-integration.atosresearch.eu:8180/filtering/ppi",
    "http://vital-integration.atosresearch.eu:8180/iot-data-adapter-ppi/ppi/system",
    "http://vital-integration.atosresearch.eu:8180/camden-footfall-ppi/ppi/system",
    "http://vital-integration.atosresearch.eu:8180/cep",
    "http://vital-integration.atosresearch.eu:8180/hireplyppi"
  ]
}
```

```
>mongo vital-orchestrator --eval
"db.CONFIGURATION.remove({})"
>mongoimport --db vital-orchestrator --collection
CONFIGURATION --file ./configuration.json
```

- Deployment

```
>mvn wildfly:deploy
```

3.10 Security

The setup steps are:

- Download¹ the OpenDJ 2.6.2 user store and install it
- Download² and install the OpenAM 12 war file
- Install and configure OpenAM according to instructions in: <http://docs.forgerock.org/en/openam/latest/install-guide>.

To enforce policies for OpenAM, a web policy agent installed in a web server intercepts requests from users trying to access a protected web resource and denies access until the user has authorization from OpenAM to access the resource. As a web server, Apache should be used; installation must be done according to instructions at:

- <http://docs.forgerock.org/en/openam-pa/3.3.0/web-install-guide/#chap-apache-24>

The Apache configuration where the Service Provider is running must be updated with the needed proxy rules that map the correct service endpoints, e.g.:

```
ProxyPass /ppi/hireply http://hireplyhost/endpoint/ppi
```

¹ <https://forgerock.org/opendj/>

² <http://forgerock.org/downloads/openam-builds/>

The custom module described in section 2.10 (as well as the management application) can be built from source as described in the bundled README.md file and deployed in a WildFly container. The basic steps are:

- Configure the modules to connect correctly to the other services: modify the values in "src/main/resources/config.properties" and those in "src/main/javascript/js/common/resources/resources.js" for the management module.

Repeat the steps to clone git repository, build the package and deploy for the three components of security. The Github repositories are:

- <http://gitlab.atosresearch.eu/vital-iot/vital-core-security-snmp-exposer.git>
- <http://gitlab.atosresearch.eu/vital-iot/vital-core-security-adapter.git>
- <http://gitlab.atosresearch.eu/vital-iot/vital-management-security-ui.git>

- **Clone git repository**

```
>git clone {git repository}
```

- **Build package**

```
>cd {project_source_dir}
```

```
>mvn clean package
```

- **Deployment**

```
>mvn wildfly:deploy
```

3.11 PPI for Camden Footfall

- **Clone git repository:**

```
>git clone http://gitlab.atosresearch.eu/vital-iot/vital-ppi-camden-footfall.git
```

- **Build package**

```
>cd {project_source_dir}
```

```
>mvn clean package
```

```
>{project_source_dir}/bin/jboss-cli.sh -connect
```

```
/system-property=camden-footfall-
```

```
ppi.properties:add(value=${jboss.server.config.dir}/camden-footfall-ppi.properties)
```

```
>curl -X PUT http://\[es-host\]:\[es-port\]/iot-data-adapter
```

```
>cp {project_source}/main/resource/ camden-footfall-ppi.properties $WILDFLY_HOME/standalone/configuration
```

- **Deployment**

```
>mvn wildfly:deploy
```

3.12 PPI for HI Reply Istanbul Traffic

- Clone git repository:

```
>git clone http://gitlab.atosresearch.eu/vital-iot/vital-  
ppi-istanbul-traffic.git
```

- Build package

```
>cd {project_source_dir}  
>vi src/main/resources/config.properties
```

(set the values defined there to match you deployment scenario)

```
>mvn clean package
```

- Configure Wildfly

```
>vi $WILDFLY_HOME/standalone/configuration/standalone.xml
```

Be sure to have into section “management→security-realms”:

```
<security-realm name="UndertowRealm">  
  <server-identities>  
    <ssl>  
      <keystore path="my.jks" relative-to="jboss.server.config.dir" keystore-  
password="password" alias="mycert" key-password="password"/>  
    </ssl>  
  </server-identities>  
</security-realm>
```

Be sure to have into section “profile→subsystem (the undertow one)→server”:

```
<http-listener name="default" redirect-socket="https" socket-binding="http"/>  
<https-listener name="https" security-realm="UndertowRealm" socket-binding="https"/>
```

- Deployment

```
>mvn wildfly:deploy
```

4 FUNTIONAL MODULES INTERACTIONS

This section describes the main interactions between VITAL modules showing the current status of integration between components at the time of writing this document.

4.1 IoT Data Adapter

IoT Data Adapter is responsible for collecting information about and from the registered IoT systems. As soon as a new IoT system is registered to the VITAL platform, IoT Data Adapter retrieves metadata about the system itself, the IoT

services it provides, and the sensors it manages. The retrieved metadata are pushed to DMS through a **pushData** VUAI the latter exposes for that purpose.

Once an IoT system has been registered to the VITAL platform, its provider can explicitly request VITAL (through the IoT Data Adapter user interface) to refresh the metadata stored about it. IoT Data Adapter is then responsible to retrieve a fresh version of the IoT system, IoT service and sensor metadata, and push them to DMS, using the same VUAI described above. This is how IoT Data Adapter manages information that is exposed by the registered IoT systems and that is considered to be rarely modified (e.g. the name of the system). Figure 4 depicts this sequence of actions.

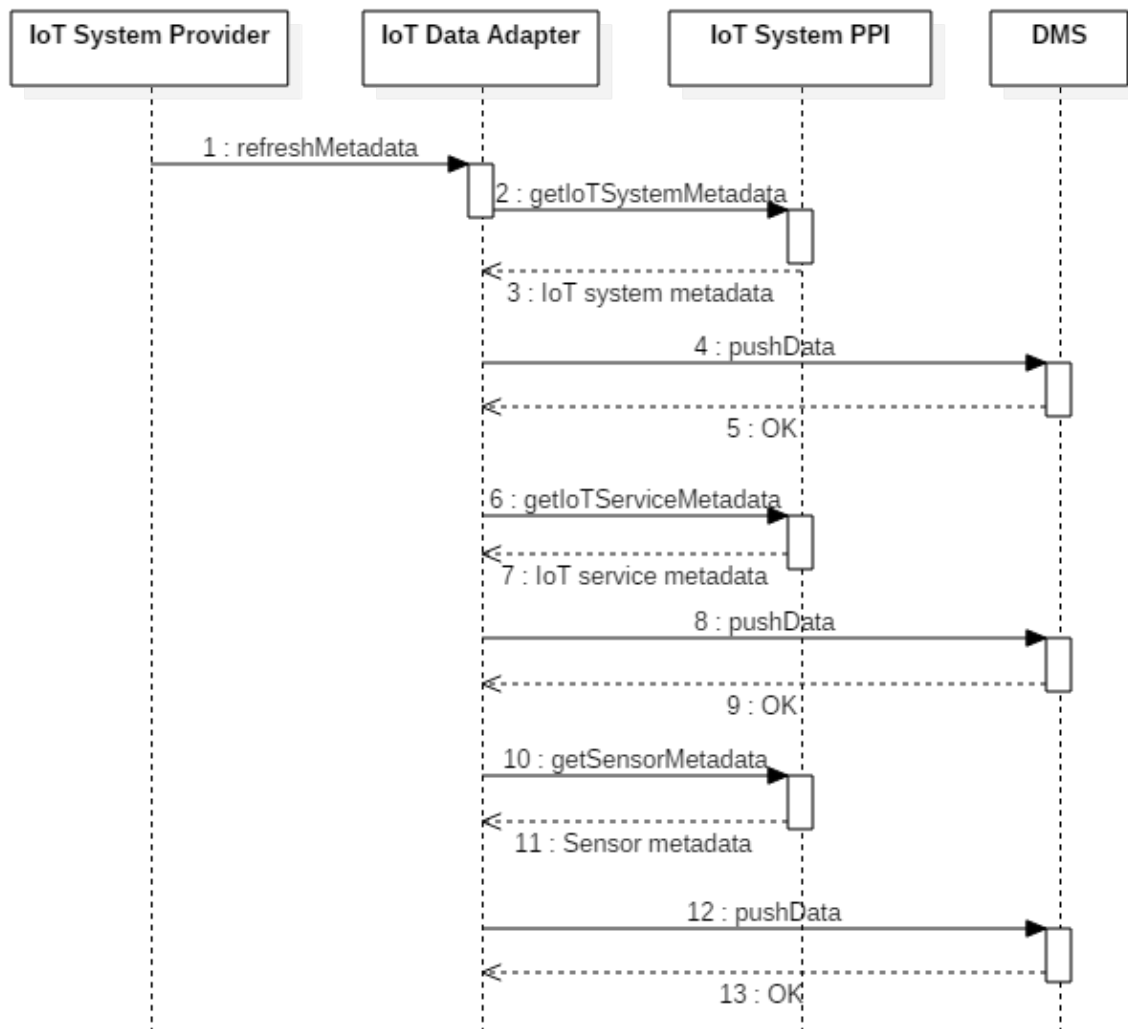


Figure 4 IoT system provider requests from IoT Data Adapter to refresh all related metadata.

Apart from static information, IoT systems provide also access to dynamic information (i.e. information that changes or is generated during the lifetime of the system). At the moment, this dynamic information includes:

- the current status of the IoT system
- the current status of all sensors managed by the IoT system

- the current locations of all mobile sensors managed by the IoT system
- new observations made by sensors managed by the IoT system

In order to guarantee that VITAL has an almost up-to-date version of this dynamic information, IoT Data Adapter pulls periodically this information, and pushes it to DMS. The frequency with which each one of the above pieces of information is pulled from an IoT system is configurable (through the IoT Data Adapter web interface) and might vary from system to system, depending also on business requirements of the targeted smart cities applications. Figure 5 illustrates how IoT Data Adapter periodically pulls from the PPI of an IoT system new observations that have been made in the meantime by a sensor.

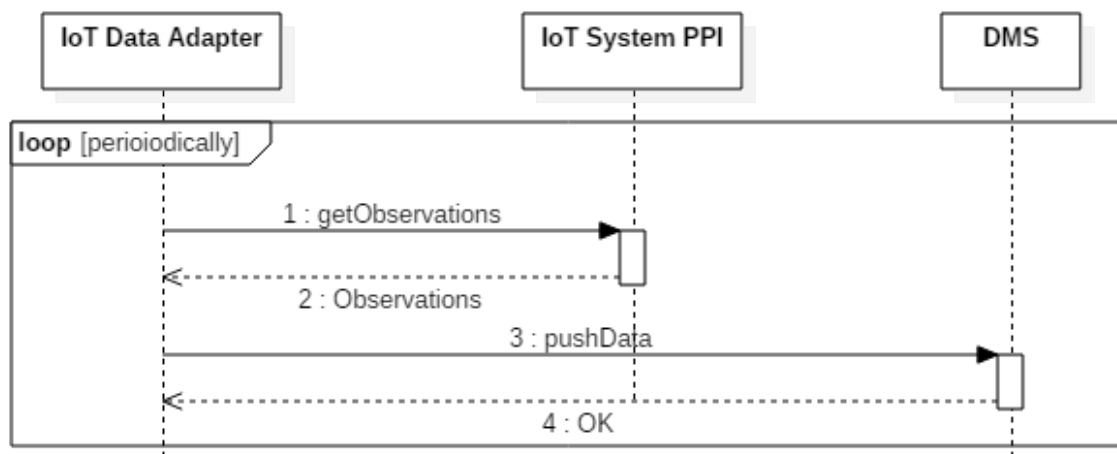


Figure 5 IoT Data Adapter pulls periodically observations made by a sensor.

Finally, IoT Data Adapter always checks for the provision of a push-based mechanism that it can probably use to collect any of the above pieces of information, and prefers it over a pull-based mechanism if both are supported by the corresponding PPI implementation. Figure 6 illustrates how IoT Data Adapter uses the push-based mechanism to get observations from the PPI of an IoT system.

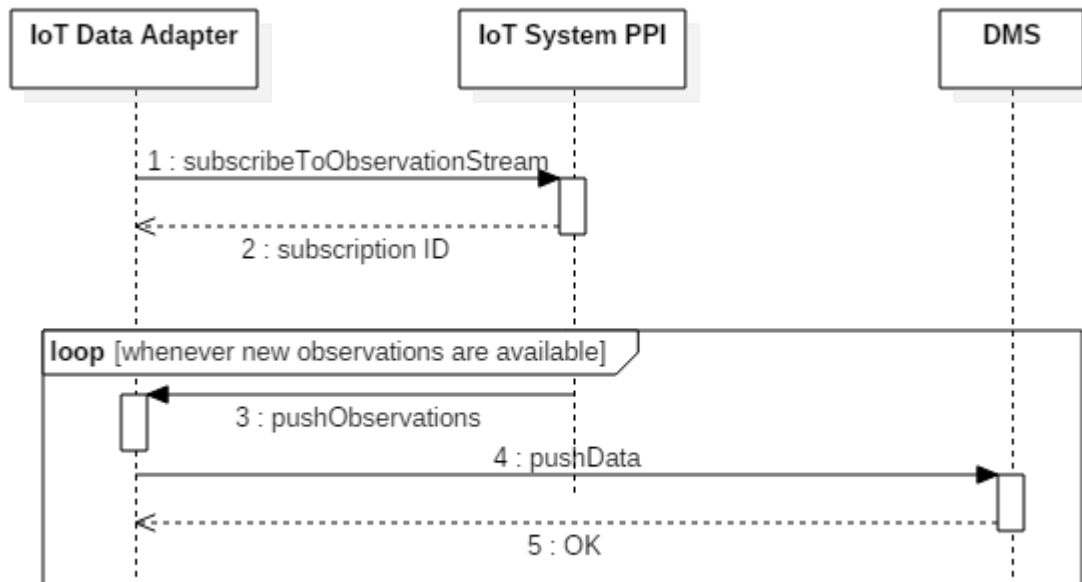


Figure 6 IoT system pushes new observations made by a sensor to IoT Data Adapter.

4.2 Data Management Services (DMS)

Sequence diagrams below show the sequence of interactions between the DMS and various VITAL modules. Data is pushed in the DMS by the IoT Data Adapter module using the insert interfaces. Whereas, the observations and metadata are accessed by different VITAL modules e.g. Service discover, filtering, CEP, etc.

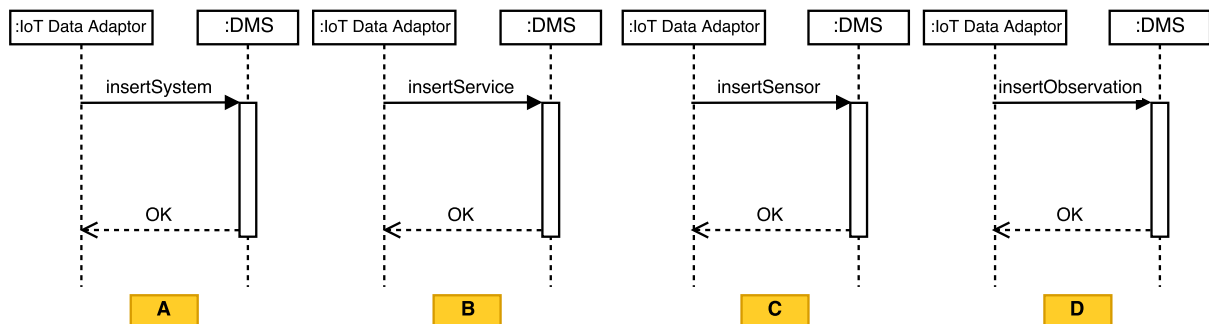


Figure 7 IoT Data Adapter pushes data into DMS.

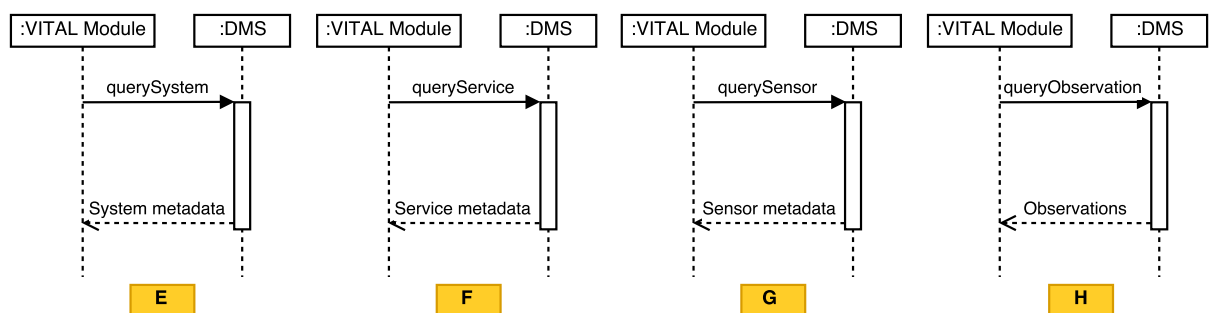


Figure 8 VITAL modules access ICO observations.

4.3 Discovery service

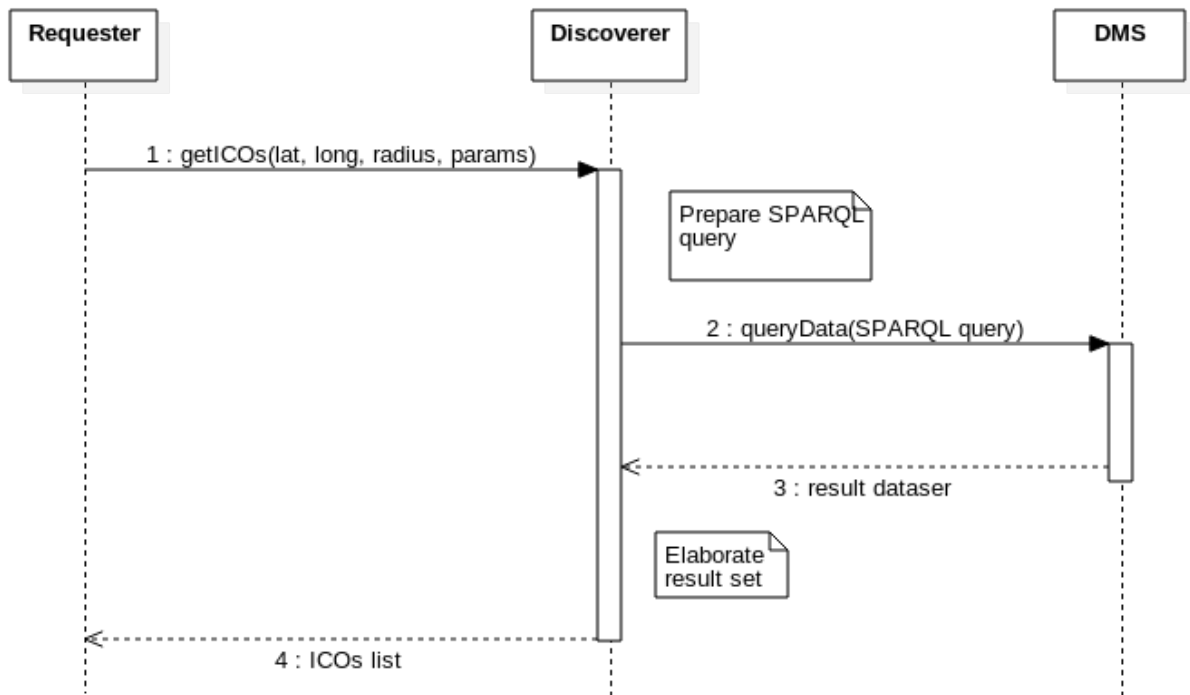


Figure 8 Discovery Service.

4.4 Filtering Service

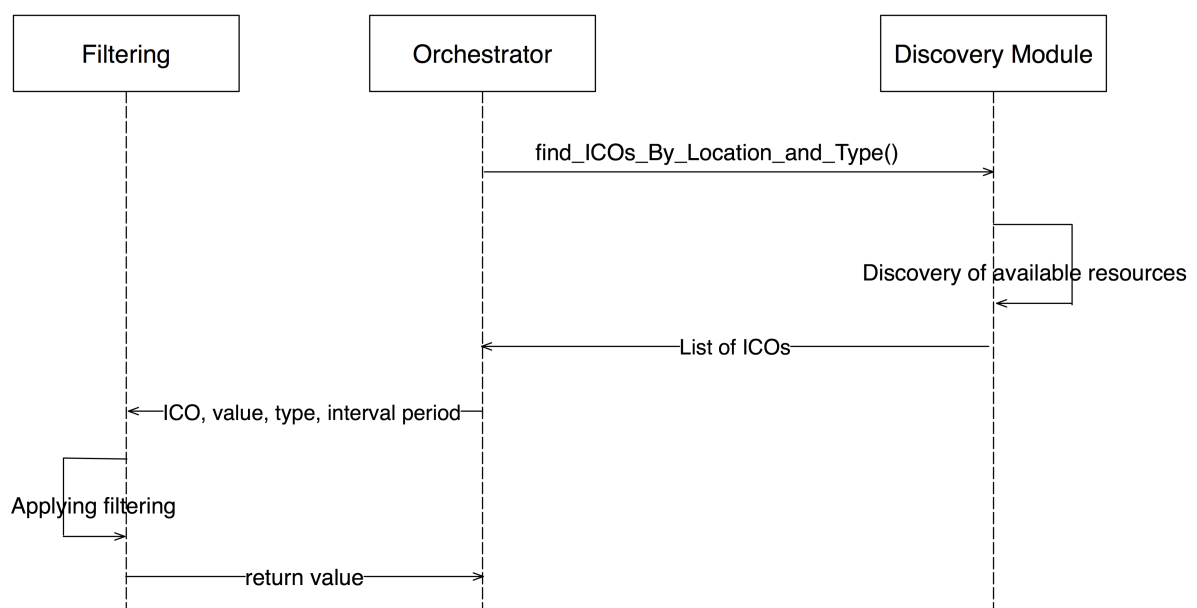


Figure 9 Filtering Service.

4.5 Orchestration Service

In the diagram below, we display a generic case of how the Orchestrator interacts with other VITAL modules. The workflow is split into steps, where in each step the orchestrator performs the following actions: (1) local processing, (2) retrieve metadata information from SD, (3) retrieve data from VUALs (PPIS or VITAL modules), (4) process the result and return it.

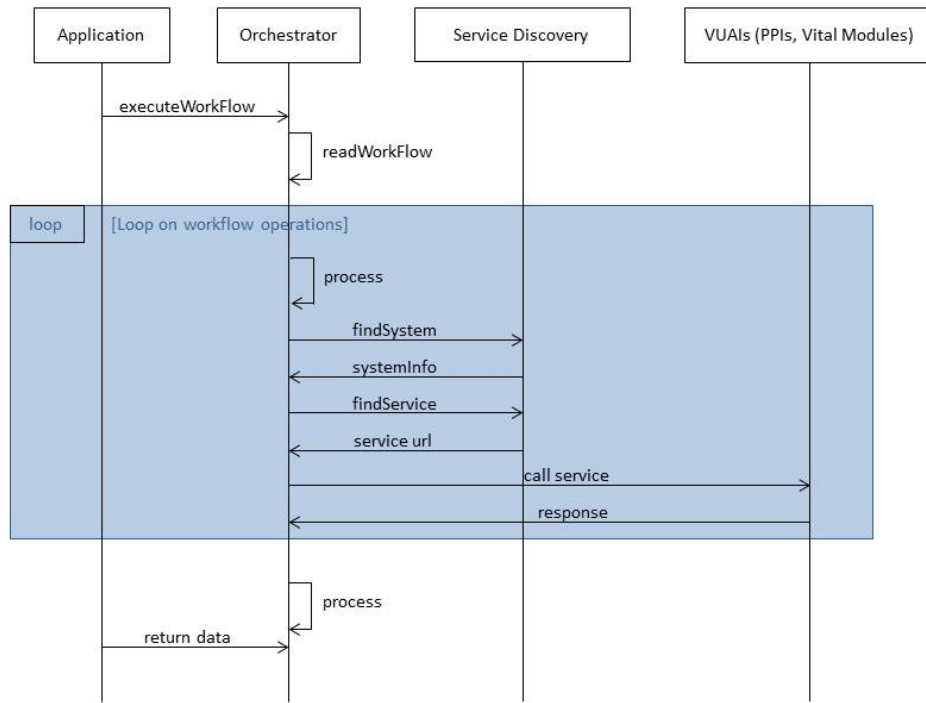


Figure 10 Orchestration Service.

4.6 Complex Event Processing (VITALCEP)

4.6.1 VITALCEP instance creation

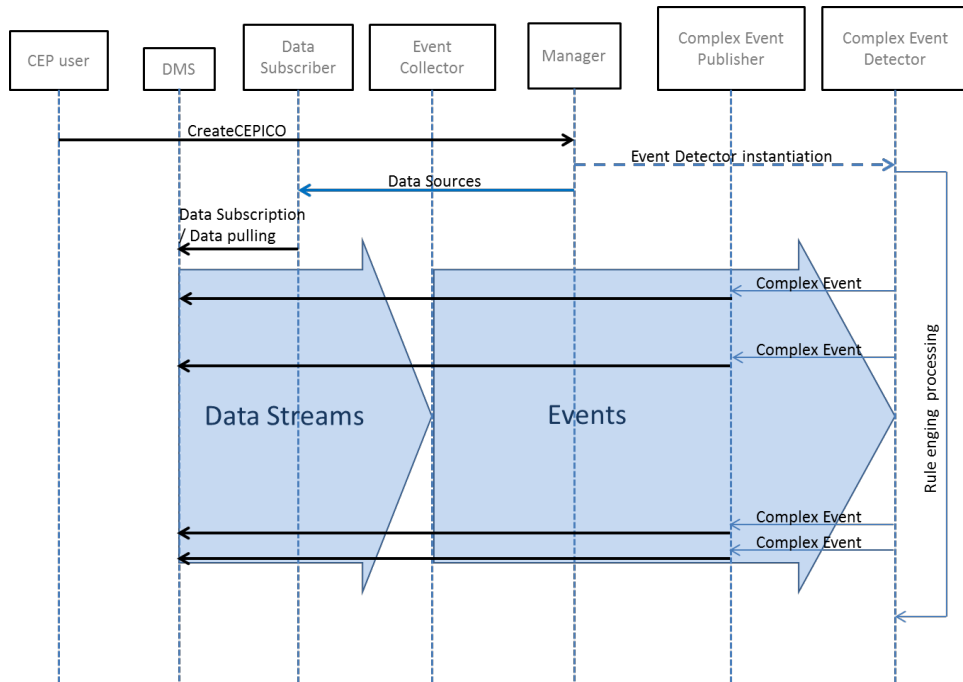


Figure 11 VITALCEP instance creation.

4.6.2 VITALCEP instance update

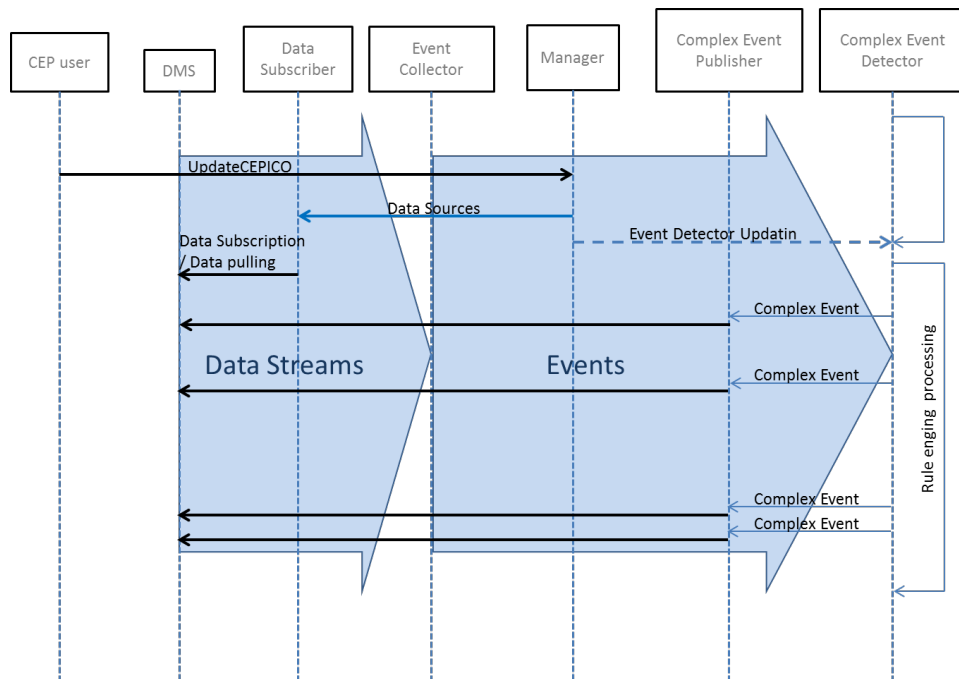


Figure 12 VITALCEP instance update.

4.6.3 VITALCEP instance deletion

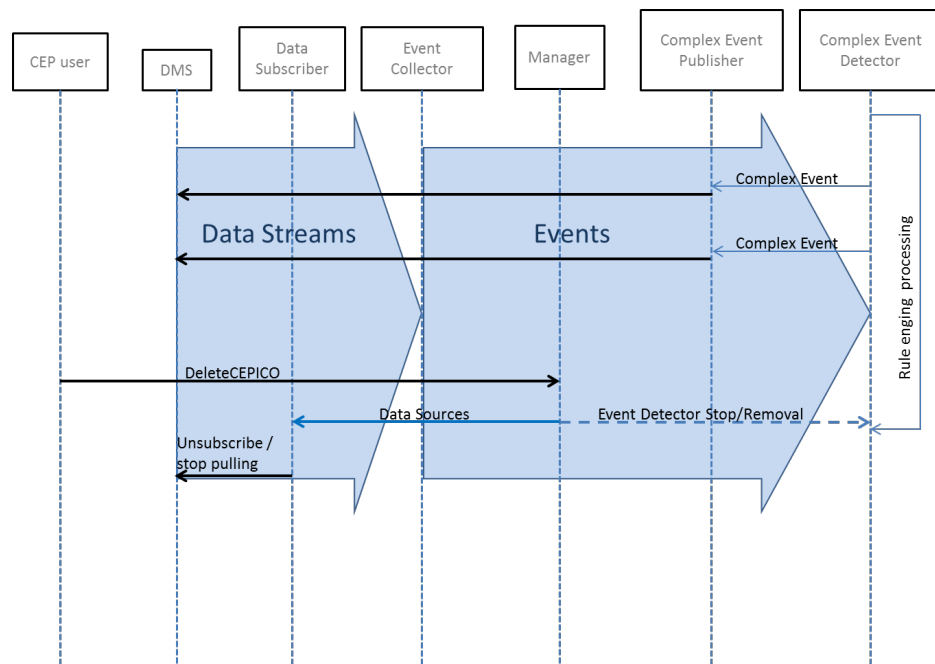


Figure 13 VITACEP instance deletion.

4.7 Development Tools

The development and deployment environment is prepared to integrate the functionalities provided by the VITAL framework. At the time of writing this document no all the functionalities are integrated yet. The list of all functionalities that will be provided by VITAL is:

- Resource (i.e. ICO and service) discovery
- Complex Event Processing (CEP)
- Filtering
- IoT system, ICO, data stream, security, configuration, SLA (Service Level Agreements) and VITAL component management
- Business Process Management (BPM) as implemented through the VITAL Orchestrator module

Part of the development and deployment environment is the VITAL development tool that serves as a single entry point to developers that want to build and deploy IoT applications that transcend multiple IoT platforms, architectures and business contexts by leveraging the capabilities provided by the VITAL framework. All these capabilities are exposed through Virtualized Unified Access Interfaces.

Based on the above, the development tools do not provide any sophisticated integration or interaction with any other VITAL component. In essence, all these tools do is expose all functionalities that the above components provide (or a simplified version of them) through an easy-to-use web interface with text-fields and drop-downs. Behind the scenes, the tools collect the information that the developer has

provided, form a request, and forward it to the appropriate VUAL (e.g. getICOs) of the appropriate component (e.g. service discovery).

4.8 Management Tools

In the diagram below, we display two examples of how the Management platform interacts with other components. The first case is a simple System List, whereas the second is a request from the UI to display a list of sensors in an area, alongside a performance metric (load).

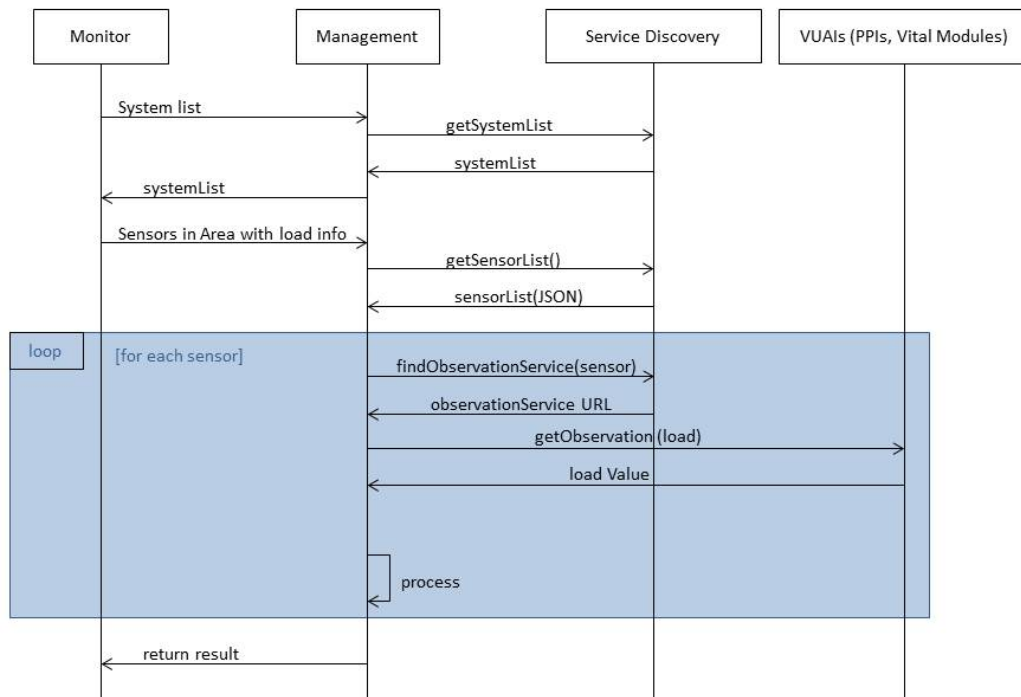


Figure 14 Management tool interactions.

4.9 Security

The Security system currently exposes the services needed by the other modules for integration. At the time of writing the modules are switching to a deployment with proxy and policy agent, which ensures protection of system services with authentication and authorization to access resources. A first implementation of fine grained data access control for DMS and PPIs has also been developed.

In the remaining of this section you can find some diagrams describing how the interaction between the security system modules and other VITAL services works.

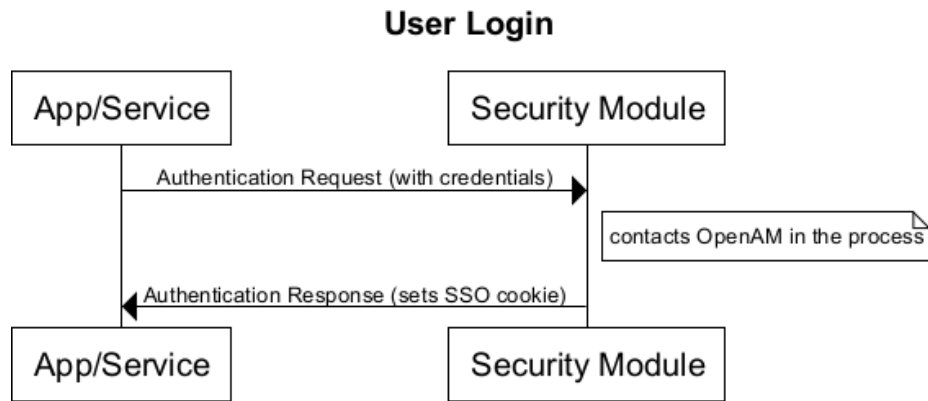


Figure 15 An application or VITAL module logs in a user

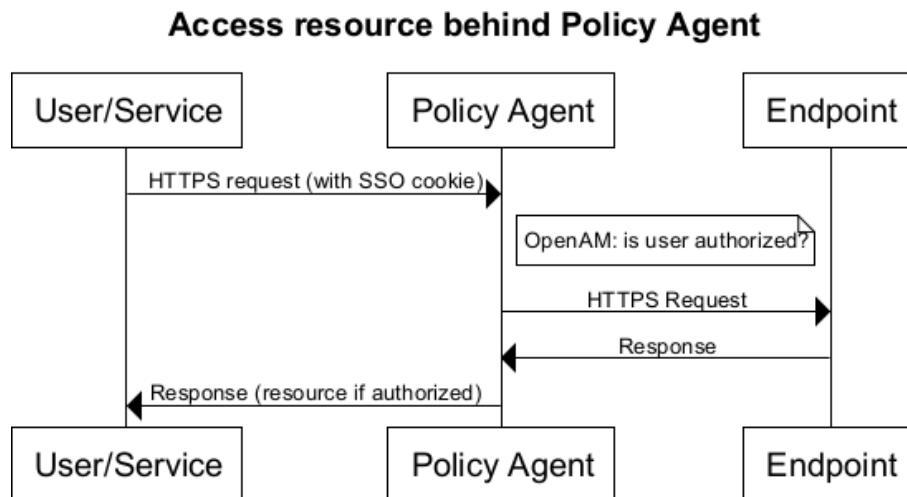


Figure 16 Endpoints protection

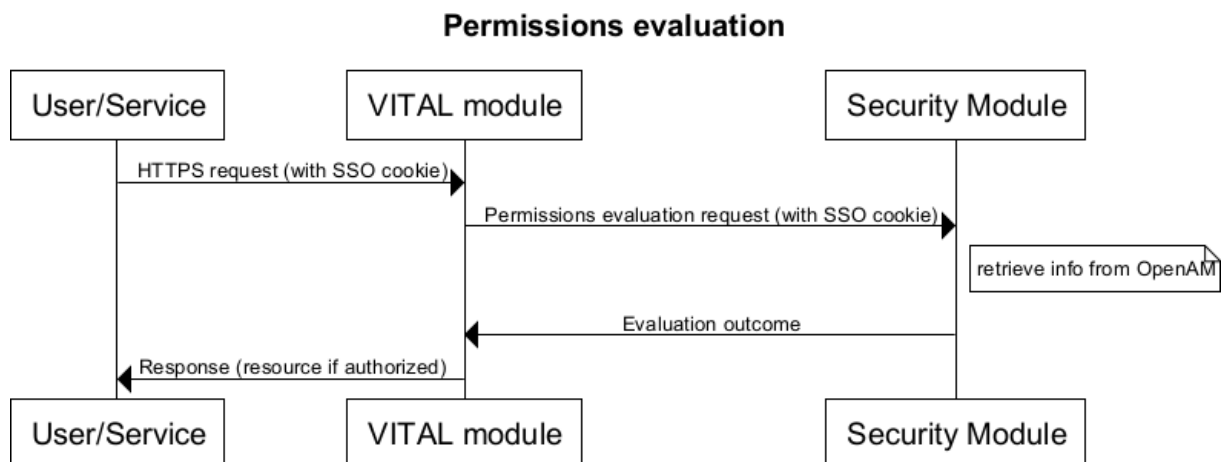


Figure 17 On demand permissions evaluation to authorize access to data

5 LICENSING

This section depicts the licenses of the libraries and components used and the final license we would like to provide.

The commitment of all partners in the project is to provide the developments as Open Source, but it is not clear yet which will be the license of each module and not all licensing tables are completely filled-in.

5.1 IoT Data Adapter

Table 1 IoT Data Adapter licensing

Component	Library	License	Licence after the project ends
IoT Data Adapter	Java	LGPL	
	Wildfly	LGPL	
	Elastic Search	Apache	
	Apache Maven	Apache	
	Bootstrap	MIT	
	jQuery	MIT	
IoT Data Adapter PPI Implementation	Java	LGPL	
	Wildfly	LGPL	
	Apache Maven	Apache	

5.2 Data Management Services (DMS)

Table 2 Data management system licensing

Component	Library	License	Licence after the project ends
DMS	spark-core 1.1.1	Apache	
	mongo-java-driver 2.13.1	Apache License v2.0.	

5.3 Discovery service

Table 3 Discovery service licensing

Component	Library	License	Licence after the project ends

Discovery	Wildfly 8.2	LGPL	LGPL
	resteasy-jaxrs 2.2.1 GA	Apache	
	resteasy-jackson-provider 2.2.1 GA	Apache	
	json-simple 1.1.1	Apache	
	log4j 1.2.17	Apache	

5.4 Filtering Service

Table 4 IoT Filtering service licensing

Component	Library	License	Licence after the project ends
Filtering	Wildfly 8.2	LGPL	LGPL
	resteasy-jaxrs 2.2.1 GA	Apache	
	resteasy-jackson-provider 2.2.1 GA	Apache	
	json-simple 1.1.1	Apache	
	log4j 1.2.17	Apache	

5.5 Orchestration Service

Table 5 Orchestration service licensing

Component	Library	License	Licence after the project ends
Orchestrator Backend	Java 1.8	LGPL	LGPL
	Wildfly 8.2	LGPL	
	Elastic Search 1.7.2	Apache	
	JSON-LD Java	BSD	
Orchestrator Frontend	AngularJS	MIT	LGPL
	Bootstrap	MIT	
	AdminLTE	MIT	
	JQuery	MIT	
	CodeMirror	MIT	

5.6 Complex Event Processing (VITALCEP)

Table 6 Complex Event Processing licensing

Components	Subcomponents	Ownership	Library or component included & Licenses for the Library	License after the project ends
VITALCEP	Complex Event Detector	ATOS	libstdc++.so.6.0.1-GNU libc-2.17.so - GNU libgcc_s.so.1 - GNU libmosquitto 1.4.0 - BSD	GPL
	Event Collector Complex Event Publisher Data subscriber	ATOS	Wildfly 8.2 - LGPL log4j 1.2.17 - Apache json 10140107- Json genson 0.99 - Apache mongo-java-driver 2.10.1- Apache jsonld-java 0.6.0 - BSD mqtt-client 1.0 - Apache	LGPL
	Manager	ATOS	javax.servlet 3.0-alpha-1 - GPL jersey-json 1.9 - GPL jaxb-impl - GPL	GPL

5.7 Development Tools

Table 7 IoT Development Tools licensing

Component	Library	License	Licence after the project ends
Development tools	Express	MIT	
	Grunt	MIT	
	Node.js	MIT	
	rstats	GPL v2	
	sqlite3	Open-source	

5.8 Management Tools

Table 8 Management Tools licensing

Component	Library	License	Licence after the project ends
Management	Java 1.8	LGPL	LGPL

Backend	Wildfly 8.2	LGPL	
	Elastic Search 1.7.2	Apache	
	JSON-LD Java	BSD	
Management Frontend	AngularJS	MIT	
	Bootstrap	MIT	
	AdminLTE	MIT	
	CodeMirror	MIT	
	JQuery	MIT	
	Leaflet	BSD	
	OpenStreetMap	ODbL	

5.9 Security

Table 9 Security licensing

Component	Library/Software	License	License after the project end (to be reviewed at the end of the project development, considering included components)
Identity provider	Forgerock OpenAM	CDDL-1.0 / CC BY-NC-ND 3.0	As external dependencies built and distributed by third parties, not linked to VITAL code, their respective licences apply
	Forgerock OpenDJ	CDDL-1.0	
	Apache Tomcat 7	Apache 2.0	
Service provider	Forgerock Web Policy Agent	CDDL-1.0 / CC BY-NC-ND 3.0	As external dependencies built and distributed by third parties, not linked to VITAL code, their respective licences apply
	Apache	Apache 2.0	
Security module	WildFly	LGPL 2.1	As external dependencies built and distributed by third parties, not linked to VITAL code, their respective licences apply
	SNMP4J	Apache 2.0	
	Jackson 2.x	Apache 2.0	
	Commons-lang3	Apache 2.0	

	HttpComponents	Apache 2.0	
	Javax	CDDL + GPLv2 with classpath exception	
	RESTeasy	Apache 2.0	
Security management module	WildFly	LGPL 2.1	GPL
	JQuery	MIT	
	Lodash	MIT	
	AngularJS	MIT	
	Bootstrap	MIT	
	AdminLTE	MIT	
	HammerJS	MIT	
	Fontawesome	MIT	
PPI gateway	WildFly	LGPL 2.1	GPL
	RESTeasy	Apache 2.0	
	Jackson 2.x	Apache 2.0	
	Commons-lang3	Apache 2.0	
	HttpComponents	Apache 2.0	
	Javax	CDDL + GPLv2 with classpath exception	

6 USE

6.1 IoT Data Adapter

IoT Data Adapter offers a web-based interface (shown in Figure 18) through which the IoT systems connected to VITAL can be managed (e.g. register/deregister, enable/disable, change settings, refresh metadata). The interface is available at [http://\[host\]:\[port\]/iot-data-adapter](http://[host]:[port]/iot-data-adapter), where **host** and **port** are the host and port where the WildFly instance, where the IoT Data Adapter is deployed listens.

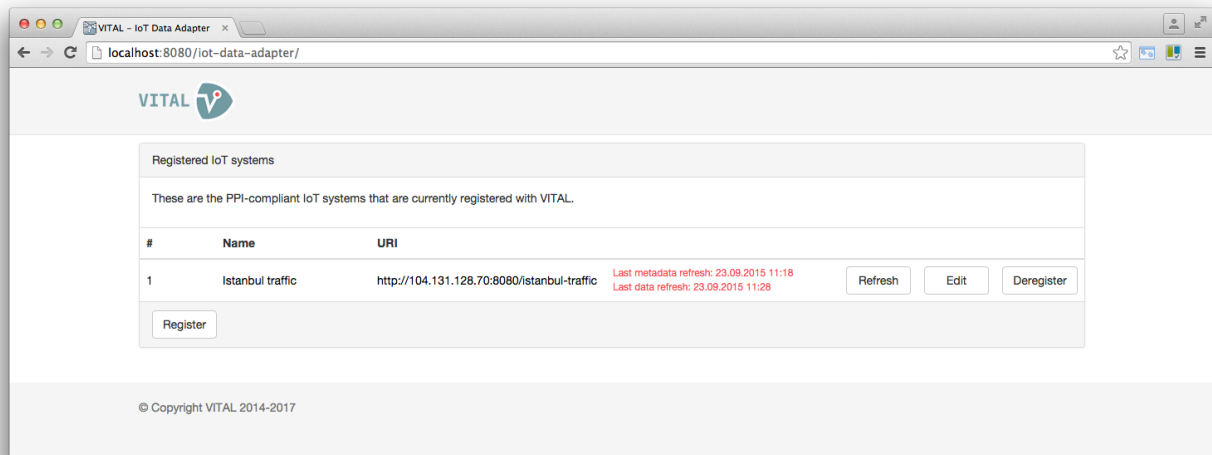


Figure 18 IoT Data Adapter web interface.

It also provides a web service that can be used to retrieve information about the IoT systems that are currently registered in the VITAL platform. Details about this web service can be found in [D3.2.2].

Finally, the PPI implementation that IoT Data Adapter offers mainly for monitoring purposes can be accessed as described in [D3.2.2].

6.2 Data Management Services (DMS)

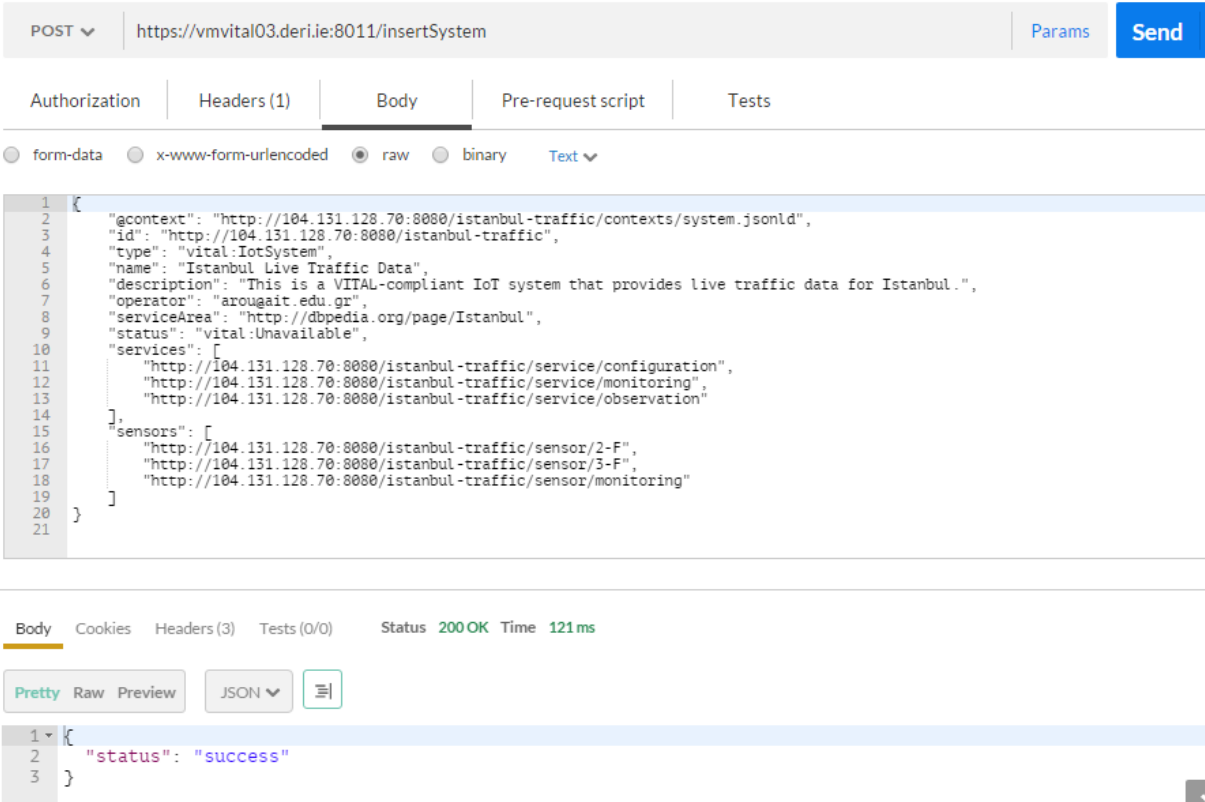
There are four entities modelled in VITAL platform: system, service, sensor, and observation. In order to allow IoT Data Adapter to store information about these entities into the DMS, following interfaces are provided:

- A. insertSystem
- B. insertService
- C. insertSensor
- D. insertObservation

These interfaces are provided to IoT Data Adapter for pushing data retrieved from various supported platforms via the PPI implementation that each of these platforms exposes.

In order to allow access to metadata and observations stored in DMS, four interfaces are provided. VITAL applications and components, such as filtering, CEP, Orchestrator, and Resource Discovery, use this interface to access stored metadata and observations. The request body is a JSON object that contains the query parameters based on the MongoDB query language. Upon a valid request the interfaces return data in JSON-LD (or JSON) format. In order to allow VITAL modules to query metadata and data stored in DMS, following interfaces are provided:

- A. querySystem
- B. queryService
- C. querySensor
- D. queryObservation



POST ▼ | <https://vmvital03.deri.ie:8011/insertSystem> | Params | **Send**

Authorization | Headers (1) | **Body** | Pre-request script | Tests


☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary Text ▼

```

1 {
2   "@context": "http://104.131.128.70:8080/istanbul-traffic/contexts/system.jsonld",
3   "id": "http://104.131.128.70:8080/istanbul-traffic",
4   "type": "vital:IotSystem",
5   "name": "Istanbul Live Traffic Data",
6   "description": "This is a VITAL-compliant IoT system that provides live traffic data for Istanbul.",
7   "operator": "arou@ait.edu.gr",
8   "serviceArea": "http://dbpedia.org/page/Istanbul",
9   "status": "vital:Unavailable",
10  "services": [
11    "http://104.131.128.70:8080/istanbul-traffic/service/configuration",
12    "http://104.131.128.70:8080/istanbul-traffic/service/monitoring",
13    "http://104.131.128.70:8080/istanbul-traffic/service/observation"
14  ],
15  "sensors": [
16    "http://104.131.128.70:8080/istanbul-traffic/sensor/2-F",
17    "http://104.131.128.70:8080/istanbul-traffic/sensor/3-F",
18    "http://104.131.128.70:8080/istanbul-traffic/sensor/monitoring"
19  ]
20 }
21

```

Body | Cookies | Headers (3) | Tests (0/0) | Status **200 OK** Time **121 ms**

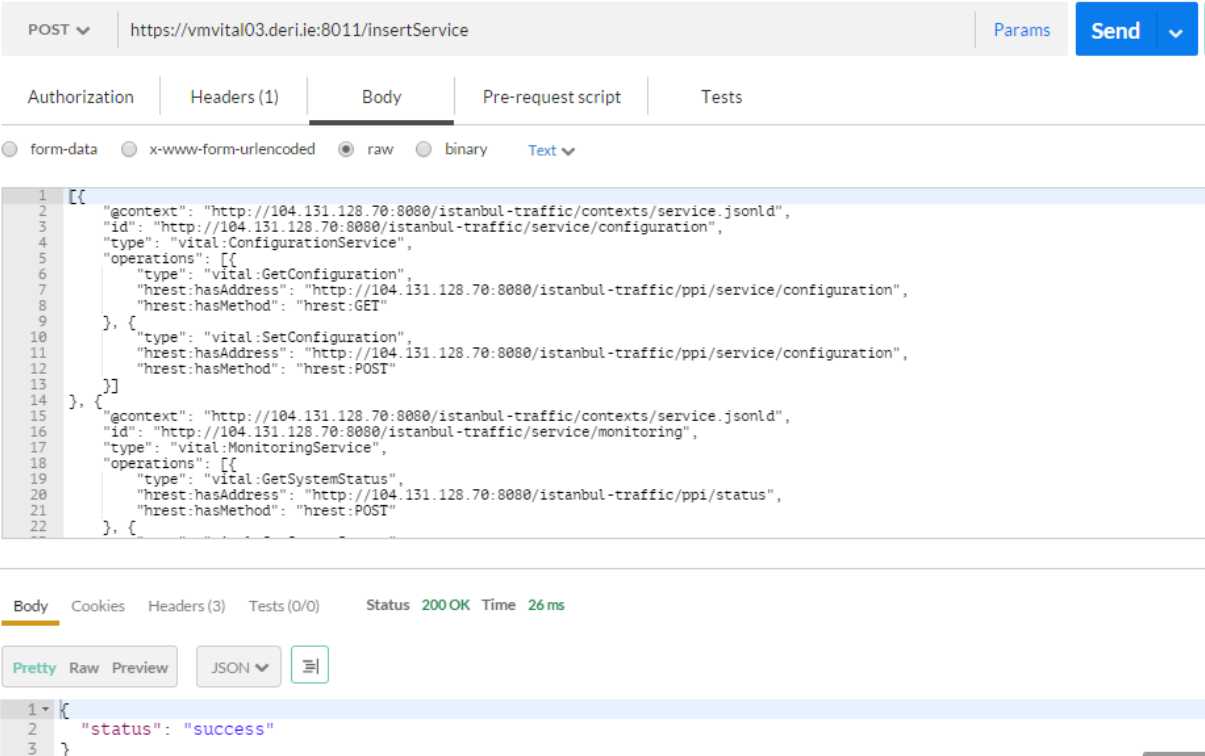
Pretty Raw Preview | JSON ▼ | 

```

1 {
2   "status": "success"
3 }

```

Figure 19 insertSystem to DMS.



POST ▼ | <https://vmvital03.deri.ie:8011/insertService> | Params | **Send** ▼

Authorization | Headers (1) | **Body** | Pre-request script | Tests


☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary Text ▼

```

1 [{
2   "@context": "http://104.131.128.70:8080/istanbul-traffic/contexts/service.jsonld",
3   "id": "http://104.131.128.70:8080/istanbul-traffic/service/configuration",
4   "type": "vital:ConfigurationService",
5   "operations": [
6     {
7       "type": "vital:GetConfiguration",
8       "hrest:hasAddress": "http://104.131.128.70:8080/istanbul-traffic/ppi/service/configuration",
9       "hrest:hasMethod": "hrest:GET"
10    },
11    {
12       "type": "vital:SetConfiguration",
13       "hrest:hasAddress": "http://104.131.128.70:8080/istanbul-traffic/ppi/service/configuration",
14       "hrest:hasMethod": "hrest:POST"
15    }
16  ],
17   "@context": "http://104.131.128.70:8080/istanbul-traffic/contexts/service.jsonld",
18   "id": "http://104.131.128.70:8080/istanbul-traffic/service/monitoring",
19   "type": "vital:MonitoringService",
20   "operations": [
21     {
22       "type": "vital:GetSystemStatus",
23       "hrest:hasAddress": "http://104.131.128.70:8080/istanbul-traffic/ppi/status",
24       "hrest:hasMethod": "hrest:POST"
25    }
26  ]
27 }], {

```

Body | Cookies | Headers (3) | Tests (0/0) | Status **200 OK** Time **26 ms**

Pretty Raw Preview | JSON ▼ | 

```

1 {
2   "status": "success"
3 }

```

Figure 20 insertService to DMS.

The screenshot shows a REST client interface with a POST request to `https://vmvital03.deri.ie:8011/insertSensor`. The request body is a JSON array of two sensor objects. The first sensor has a type of `vital:VitalSensor`, a description of `"A traffic sensor in Istanbul."`, and a status of `"vital:Unavailable"`. It also includes a `hasLastKnownLocation` object with `geo:Point` type and coordinates. The second sensor is similar but has a different ID and name. The response status is `200 OK` with a time of `1251ms`. The response body is a JSON object with `"status": "success"`.

```

1 [{"@context": "http://104.131.128.70:8080/istanbul-traffic/contexts/sensor.jsonld",
2   "type": "vital:VitalSensor",
3   "description": "A traffic sensor in Istanbul.",
4   "status": "vital:Unavailable",
5   "hasLastKnownLocation": {
6     "type": "geo:Point",
7     "geo:lat": 41.09301817,
8     "geo:lon": 29.0270595
9   },
10  "id": "http://104.131.128.70:8080/istanbul-traffic/sensor/2-F",
11  "name": "TEM Karanfilköy ( Forward Direction )",
12  "ssn:observes": [{
13    "type": "vital:Speed",
14    "id": "http://104.131.128.70:8080/istanbul-traffic/sensor/2-F/speed"
15  }],
16 }, {
17   "@context": "http://104.131.128.70:8080/istanbul-traffic/contexts/sensor.jsonld",
18   "type": "vital:VitalSensor",
19   "description": "A traffic sensor in Istanbul.",
20   "status": "vital:Unavailable",
21   "hasLastKnownLocation": {
22

```

```

1 {
2   "status": "success"
3 }

```

Figure 21 insertSensor to DMS.

The screenshot shows a REST client interface with a POST request to `https://vmvital03.deri.ie:8011/insertObservation`. The request body is a JSON object representing an observation. It includes a `ssn:observationResultTime` with a date-time, a `ssn:featureOfInterest` pointing to a traffic context, and a `ssn:observationResult` with a `ssn:SensorOutput` type and a `"vital:Unavailable"` value. The response status is `200 OK` with a time of `42ms`. The response body is a JSON object with `"status": "success"`.

```

1 {
2   "@context": "http://104.131.128.70:8080/istanbul-traffic/contexts/measurement.jsonld",
3   "id": "http://104.131.128.70:8080/istanbul-traffic/sensor/monitoring/observation/status/1435743568541",
4   "type": "ssn:Observation",
5   "ssn:observedBy": "http://104.131.128.70:8080/istanbul-traffic/sensor/monitoring",
6   "ssn:observationProperty": {
7     "type": "vital:OperationalState"
8   },
9   "ssn:observationResultTime": {
10    "time:inXSDDateTime": "2015-07-01T09:39:28Z"
11  },
12  "ssn:featureOfInterest": "http://104.131.128.70:8080/istanbul-traffic",
13  "ssn:observationResult": {
14    "type": "ssn:SensorOutput",
15    "ssn:hasValue": {
16      "type": "ssn:ObservationValue",
17      "value": "vital:Unavailable"
18    }
19  }
20 }
21

```

```

1 {
2   "status": "success"
3 }

```

Figure 22 insertObservation to DMS.

The screenshot displays a REST client interface. At the top, a POST request is configured to the URL `https://vmvital03.deri.ie:8011/querySystem`. The request body is set to 'raw' and contains the following JSON:

```
1 {
2   "@type": "http://vital-iot.eu/ontology/ns/IotSystem"
3 }
4
```

The response status is 200 OK, and the response body is shown in 'Pretty' JSON format:

```
1 [
2   {
3     "@context": "http://104.131.128.70:8080/istanbul-traffic/contexts/system.jsonld",
4     "id": "http://104.131.128.70:8080/istanbul-traffic",
5     "type": "vital:IotSystem",
6     "name": "Istanbul Live Traffic Data",
7     "description": "This is a VITAL-compliant IoT system that provides live traffic data for Istanbul.",
8     "operator": "arouga.it.edu.gr",
9     "serviceArea": "http://dbpedia.org/page/Istanbul",
10    "status": "vital:Unavailable",
11    "services": [
12      "http://104.131.128.70:8080/istanbul-traffic/service/configuration",
13      "http://104.131.128.70:8080/istanbul-traffic/service/monitoring",
14      "http://104.131.128.70:8080/istanbul-traffic/service/observation"
15    ],
16    "sensors": [
17      "http://104.131.128.70:8080/istanbul-traffic/sensor/2-F",
18      "http://104.131.128.70:8080/istanbul-traffic/sensor/3-F",
19      "http://104.131.128.70:8080/istanbul-traffic/sensor/monitoring"
20    ]
21  }
22 ]
```

Figure 23 querySystem from DMS.

The screenshot displays a REST client interface with the following details:

- Method:** POST
- URL:** `https://vmvital03.deri.ie:8011/queryService`
- Params:** (empty)
- Send:** (button)
- Authorization:** (tab)
- Headers (1):** (tab)
- Body:** (active tab)
 - Form: `form-data` (selected), `x-www-form-urlencoded`, `raw`, `binary`
 - Text: (dropdown menu)
 - Content: `{"@type" : "http://vital-iot.eu/ontology/ns/ConfigurationService"}`
- Pre-request script:** (tab)
- Tests:** (tab)
- Status:** 200 OK
- Time:** 1782 ms
- Body:** (active tab)
 - Format: `Pretty` (selected), `Raw`, `Preview`
 - JSON: (dropdown menu)
 - Content:


```

1  [
2    {
3      "@context": "http://104.131.128.70:8080/istanbul-traffic/contexts/service.jsonld",
4      "id": "http://104.131.128.70:8080/istanbul-traffic/service/configuration",
5      "type": "vital:ConfigurationService",
6      "operations": [
7        {
8          "type": "vital:GetConfiguration",
9          "hrest:hasAddress": "http://104.131.128.70:8080/istanbul-traffic/ppi/service/configuration",
10         "hrest:hasMethod": "hrest:GET"
11       },
12       {
13         "type": "vital:SetConfiguration",
14         "hrest:hasAddress": "http://104.131.128.70:8080/istanbul-traffic/ppi/service/configuration",
15         "hrest:hasMethod": "hrest:POST"
16       }
17     ]
18   }
19 ]
          
```

Figure 24 queryService from DMS.

The screenshot displays a REST client interface with a POST request to `https://vmvital03.deri.ie:8011/querySensor`. The request body is a JSON object with nested property matches. The response status is 200 OK, and the response body is a JSON array of two sensor objects.

Request:

```
POST https://vmvital03.deri.ie:8011/querySensor
```

Request Body (raw):

```
{
  "http://vital-iot.eu/ontology/ns/hasLastKnownLocation": {
    "$elemMatch": {
      "http://www.w3.org/2003/01/geo/wgs84_pos#lat": {
        "$elemMatch": {
          "@value": {
            "$gt": 51
          }
        }
      }
    }
  }
}
```

Response: Status 200 OK, Time 47 ms

Response Body (JSON):

```
[
  {
    "@context": "http://104.131.128.70:8080/istanbul-traffic/contexts/sensor.jsonld",
    "type": "vital:VitalSensor",
    "description": "A traffic sensor in Istanbul.",
    "status": "vital:Unavailable",
    "hasLastKnownLocation": {
      "type": "geo:Point",
      "geo:lat": 41.09977412,
      "geo:lon": 29.00628378
    },
    "id": "http://104.131.128.70:8080/istanbul-traffic/sensor/13-F",
    "name": "Sensor with ID 13 ( Forward Direction )",
    "ssn:observes": [
      {
        "type": "vital:Speed",
        "id": "http://104.131.128.70:8080/istanbul-traffic/sensor/13-F/speed"
      }
    ]
  },
  {
    "@context": "http://104.131.128.70:8080/istanbul-traffic/contexts/sensor.jsonld",
    "type": "vital:VitalSensor",
    "description": "A traffic sensor in Istanbul.",
    "status": "vital:Unavailable",
    "hasLastKnownLocation": {
      "type": "geo:Point",
      "geo:lat": 41.10021035,
      "geo:lon": 28.98199436
    },
    "id": "http://104.131.128.70:8080/istanbul-traffic/sensor/20-F",
    "name": "TEM Arıcilar ( Forward Direction )",
    "ssn:observes": [
      {
        "type": "vital:Speed",

```

Figure 25 querySensor from DMS.

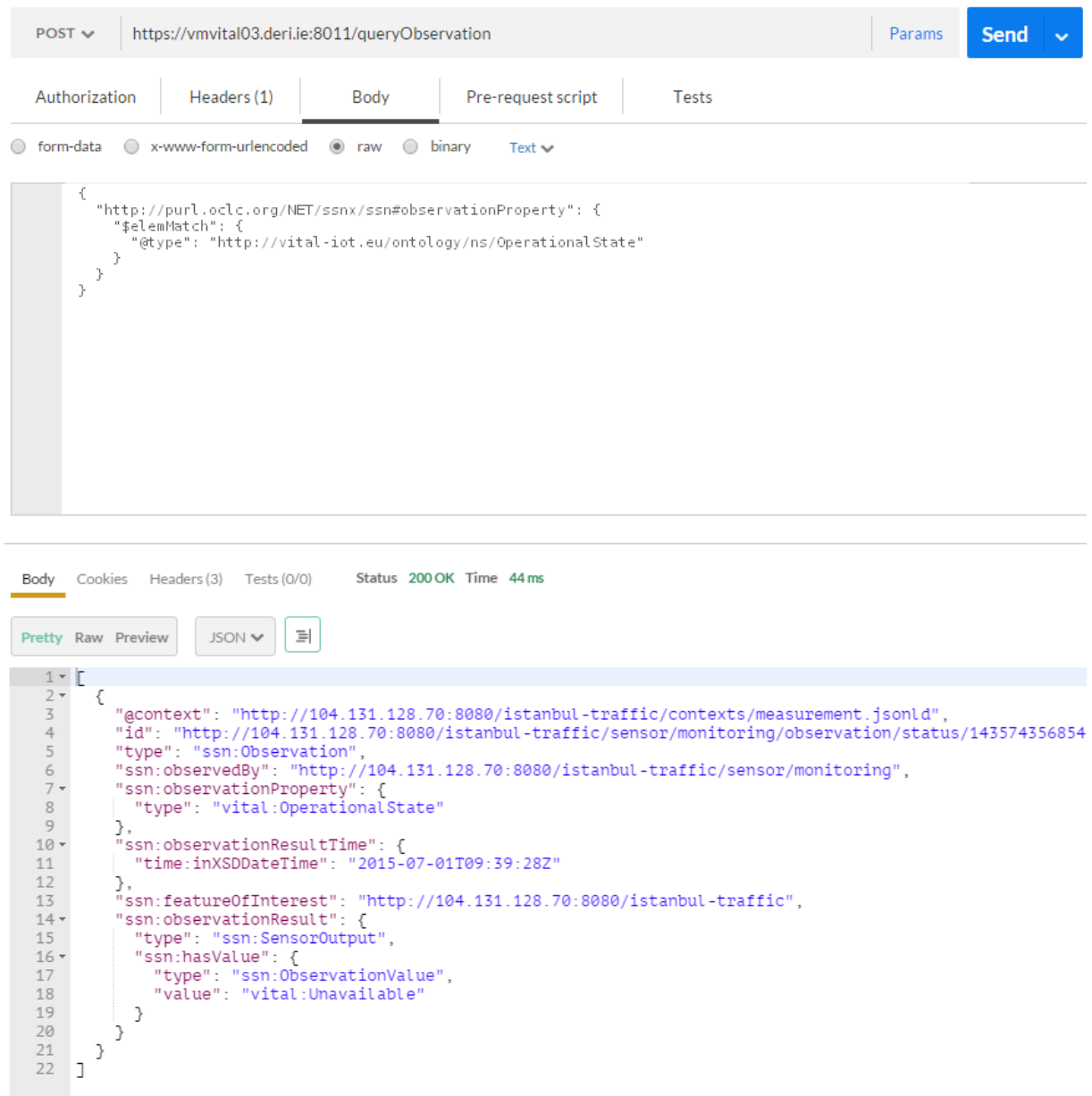


Figure 26 queryObservation from DMS.

6.3 Discovery service

The Discovery module is a RESTful web service which can be invoked through http requests. The web service root address depends on the web address where the server is running and accessible. In order to keep the addresses universal, we define “BASE_URL” as the url of the server hosting the service. As an example, we can consider the reference url “BASE_URL/discoverer”. Such reference url has to be translated accordingly e.g.:

- **Deployment on local machine:**
 - “localhost:8080/discoverer”
- **Deployment on public url “www.example.com”:**
 - “www.example.com/discoverer”

Once you successfully deployed the Web Service you can start to invoke his functions. The http request can take place in different ways. The easier way is through the web browser. For Linux or Unix-like operating system it is also possible to use the **curl** command. To request the reference url using curl command the syntax is as follows:

```
curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET  
BASE_URL/discoverer
```

Here follow some examples on how to request the main functionalities to the Discovery and tested with a simple web browser:

- *Test Connection to the DMS:* To verify if the Discovery is properly connected with the Data Management Service you can request the url “BASE_URL/discoverer/ConnDMS”. The http response will vary accordingly to the connection status. Here follows an example showing the response:

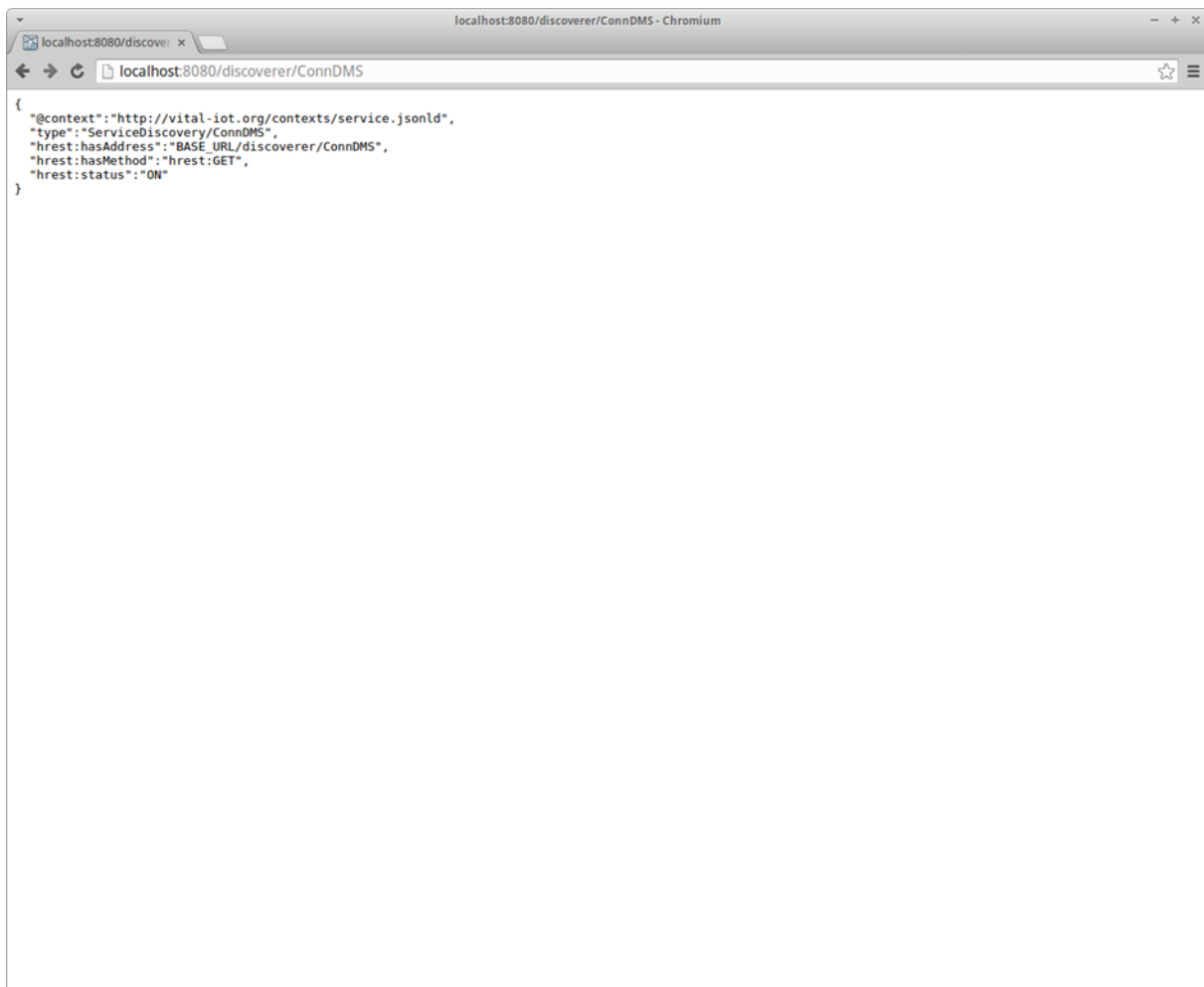


Figure 27 Result for ConnDMS function

- **Discovery Service description:** The web service provides a description of the services available to the users. This is the result of a request to the address “BASE_URL/discoverer”. The result is depicted in Figure 27 Result for ConnDMS function

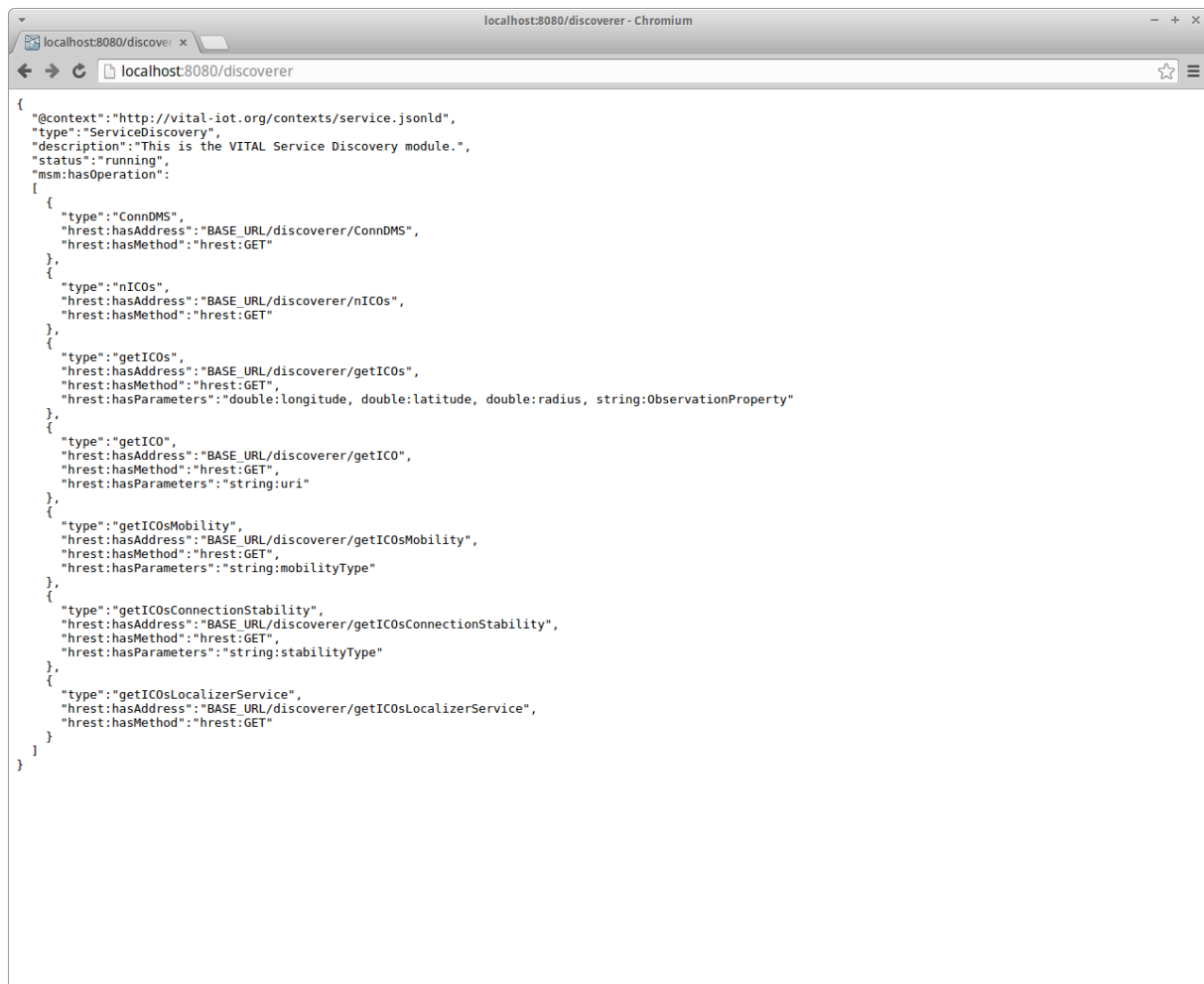


Figure 28 Discovery Service description

- Discover ICOs in an area: One of the primary functions of the Discovery module is to provide a list of the ICO available in a specific area. To access the service the url has to be formed as follows: "BASE_URL/discoverer/getICOs?latitude=LATITUDE&longitude=LONGITUDE&radius=RADIUS" where LATITUDE, LONGITUDE and RADIUS has to be replaced with the parameters of interest. An example of the response for this kind of request is shown in Figure 28.

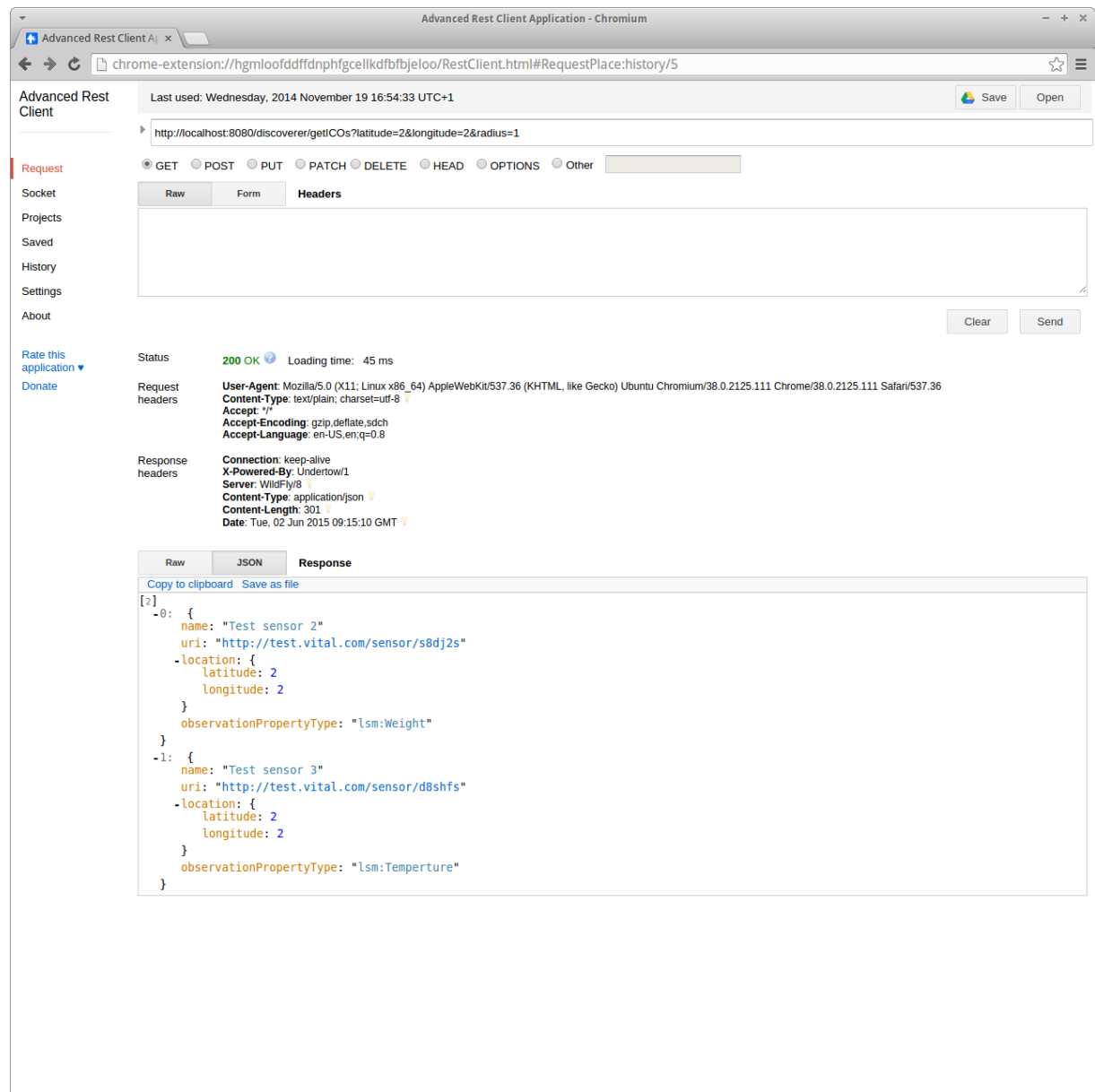


Figure 29 Result for getICOs function

6.4 Filtering Service

Filtering module is a RESTful web service which can be invoked through http requests. The web service root address depends on the web address where the server is running and accessible. In order to keep the addresses universal, we define “BASE_URL” as the url of the server hosting the service. As an example, we can consider the reference url “BASE_URL/filtering”. Such reference url has to be translated accordingly e.g.:

- **Deployment on local machine:**
 - “localhost:8080/filtering”
- **Deployment on public url “www.example.com”:**
 - “www.example.com/filtering”

Once you successfully deployed the Web Service you can start to invoke his functions. The http request can take place in different ways. The easier way is through the web browser. Metadata about Filtering module are accessible through a POST request on the address “BASE_URL/filtering/ppi/metadata”. Result is shown in Figure 30.

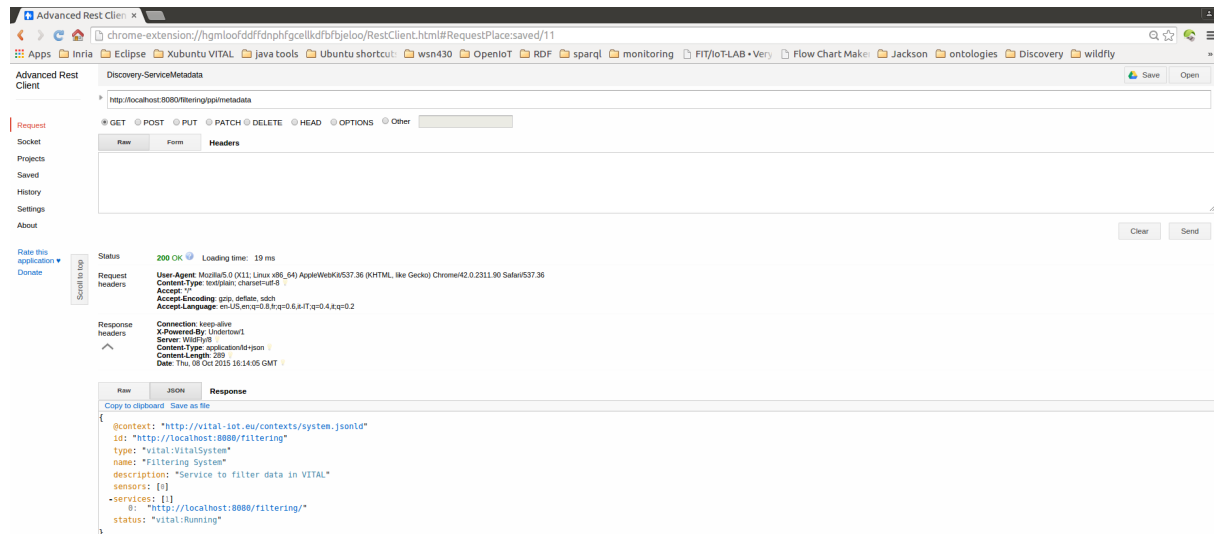
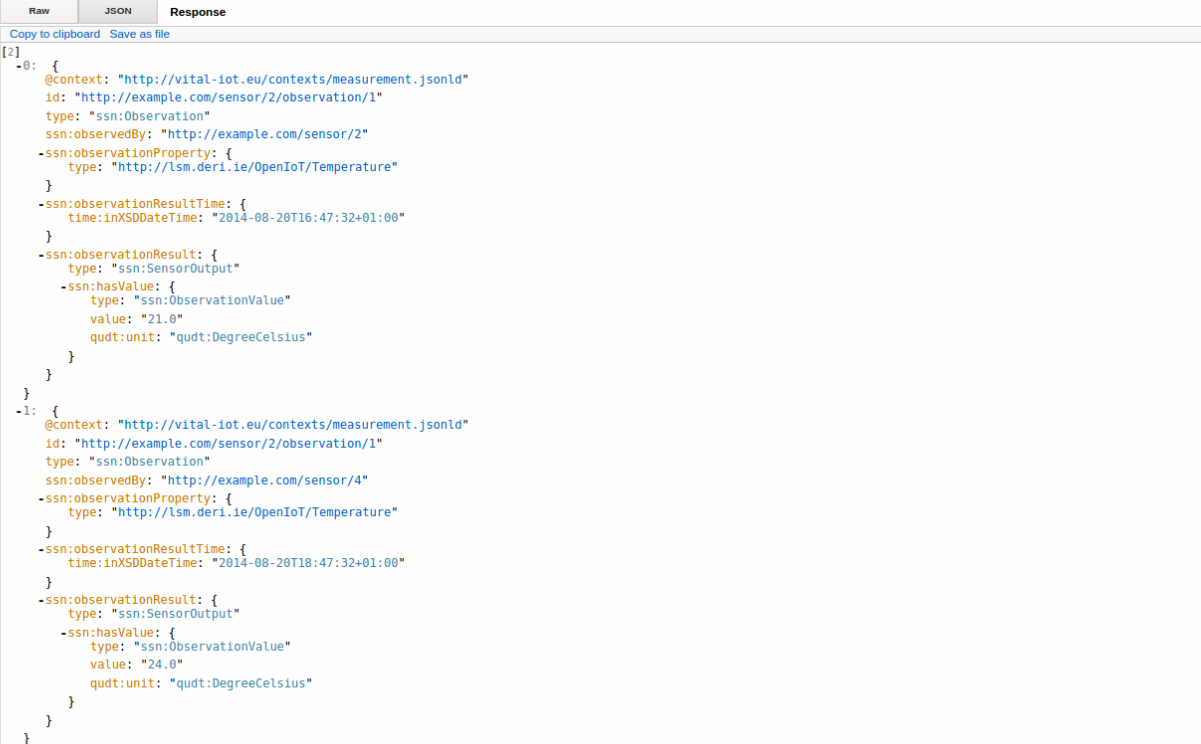


Figure 30 Filtering service description

One of the functionalities offered by the Filtering is to retrieve the observations whose values are over or under a certain threshold. To use such operation it is first necessary to configuring thresholds by sending a POST request to the address “BASE_URL/filtering/threshold”. For more details, check deliverable [D4.2.1]. An example of output is shown in Figure 31.



```

[2]
-0: {
  @context: "http://vital-iot.eu/contexts/measurement.jsonld"
  id: "http://example.com/sensor/2/observation/1"
  type: "ssn:Observation"
  ssn:observedBy: "http://example.com/sensor/2"
  -ssn:observationProperty: {
    type: "http://lsm.der1.ie/OpenIoT/Temperature"
  }
  -ssn:observationResultTime: {
    time:inXSDdateTime: "2014-08-20T16:47:32+01:00"
  }
  -ssn:observationResult: {
    type: "ssn:SensorOutput"
    -ssn:hasValue: {
      type: "ssn:ObservationValue"
      value: "21.0"
      qudt:unit: "qudt:DegreeCelsius"
    }
  }
}
-1: {
  @context: "http://vital-iot.eu/contexts/measurement.jsonld"
  id: "http://example.com/sensor/2/observation/1"
  type: "ssn:Observation"
  ssn:observedBy: "http://example.com/sensor/4"
  -ssn:observationProperty: {
    type: "http://lsm.der1.ie/OpenIoT/Temperature"
  }
  -ssn:observationResultTime: {
    time:inXSDdateTime: "2014-08-20T18:47:32+01:00"
  }
  -ssn:observationResult: {
    type: "ssn:SensorOutput"
    -ssn:hasValue: {
      type: "ssn:ObservationValue"
      value: "24.0"
      qudt:unit: "qudt:DegreeCelsius"
    }
  }
}
}

```

Figure 31 Result of filtering according to threshold

6.5 Orchestration Service

The VITAL Orchestrator comes with a visual tool (Orchestrator UI), which enables the discovery of available sensors and IoT services along with the management of operations and workflows. The Orchestrator UI is a web-based tool, which makes it ubiquitously accessible for application developers. The tool provides a visual interface over all the functions/functionalities of the VITAL orchestrator API, including those for defining and managing operations and workflows and those for defining and implementing operations/workflows based on the scripting language of the VITAL orchestrator [D4.4.1-2015].

Figure 32 illustrates how service developers can access the VITAL Orchestrator operations through the UI, while Figure 33 illustrates how an operation could be updated.

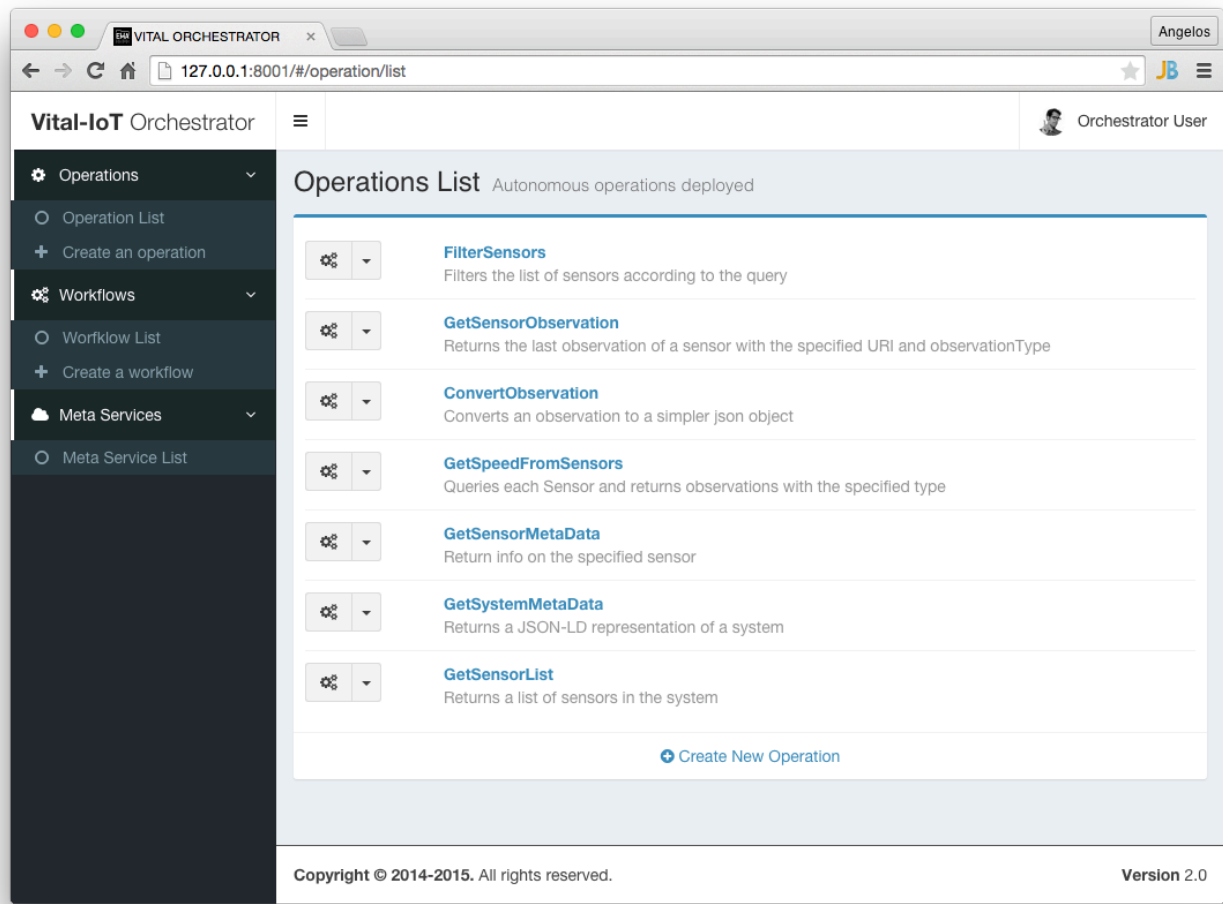


Figure 32 List of Operations in the VITAL Orchestrator UI

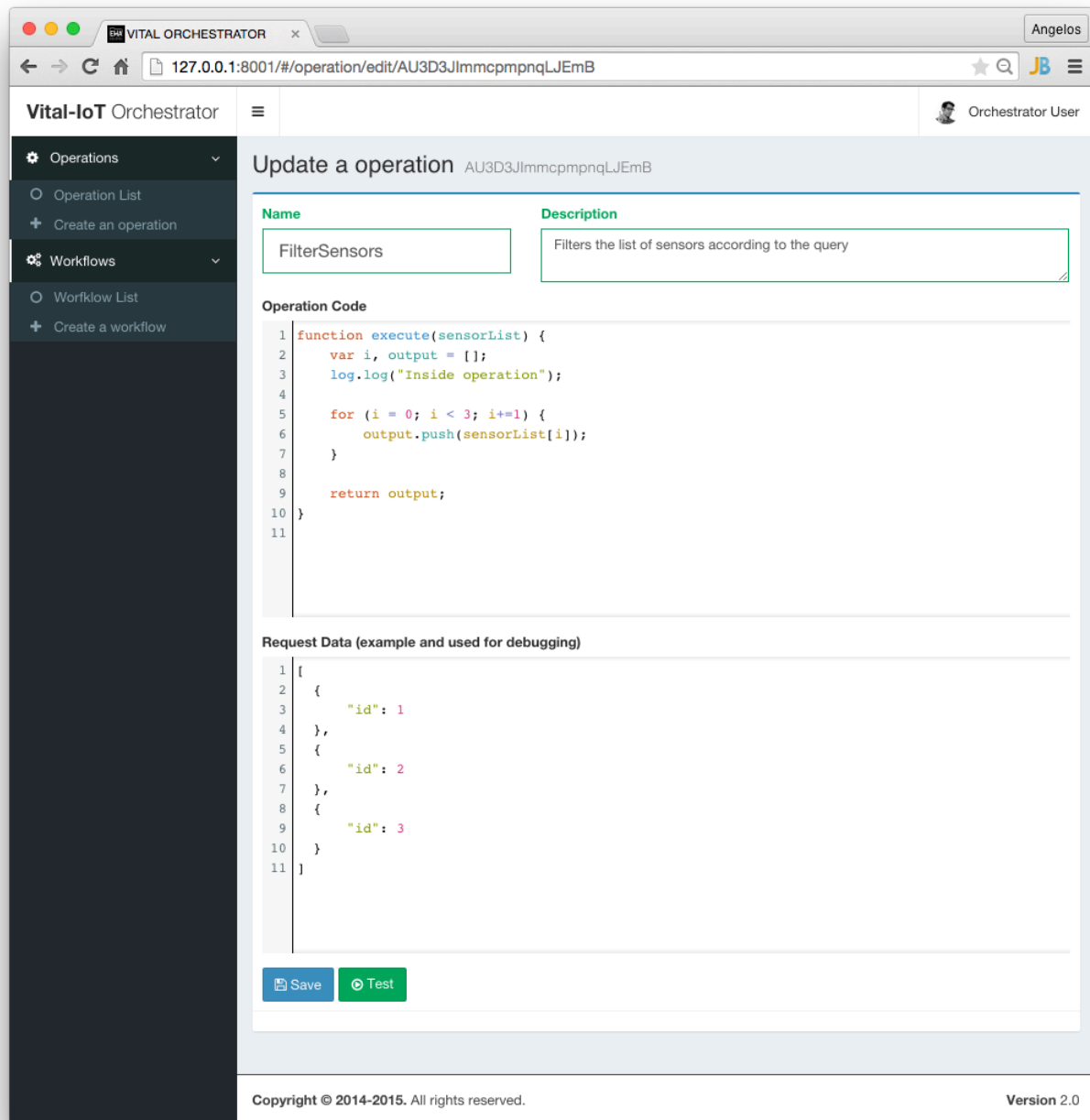


Figure 33 GUI for updating operations in the VITAL Orchestrator UI

The figure below displays the result of testing the operation.

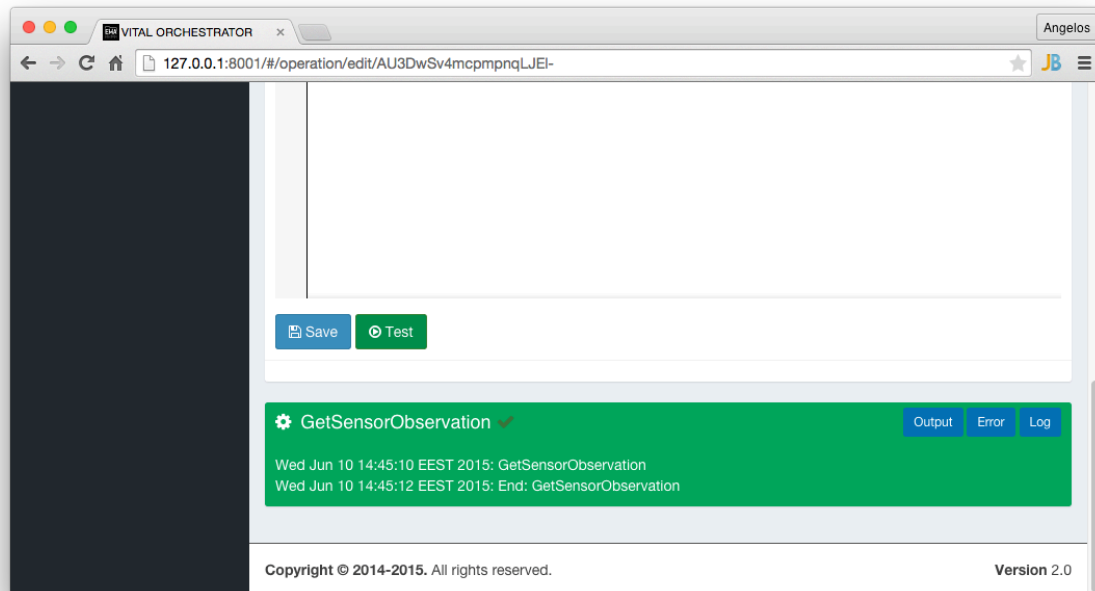


Figure 34 Result of executing an operation in the VITAL Orchestrator UI

Figure 35 and Figure 36 are snapshots of the VITAL Orchestrator UI, illustrating how a developer can visually manage workflows that are based on the orchestrator. A developer starts by adding operations to the workflow from the already defined operations. He can then edit the operations if he so chooses and, after saving, he can test the execution.

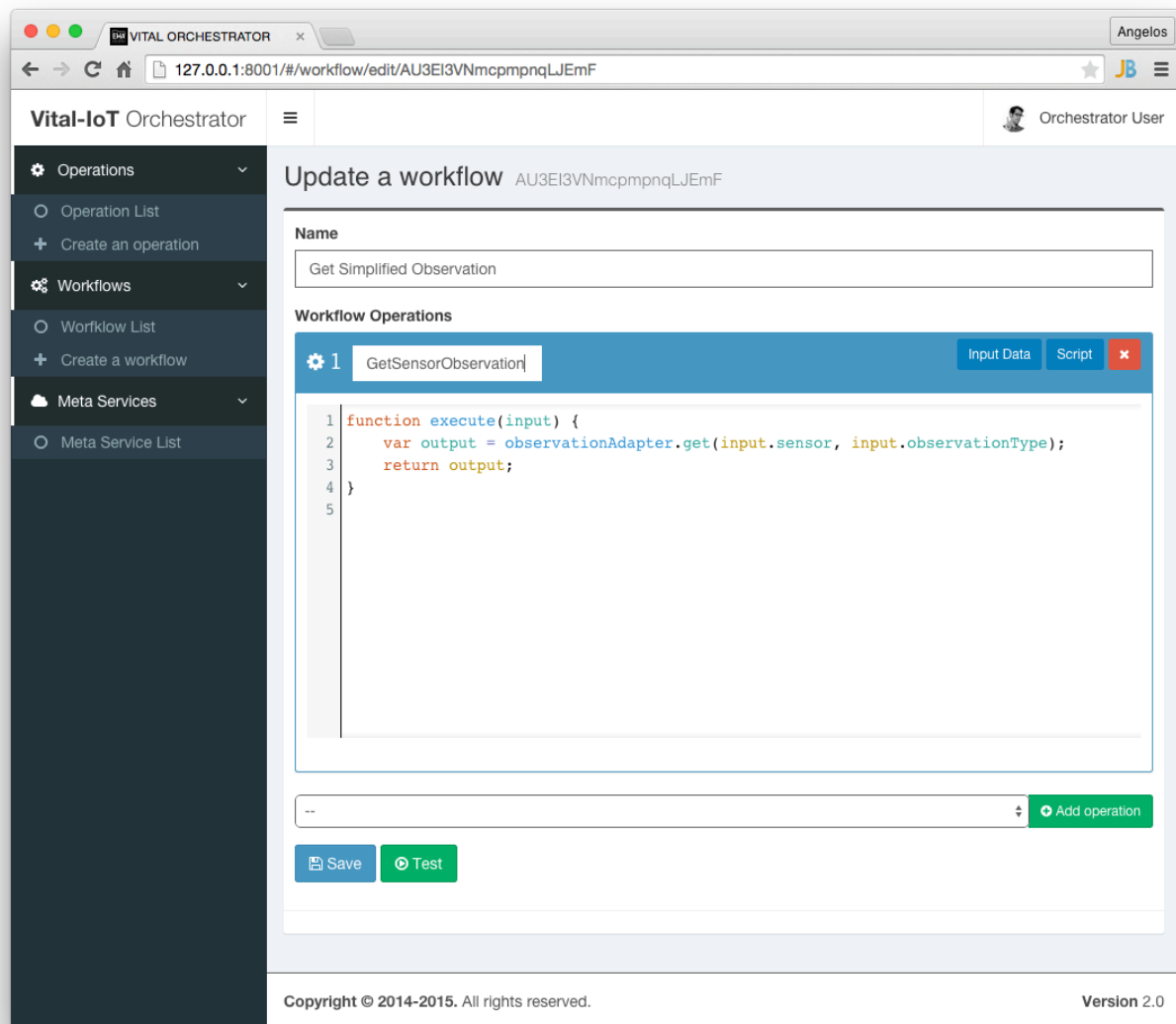


Figure 35 GUI for updating workflows in the VITAL Orchestrator UI (1)

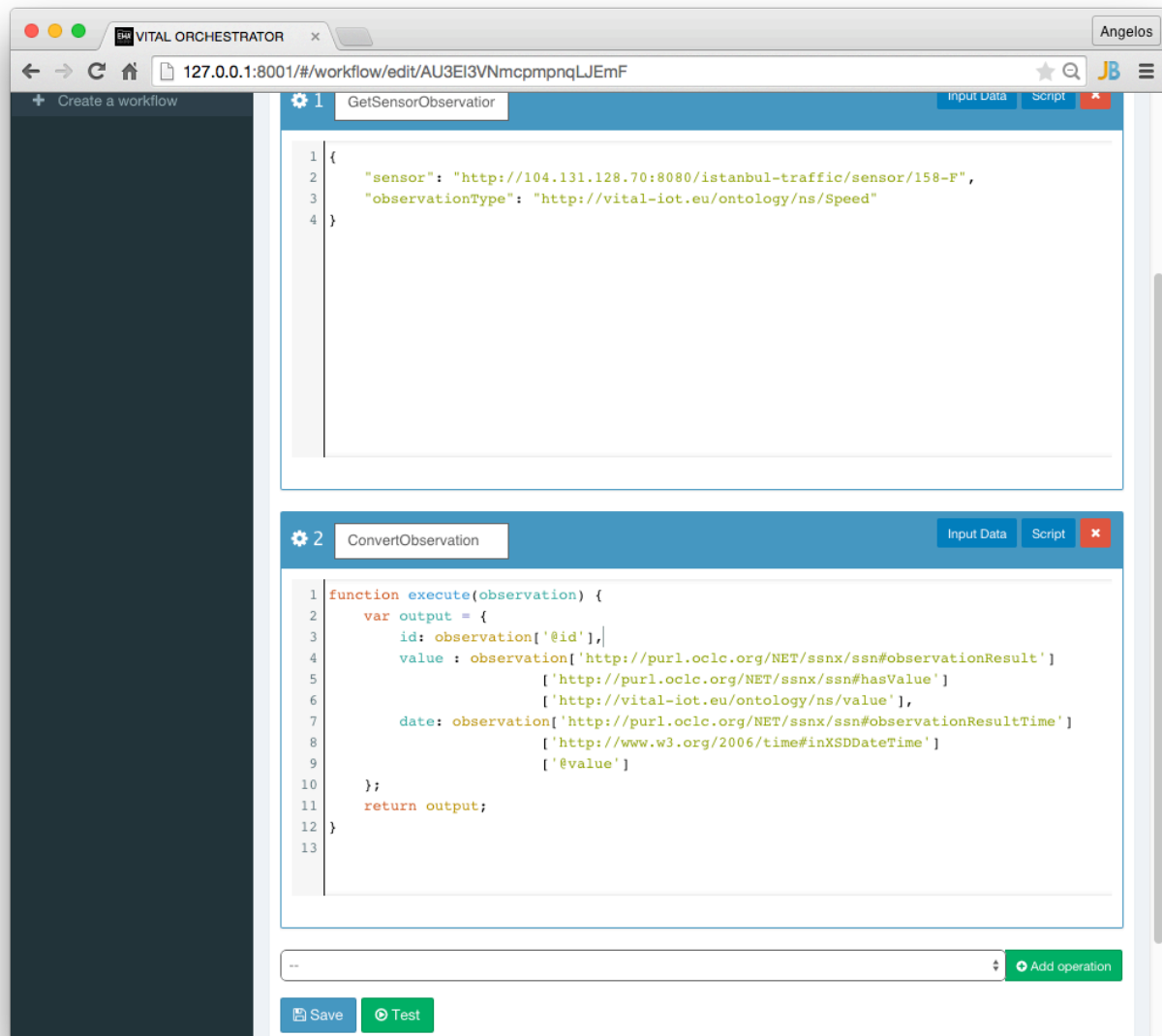


Figure 36 GUI for updating workflows in the VITAL Orchestrator UI (2)

In the figure below, the result of testing a workflow is illustrated.

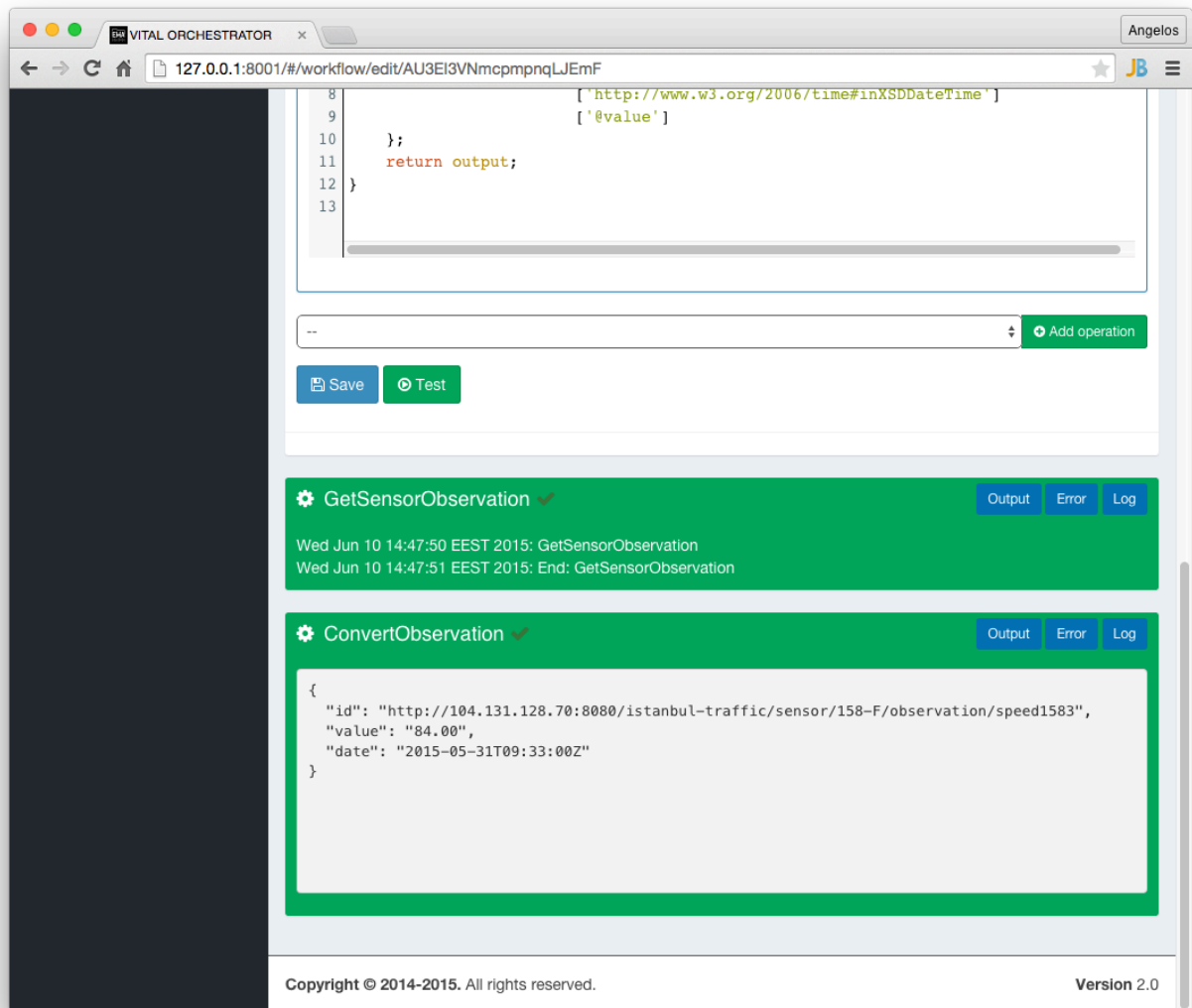


Figure 37 Workflow Execution Result in the VITAL Orchestrator UI

Figure 38 displays the list of deployed workflows. From this list, the developer can remove workflows, but can also deploy them as Meta Services.

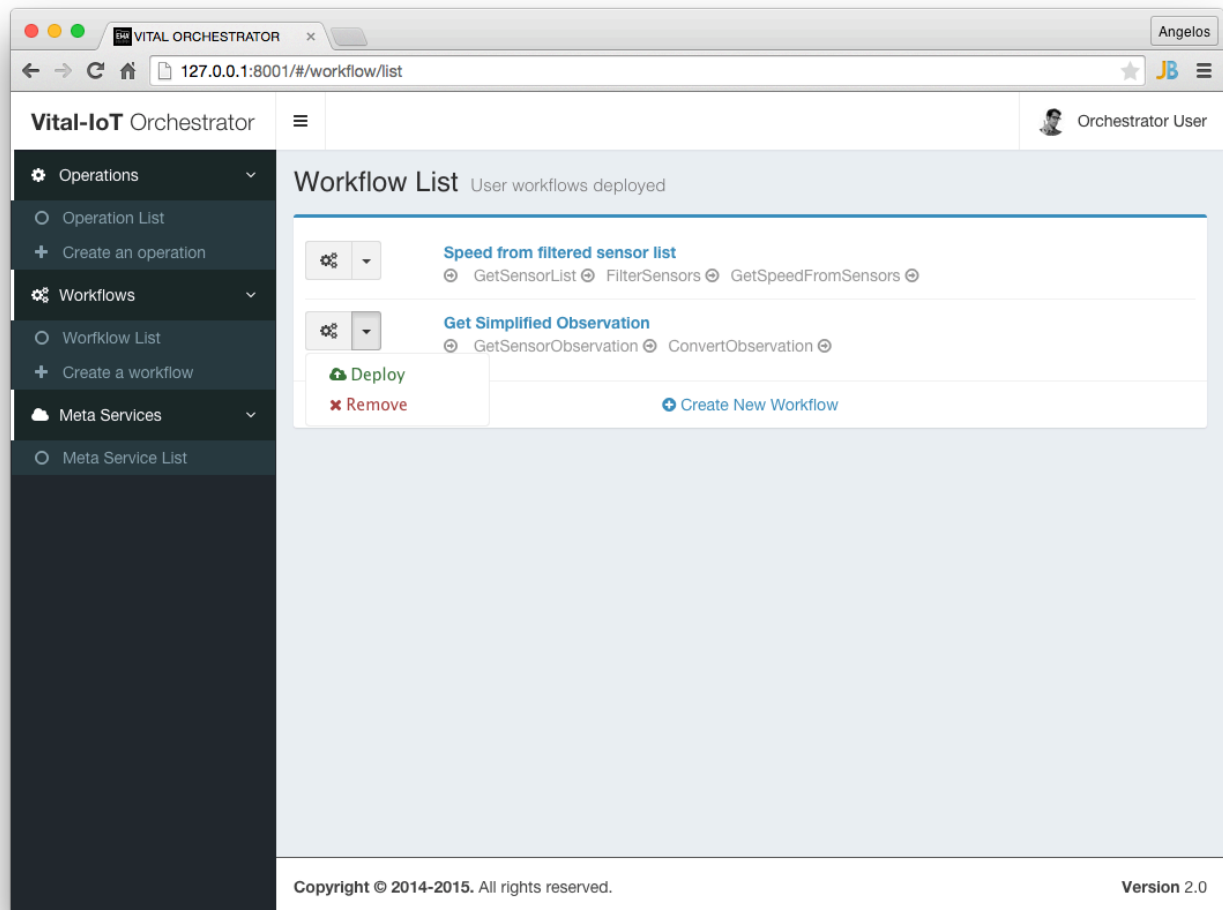


Figure 38 List of Workflows in the VITAL Orchestrator UI

The final two figures below depict the list of deployed services and the informational view of a single service.

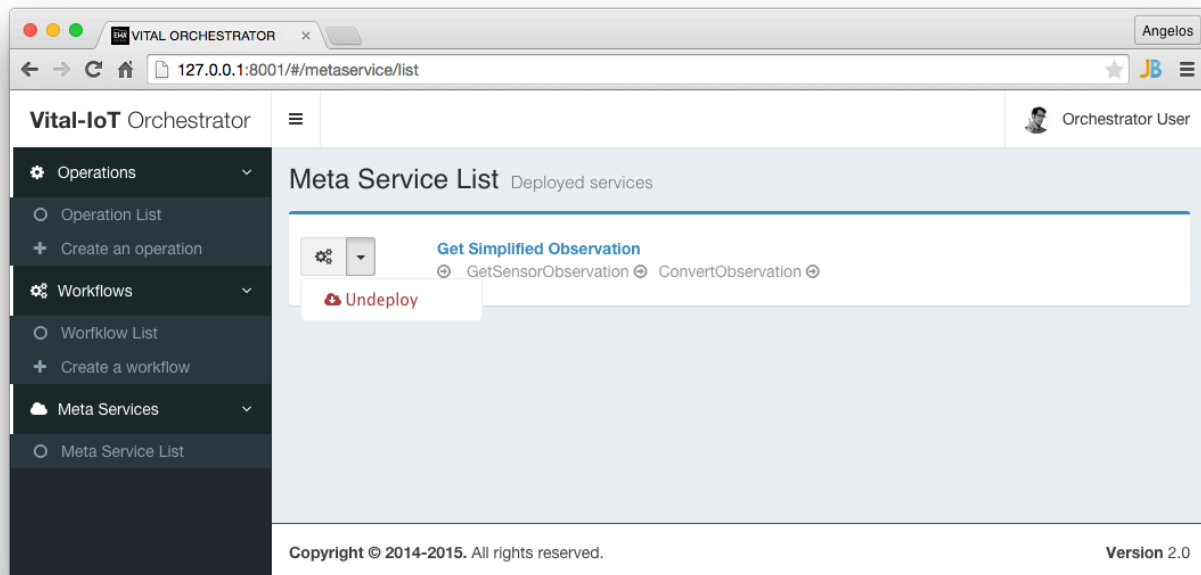


Figure 39 List of deployed meta service in the VITAL Orchestrator UI

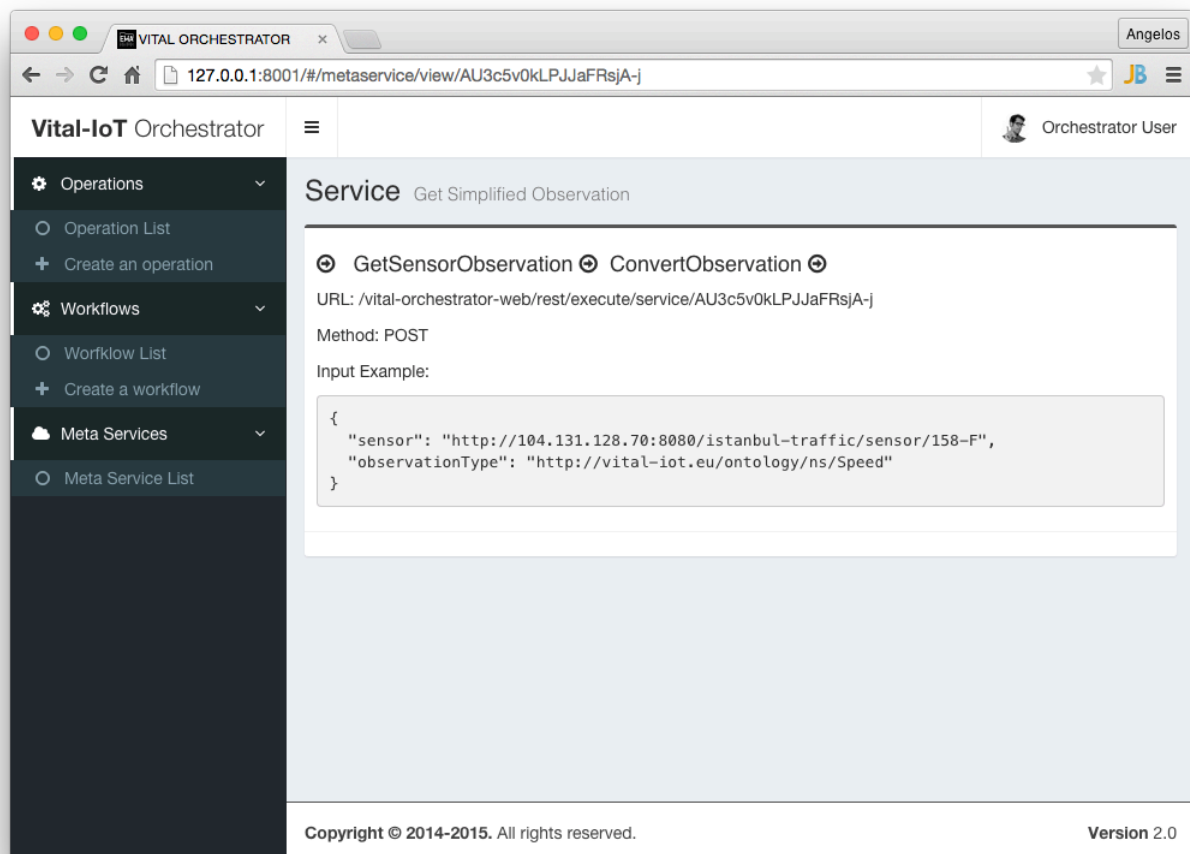


Figure 40 View of a single meta service in the VITAL Orchestrator UI

6.6 Complex Event Processing (VITALCEP)

The VITALCEP module provides a RESTful web service which can be invoked through HTTP requests. In this first version this service does not implements security mechanism. Using BASE_URL of VITALCEP, there are three main interfaces of CEP that can be used:

- getCEPICOS

This interface provide metadata of all instances of CEP

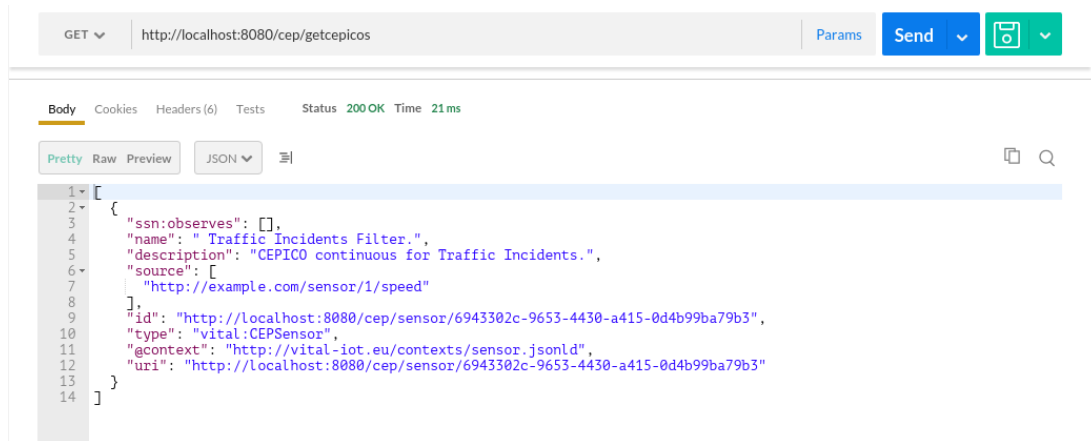


Figure 41 Retrieve VITALCEPs instances.

- getCEPICO

This interface provides the functionality to retrieve the metadata and the name of the DOLCE specification that is being used by the CEP in order to detect complex events.

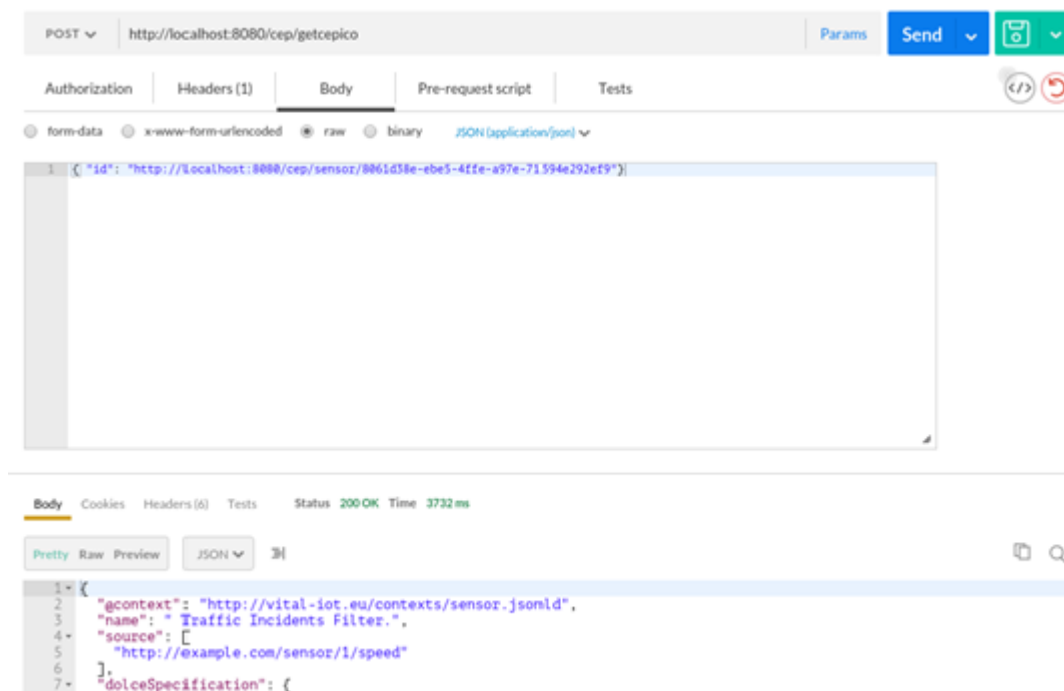


Figure 42 Retrieve a VITALCEP instance.

- createUpdateCEPICO

This interface provides the functionality of creating a new CEP instance through a POST request. A status message is returned.

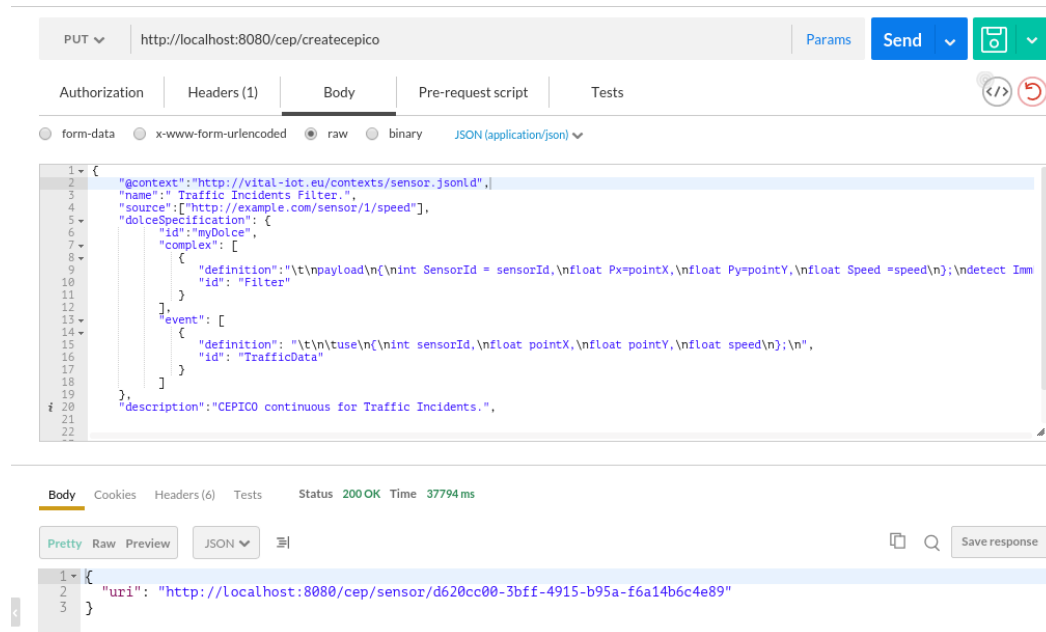


Figure 43 Update VITALCEP instance.

- deleteCEPICO

This interface is used to remove an instance of CEP once it is no longer needed. A status message is returned with the result of the operation.

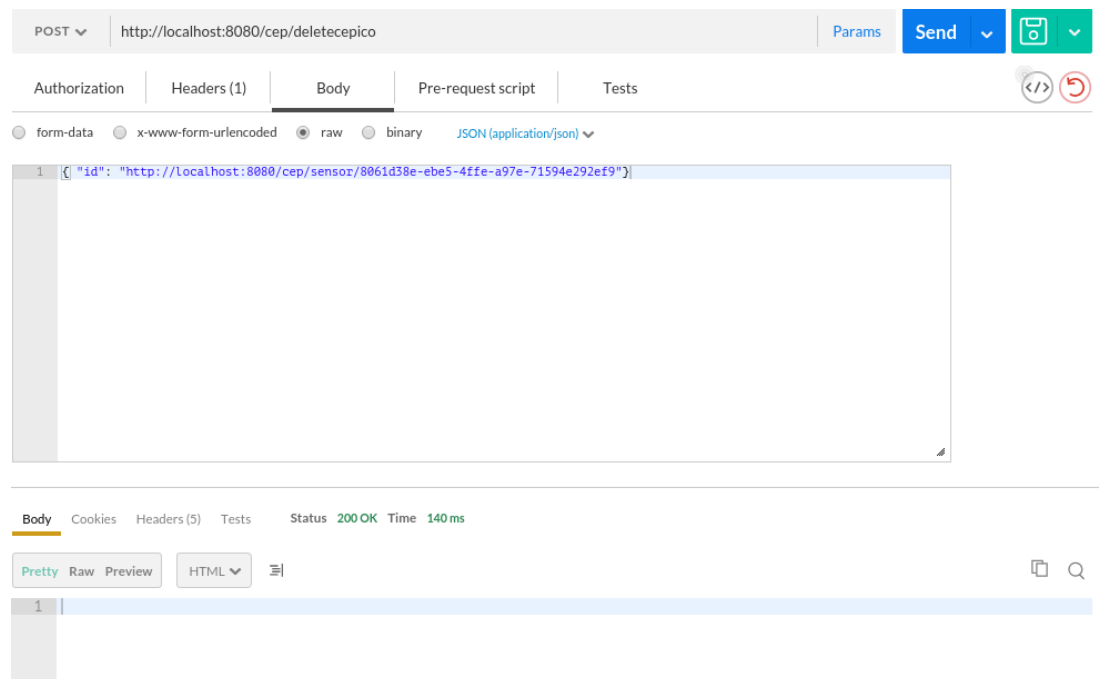


Figure 44 Delete VITALCEP instance.

6.7 Development Tools

The VITAL development and deployment environment is shown in Figure 45. On the left side of the tool is the palette of nodes that can be used to build workflows. In order to create a workflow, drag the nodes you need from the palette, drop them into the workspace (located in the centre of the editor), set their properties, and wire them together. In order to have more space to work, the tool supports multiple, tabbed workspaces. Once the workflows are ready, we can deploy them to the runtime, using the Deploy button located at the top of the editor.

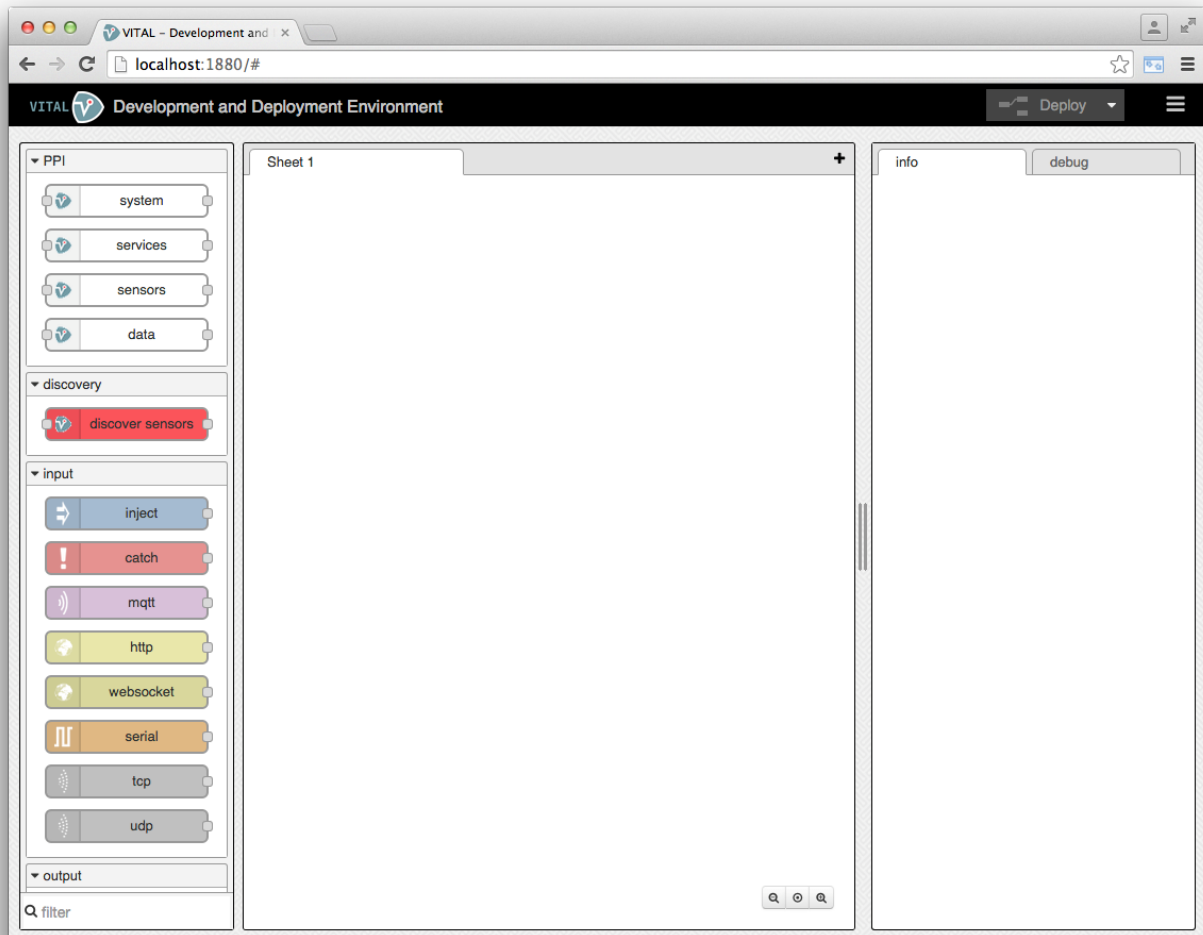


Figure 45 VITAL development and deployment environment.

More details about how to develop and use applications using the development tools can be found in [D5.2.1].

6.8 Management Tools

The VITAL Management Toolkit is a web-based application with a rest/ld+json backend. From a user/administrator perspective, only a browser is required to access the platform's functionality.

Given the url of the installation a user needs to open the browser and enter the url in the location bar. From there he can access the following functions / views:

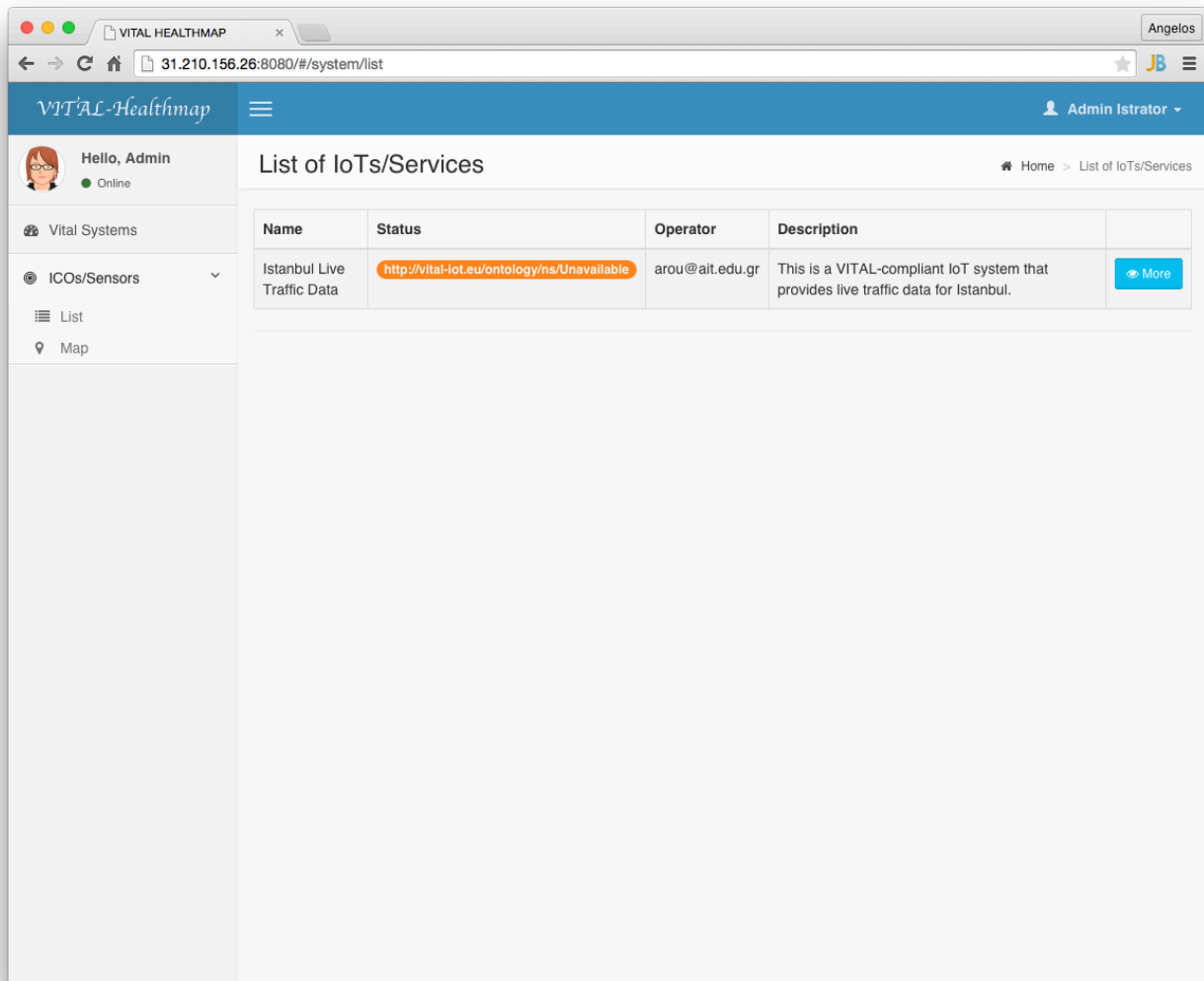


Figure 46 Services Dashboard.

List of Systems, available in the VITAL ecosystem with their current status.

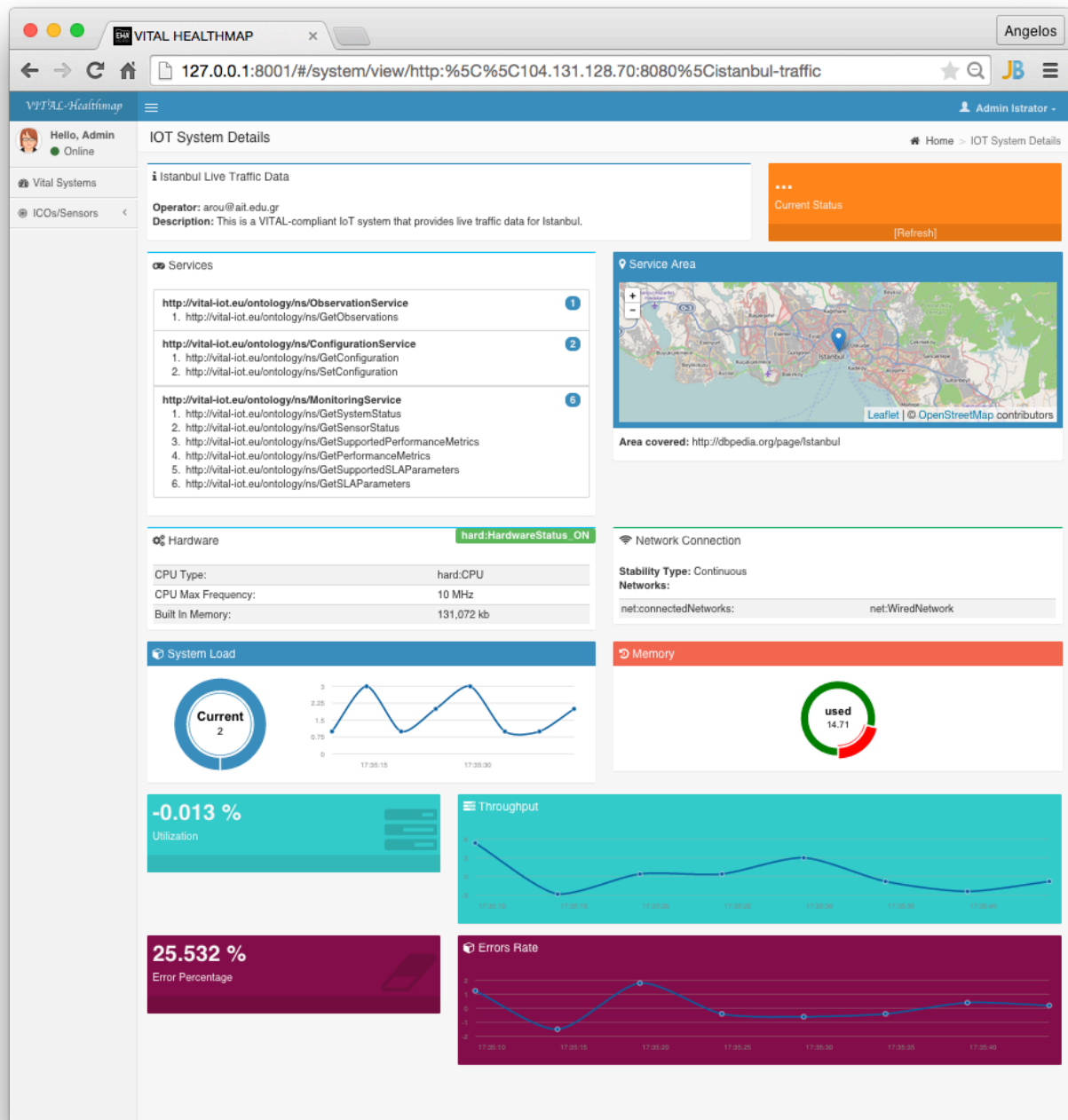
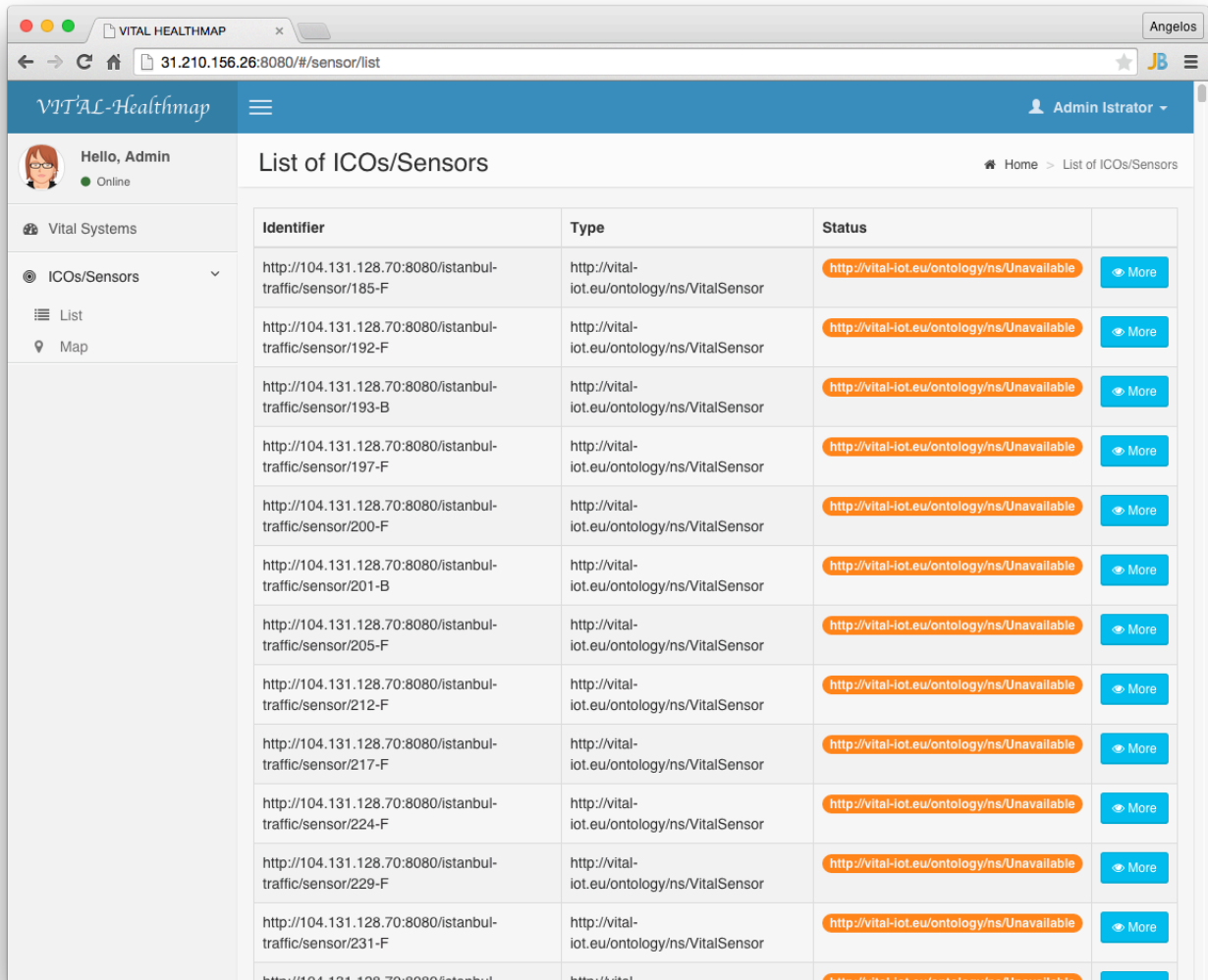


Figure 47 IoT system dashboard.

It displays generic information, services supported, area, current status and charts with performance metrics if they are reported by the system.



VITAL Healthmap | Hello, Admin | Online | Admin Istrator

List of ICOs/Sensors

Identifier	Type	Status	More
http://104.131.128.70:8080/istanbul-traffic/sensor/185-F	http://vital-iot.eu/ontology/ns/VitalSensor	http://vital-iot.eu/ontology/ns/Unavailable	More
http://104.131.128.70:8080/istanbul-traffic/sensor/192-F	http://vital-iot.eu/ontology/ns/VitalSensor	http://vital-iot.eu/ontology/ns/Unavailable	More
http://104.131.128.70:8080/istanbul-traffic/sensor/193-B	http://vital-iot.eu/ontology/ns/VitalSensor	http://vital-iot.eu/ontology/ns/Unavailable	More
http://104.131.128.70:8080/istanbul-traffic/sensor/197-F	http://vital-iot.eu/ontology/ns/VitalSensor	http://vital-iot.eu/ontology/ns/Unavailable	More
http://104.131.128.70:8080/istanbul-traffic/sensor/200-F	http://vital-iot.eu/ontology/ns/VitalSensor	http://vital-iot.eu/ontology/ns/Unavailable	More
http://104.131.128.70:8080/istanbul-traffic/sensor/201-B	http://vital-iot.eu/ontology/ns/VitalSensor	http://vital-iot.eu/ontology/ns/Unavailable	More
http://104.131.128.70:8080/istanbul-traffic/sensor/205-F	http://vital-iot.eu/ontology/ns/VitalSensor	http://vital-iot.eu/ontology/ns/Unavailable	More
http://104.131.128.70:8080/istanbul-traffic/sensor/212-F	http://vital-iot.eu/ontology/ns/VitalSensor	http://vital-iot.eu/ontology/ns/Unavailable	More
http://104.131.128.70:8080/istanbul-traffic/sensor/217-F	http://vital-iot.eu/ontology/ns/VitalSensor	http://vital-iot.eu/ontology/ns/Unavailable	More
http://104.131.128.70:8080/istanbul-traffic/sensor/224-F	http://vital-iot.eu/ontology/ns/VitalSensor	http://vital-iot.eu/ontology/ns/Unavailable	More
http://104.131.128.70:8080/istanbul-traffic/sensor/229-F	http://vital-iot.eu/ontology/ns/VitalSensor	http://vital-iot.eu/ontology/ns/Unavailable	More
http://104.131.128.70:8080/istanbul-traffic/sensor/231-F	http://vital-iot.eu/ontology/ns/VitalSensor	http://vital-iot.eu/ontology/ns/Unavailable	More
http://104.131.128.70:8080/istanbul-traffic/sensor/231-F	http://vital-iot.eu/ontology/ns/VitalSensor	http://vital-iot.eu/ontology/ns/Unavailable	More

Figure 48 List of all sensors of the infrastructure.

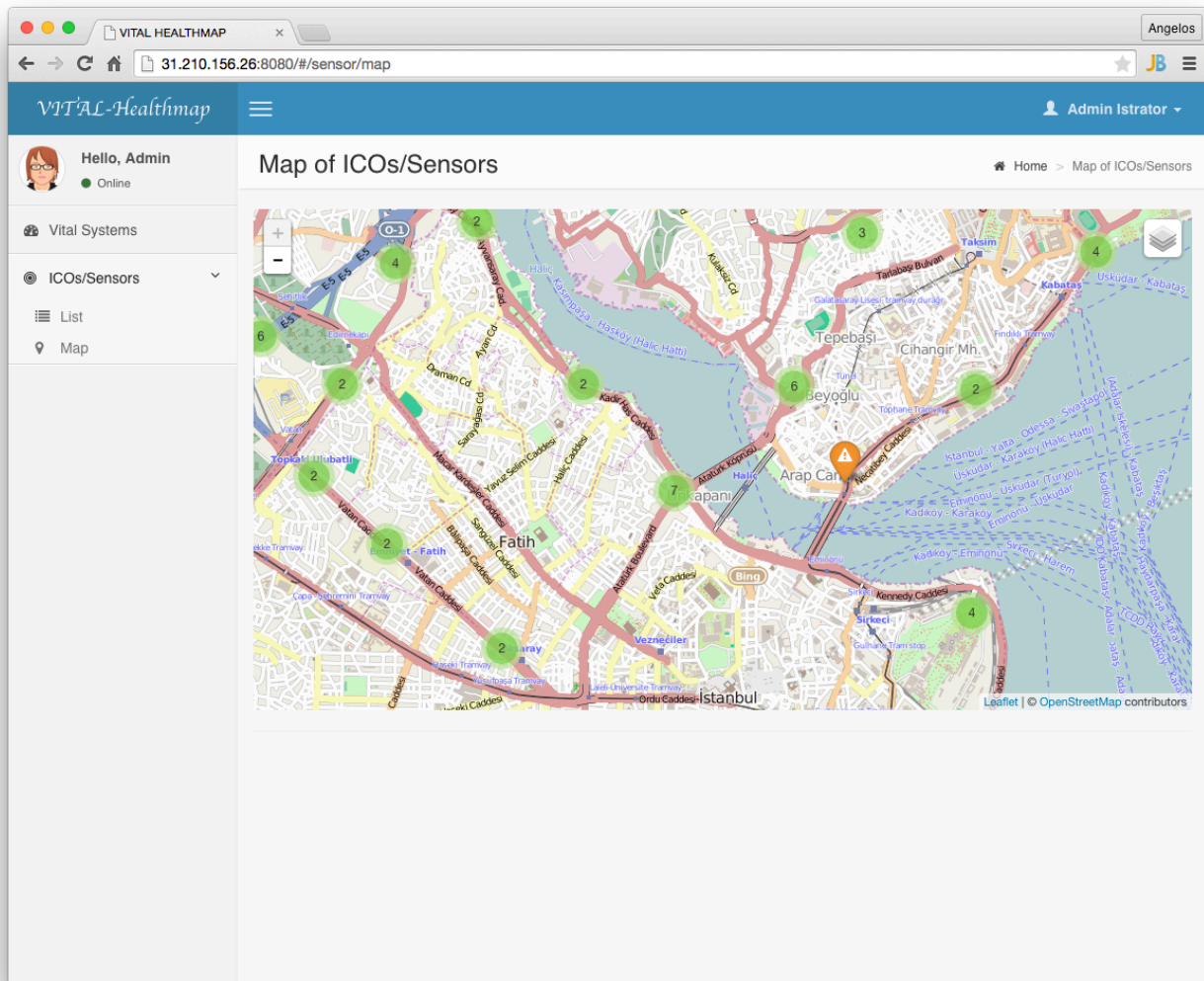


Figure 49 List of sensors in a map, for sensors with location data available.

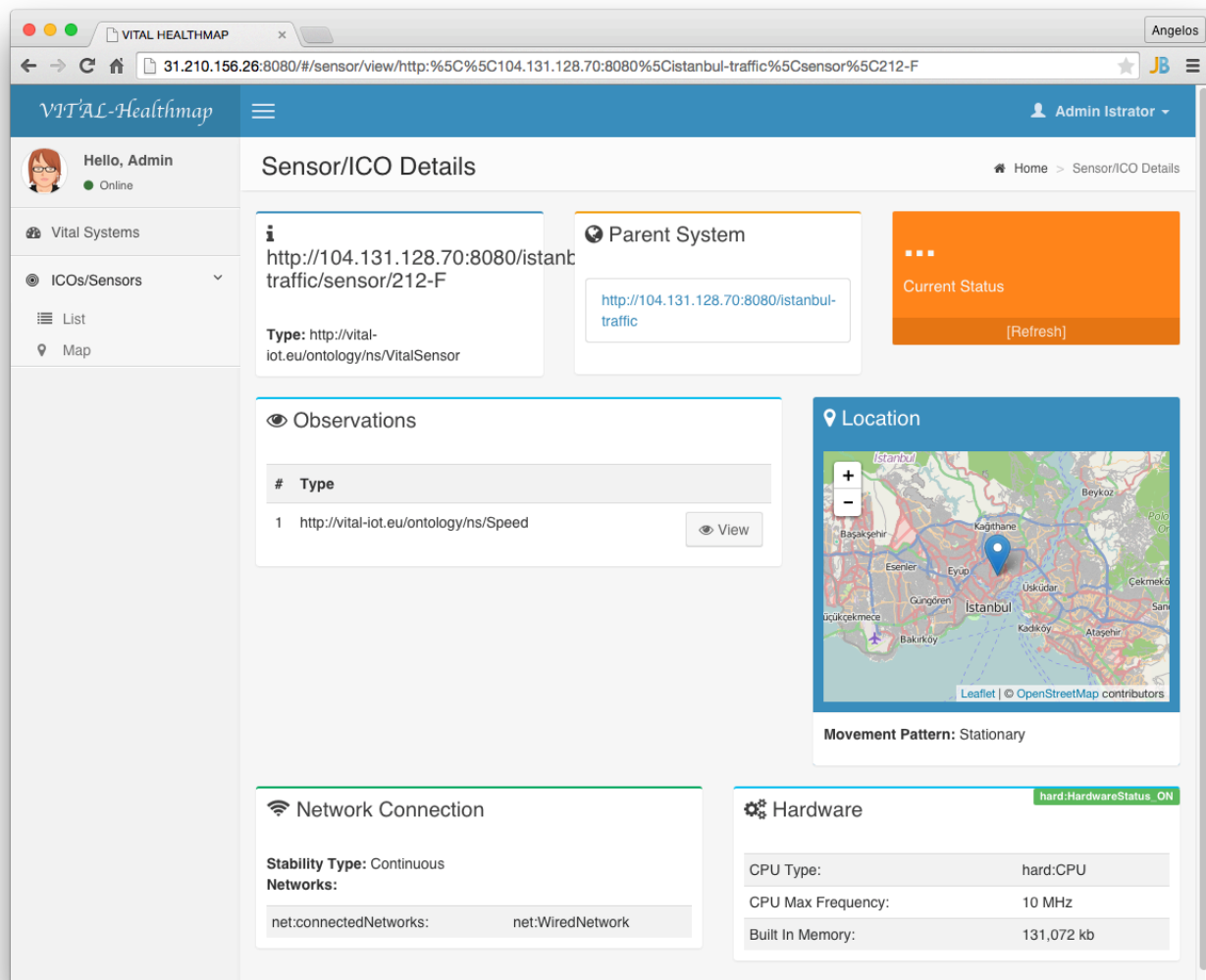


Figure 50 Overview of a single sensor.

Overview of a single sensor, with information on the system it belongs, its location, the types of observations it performs.

6.9 Security

The VITAL Security Module introduced in section 2.10 exposes a set of REST endpoints to perform all the needed operations:

- Users, groups and policies management
- Authentication
- Permissions evaluations

Here we will give a few examples of interaction with the endpoints, for a full description and documentation for developers please refer to [D5.1.2] and the "README.md" file included in the project source code.

The first example is about logging in a user and obtaining the SSO token (returned in a cookie).

Request URL:

<https://vital-integration.atosresearch.eu:8443/securitywrapper/rest/authenticate>

Request method:

POST

POST data:

name=jsmith

password=userpassword

testCookie=false

Response headers:

Access-Control-Allow-Methods: GET, POST, DELETE, PUT

Connection: Keep-Alive

Content-Length: 154

Content-Type: application/json

Date: Fri, 09 Oct 2015 14:19:14 GMT

Keep-Alive: timeout=5, max=98

Server: WildFly/9

Set-Cookie:

vitalAccessToken=AQIC5wM2LY4SfcyDC9K56N_L0z8sRPQjdjsvddKWHgZM0tY.*AAJTSQACMDEAAINLABQtMjl2NjAwNjg5MTQ0MjU0MDIyNw..*;

Domain=.atosresearch.eu; Path=/; secure; HttpOnly

X-Powered-By: Undertow/1

access-control-allow-credentials: true

access-control-allow-origin: *

Response body:

```
{
  "uid": "jsmith",
  "name": "John",
  "fullname": "John Smith",
  "creation": {
    "year": "2015",
    "month": "August",
    "day": "27"
  },
  "mailhash": "a3b100e8ba6b2bd3e8b29899a262f2d9"
}
```

Figure 51 Log in a user and obtain the SSO token

The second example shows the request for a policy evaluation of the just logged-in user; the response describes the permissions the user has to perform specific actions over the resource.

Request URL:

<https://vital-integration.atosresearch.eu:8443/securitywrapper/rest/evaluate>

Request method:

POST

POST data:

resources[]=https://vitalsp.cloud.reply.eu/resA

Request headers:

Host: vital-integration.atosresearch.eu

User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0) Gecko/20100101 Firefox/41.0

Accept: application/json, text/plain, */*

Accept-Language: en-US,en;q=0.7,it;q=0.3

Accept-Encoding: gzip, deflate

DNT: 1

Content-Type: application/x-www-form-urlencoded; charset=UTF-8

Referer: https://vitalgateway.cloud.reply.eu/secman/

Content-Length: 33

Cookie:

vitalAccessToken=AQIC5wM2LY4SfczRNQjJBzYer_oaQrQQrdvCYnL4v48dF6w.*AAJTSQACMDEAAINLABQtNzYwNzcXMTEyODk0OTMxMDQ4Mw..*;

vitalTestToken=AQIC5wM2LY4SfcyDC9K56N_L0z8sRPQjdjsvddKWHgZM0tY.*AAJTSQACMDEAAINLABQtMjl2NjAwNjg5MTQ0MjU0MDIyNw..*

Connection: keep-alive

Pragma: no-cache

Cache-Control: no-cache

Response body:

```
{
  "responses":[
    {
      "advices":{
      },
      "resource":"https://vitalsp.cloud.reply.eu/resA",
      "actions":{
        "GET":true
      },
      "attributes":{
      }
    }
  ]
}
```


}

Figure 52 Request for a policy evaluation

The Security Management interface is integrated in the Management Platform UI and allows performing the needed configurations (for users, groups and policies) in a user-friendly way and preventing incorrect configurations. The following operations have been specified in [D5.1.2] and implemented:

- Users listing;
- Group listing;
- Policy listing;
- Add, delete, read User;
- Add, delete, read Group;
- Manage group membership:
 - Add user/s to group;
 - Delete user/s from group;
 - Check if a specific user is in a specific group.
- Delete, read Policy;
- Create policy:
 - Create Group Policy to grant access to the defined resource only to the users that are member of the defined groups.
- Monitor and test policies evaluation and data access control.

The security management user interface is described in [D5.1.2, Section 7].

A typical security configuration requires creating user accounts, assigning them to groups, which will be subject of authorization policies, and configuring policies to authorize access to specific resources or applications; policy configuration can be done by specifying a resource individually through its URL or by using patterns with the wildcards * and *- , as explained in [D5.1.2, Section 7].

Additional configurations, concerning identity management, if needed, could be done through the OpenAM console. For instance, the authorization SSO token, an encrypted reference to the session stored by Identity Provider, can be configured to have a limited validity. Currently it is configured with Max Session Time (a value in minutes to express the maximum time before the session expires and the user must re-authenticate to regain access) to 60. It can be modified if needed.

Vital-IoT Hello, Lorenzo Online

Vital Systems

ICOs/Sensors

Security

- Users
- Groups
- Policies
- Applications
- Monitor
- Access test

Policies Control Panel Policy Details

Home > Policies Control Panel

Edit details

Name	Application	Description	Active
DMS generic	iPlanetAMWebAgentService	Policy to allow basic access to the DMS	true

Add group

Affected groups	Actions
Dev_Users	Remove from policy
Internal_Users	Remove from policy
Advanced_Users	Remove from policy

http://example.com/resource/* Add resource

Resources	Actions
https://vitalsp.cloud.reply.eu:443/vital/dms*	Remove resource
https://vitalsp.cloud.reply.eu:443/vital/dms/*	Remove resource

Edit actions

Action	Specify	Allow
POST	<input checked="" type="checkbox"/>	Yes
GET	<input checked="" type="checkbox"/>	Yes
PATCH	<input type="checkbox"/>	

Figure 53 Security Management UI

7 NEXT STEPS

This section describes the next steps for each component deployed in this release of the VITAL framework.

7.1 IoT Data Adapter

The development of the IoT Data Adapter, as far as functionalities are concerned, is complete. The only changes that might be required in the future are related to performance, scalability, and security and a UI integration with the Management Platform.

7.2 Data Management Service (DMS)

Next steps for Data Management Service involve the continuation of research on side line about hybrid solution to cope the problem of dealing with different data formats of JSON-LD and RDF at the same time. Currently DMS is returning data in JSON-LD format, and RDF/XML is out of picture yet. Testing will be carried out after the successful integration with other modules. Furthermore, data storage and retrieval services will be enhanced and any issues arise in future will be dealt to offer smooth integration and operations.

7.3 Discovery service

Next steps for Discovery service are mainly based on integration with other components of the architecture, in order to be able to provide a more complete service and a bigger set of functionalities. Being DMS its direct data source, integration with it will allow to have a more complex set of options for discovery of elements belonging to the system. Integration with components using Discovery service will allow them to have a dynamic way to update their status as well as their functionalities.

7.4 Filtering Service

Next steps for Filtering service regards integration and enhancement. To this end integration with DMS will provide to all Filtering users a direct access to data with the properties that they need. Integration with Orchestrator will enhance the set of advantages that final users can obtain from VITAL as system. Further enhancements can be also obtained increasing the number of filtering options and data sources.

7.5 Orchestration Service

Following the prototype implementation, the second and final version of the Orchestrator module will be enhanced in the following aspects:

- The development and deployment of additional adapters, including adapters for interacting with the Service Discovery, CEP and Filtering modules of the VITAL architecture. The goal of these interactions is to enable the exploitation and reuse of a wider set of elementary operations (in addition to PPI operations which are exploited in the first release).
- The development of reuse templates, corresponding to reusable business logic commonly used in the context of smart city applications.
- The support for graph representations of workflow to extend the current serial/array structure.
- The consideration of uncertainty as part of VITAL workflows.

Overall, the final version of the module will offer more functionalities, along with a more complete integration with other modules of the VITAL platform in line with the VITAL architecture. This will subsequently facilitate the development of applications using the VITAL platform. [D4.4.1-2015].

7.6 Complex Event Processing (VITALCEP)

The current version of the development provides the interfaces needed to provide the functionalities related to filtering and event processing, this includes the interfaces for users as described in [D4.3.1] section 4.3 (Filtering interface) and section 4.2 (Management interface). It also includes the basic functionality of the PPI interface to request data to DMS and to send the result of the processing to DMS, as described in [D3.2.2 sections 3.2.1 to 3.2.4].

The next version of VITALCEP will include a full development of PPI and VUAI in order to provide monitoring and configuration services as described in [D3.2.2 section 3].

7.7 Development Tools

The first version of the development tools provided integration only at the PPI level: developers could select the PPI they want to use, and interact directly with it. The second and final version of the tools will provide integration at the functionality level with all VITAL components. More specifically, we intend to implement one node for each functionality that each VITAL component exposes (i.e. for each VUAI). At the moment we have done that experimentally for some of the functionalities provided by the service discovery component, and we continue with the rest.

7.8 Management Tools

The current and second release of the VITAL management platform has focused on the implementation of modules for monitoring and/or configuring the various modules of the VITAL platform, as well as their integration within the VITAL management tool. This approach will be extended as part of the third and final release, which will integrate additional modules of the VITAL architecture i.e. modules whose implementation is currently work in progress as part of WP4 and WP5 of the project. The final version of the platform will therefore become a single entry points for monitoring all the modules of a VITAL deployment. Thus, the final release will integrate these modules to the platform through:

- Leveraging the dynamic discovery capabilities provided by the Discovery Service and the data management services of the VITAL platform. These capabilities will render the VITAL management plane more intelligent and dynamic.
- The integration of the VITAL management plane with the VITAL security framework. Currently management of users and policies is provided. As part of the third version the VITAL security modules/mechanisms will be used in order to ensure that all interactions between services and users are properly authenticated and authorized

- The integration of an SLA management/monitoring module within the management plane/tool in order to facilitate management of SLAs in the smart city environment.

The above-listed functionalities are not exhaustive. The third release of the deliverable will provide an integrated prototype implementation of the monitoring of smart city environments (e.g., sensors, IoT platforms), but also of the VITAL components comprising a VITAL deployment. [D5.1.2-2015]

7.9 Security

The Security Management UI is being integrated with the latest Management Tools UI; the first phase of system integration will end soon and will see all VITAL services protected and accessible through authentication and authorization.

Fine-grained data access control, intended to allow selective access to data through services such as DMS and PPIs, is already in place and will be exploited as soon as the other modules are integrated.

Development in the security governance area is in progress as part of VITAL WP5 tasks. This will result in a governance module, building upon the Security Management module, to facilitate the deployment and configuration of security in the various smart cities scenarios.

8 CONCLUSION

This document provides a report about the works done for the integration of the VITAL framework and the development of each functional component. In this second iteration of the integration, due to the short time since the previous version, there is no significant progress in modules development. The work has been focused on unifying the installation process of VITAL platform, and adapting this document to the comments received in the first review report.

Vital 2016