



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

D5.1.2 Management Services Over Federated IoT Platforms V2

Document Id:	VITAL-D512-250815-Draft
File Name:	VITAL-D512-250815-Draft.doc
Document reference:	Deliverable 5.1.2
Version:	Draft
Editor:	Fotis Stamatelopoulos, Angelos Lenis, Stelios Pantelopoulos, Anne Helmreich
Organisation:	SiLo
Date:	25 / 08 / 2015
Document type:	Deliverable
Security:	PU (Public)

Copyright © 2015 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
V20	Angelos Lenis, Fotis Stamatelopoulos	SiLo	04/06/2015	Updated Document Structure based on D5.1.1
	Angelos Lenis	SiLo	30/06/2015	Updated Chapter 3, with the new format of data and exposed services
V21	Lorenzo Bracco, Paola Dal Zovo	Reply	29/06/2015	Sections related to security management (5.7 and 4.1.5)
V22	A. Lenis, Fotis Stamatelopoulos	SiLo	01/07/2015	Updated sections 5.1 to 5.5
V23	Salvatore Guzzo Bonifacio	Inria	03/07/2015	Updated sections regarding Discovery Service
V24	Lorenzo Bracco, Paola Dal Zovo	Reply	05/08/2015	Update the Security Adapter interface (5.7.2.1)
V25	Miguel Angel Mateo	Atos	22/08/2015	Update sections 3.2.4, 3.2.5, 5.6 and 5.8
V26	Angelos Lenis	SiLo	24/08/2015	Updated sections 5.2, formatted document
V27	Miguel A. Mateo, Paola Dal Zovo	Atos, Reply	25/08/2015	Updated sections 4.1.4 and 5.2
V28	Fotis Stamatelopoulos	SiLo	25/08/2015	Updated Section 6 and prepared document for QA
V29	Martin Serrano	NUIG	23/08/2015	Circulated for Approval
Draft	Martin Serrano	NUIG	25/08/2015	EC Submitted

CHANGES & UPDATES FROM D5.1.1

New or Updated Section	Changes & Comments
Section 1 – Introduction	Updated in order to reflect enhancements over D5.1.1; A description of the updated functionalities is included
Section 5 – Prototype Implementation (V2)	Enhancements reflecting the full range of updated functionalities implemented as part of the second release
Sections 3.1, 3.2	Updated APIs and data structure based on changes of interacting components
Section 2.3	Updated Vital architecture figure
Section 4.1.5 and 5.7	Described specification and implementation of security management.
Sections 5.1, 5.2	Described additional features, as evolved from first prototype.
Sections 5.3, 5.4, 5.5, 5.6, 5.7, 5.8	Updated sections to describe in detail how topology, monitoring and configuration are implemented and displayed
Sections 5.6, 5.8	Updated sections with new feature for Event and SLA management
Section 2.3	Updated VITAL Architecture figure

TABLE OF CONTENTS

DOCUMENT HISTORY	1
CHANGES & UPDATES FROM D5.1.1.....	1
 1 INTRODUCTION	 6
1.1 SCOPE.....	6
1.2 AUDIENCE.....	7
1.3 SUMMARY AND STRUCTURE.....	7
 2 INCEPTION AND SCOPING OF VITAL MANAGEMENT LAYER	 8
2.1 MANAGEMENT LAYER FOR SMART CITIES.....	8
2.2 TYPES OF VITAL MANAGED OBJECTS.....	10
2.3 POSITIONING IN THE VITAL ARCHITECTURE	11
2.4 DESIGN AND IMPLEMENTATION APPROACH.....	12
 SPECIFICATION OF MANAGED ENTITIES	 13
3.....	13
3.1 GENERIC MANAGEMENT MEASUREMENT	13
3.1.1 Monitoring Hooks	17
3.1.2 Configuration hooks	21
3.1.3 ICO management.....	22
3.2 MANAGED ENTITIES.....	22
3.2.1 IoT Systems	23
3.2.2 Services	25
3.2.3 Sensors / ICOs.....	25
3.2.4 VITAL Platform Modules	26
3.2.4.1 Service Discover	26
3.2.4.2 VITAL Data Management Services	27
3.2.4.3 CEP	27
3.2.4.4 VITAL Orchestrator module	28
3.2.5 Service Level Agreements	29
3.2.5.1 SLA Parameters	29
3.2.6 City Specific Information	30
 4 FUNCTIONAL AND TECHNICAL SPECIFICATIONS.....	 31
4.1 MANAGEMENT FUNCTIONALITY SUPPORTED.....	31
4.1.1 Topology	31
4.1.2 Monitoring	31
4.1.3 Alerts / Events.....	32
4.1.4 Configuration.....	33
4.1.5 Security	33
4.1.6 Accounting	33
4.2 TECHNICAL ARCHITECTURE AND REQUIREMENTS	33

5	PROTOTYPE IMPLEMENTATION (V2)	36
5.1	SUPPORTED FUNCTIONALITY	36
5.2	MIGRATION PLAN: FROM PROTOTYPE TO CONCRETE IMPLEMENTATION	36
	DYNAMIC TOPOLOGY DISCOVERY	38
5.3		38
5.4	PERFORMANCE MONITORING	40
5.5	CONFIGURATION MANAGEMENT	43
5.6	ALERTING AND EVENT MANAGEMENT SERVICES	43
5.7	SECURITY MANAGEMENT FUNCTIONALITIES	44
5.7.1	Security management user interface	46
5.7.2	Security management REST interfaces	50
5.7.2.1	Security management REST API	50
5.8	SLA MONITORING & MANAGEMENT FUNCTIONALITIES	57
6	OUTLOOK AND CONCLUSIONS	59
7	REFERENCES	61

LIST OF FIGURES

FIGURE 1: VITAL ARCHITECTURE	11
FIGURE 2: MANAGEMENT LAYER TECHNICAL ARCHITECTURE	34
FIGURE 3 MANAGEMENT UI: LIST OF SYSTEMS	39
FIGURE 4: MANAGEMENT UI: LIST OF SENSORS	40
FIGURE 5: MANAGEMENT UI: SYSTEM OVERVIEW – INFO AND STATUS	41
FIGURE 6: MANAGEMENT UI: SYSTEM OVERVIEW - PERFORMANCE GRAPHS	41
FIGURE 7: MANAGEMENT UI: GEOGRAPHICAL HEALTH-MAP	42
FIGURE 8: MANAGEMENT UI: GEORGRAPHICAL HEALTH-MAP - SENSOR DETAILS	42
FIGURE 9: MANAGEMENT UI: CONFIGURATION	43
FIGURE 10 – SECURITY MANAGEMENT ARCHITECTURE	49
FIGURE 11 – MANAGEMENT MENU INCLUDING SECURITY SECTION	46
FIGURE 12 – USERS LISTING	47
FIGURE 13 – USER CREATION	47
FIGURE 14 – USER EDITING	48
FIGURE 15 – GROUP CONTROL	48
FIGURE 16 – POLICY CREATION	49
FIGURE 17 – MONITOR INTERFACE	50

LIST OF TABLES

TABLE 1: JSON-LD SAMPLE FOR THE MANAGEMENT DATA.	13
TABLE 2: IoT SYSTEM SERVICES MANAGEMENT SECTION.	15
TABLE 3: GETSYSTEMSTATUS RESPONSE DATA.	17
TABLE 4: GETSENSORSTATUS RESPONSE DATA.	18
TABLE 5: GETSUPPORTEDPERFORMANCEMETRICS RESPONSE DATA.	19
TABLE 6: GETPERFORMANCEMETRICS RESPONSE DATA.	19
TABLE 7: GETCONFIGURATIONOPTIONS RESPONSE DATA.	21
TABLE 8: UPDATECONFIGURATIONOPTIONS RESPONSE DATA.	22
TABLE 9: GETUSERINFO	51
TABLE 10: GETGROUPINFO	51
TABLE 11: GETPOLICYINFO	51
TABLE 12: GETUSERS	51
TABLE 13: GETGROUPS	52
TABLE 14: GETPOLICIES	52
TABLE 15: GETUSERGROUPS	52
TABLE 16: GETSTATS	52
TABLE 17: CREATEUSER	53
TABLE 18: DELETEUSER	53
TABLE 19: CREATEGROUP	53
TABLE 20: DELETEGROUP	54
TABLE 21: ADDUSERTOGROUP	54
TABLE 22: DELETEUSERFROMGROUP	54
TABLE 23: DELETEPOLICY	54
TABLE 24: CREATEGROUPIDENTITYPOLICY	55
TABLE 25: UPDATEUSER	55
TABLE 26: UPDATEPOLICY	56
TABLE 27: GET SLA PARAMETERS	58
TABLE 28: ADD/UPDATE SLA PARAMENTER	58
TABLE 29: REMOVE SLA PARAMETER FROM A SOURCE	59

TERMS AND ACRONYMS

FCAPS	Fault, Configuration, Accounting, Performance, Security
FIT	Future Internet Technology
GSN	Global Sensor Networks
GUI	Graphical User Interface
ICO	Internet-Connected-Object
IoT	Internet-of-Things
KPI	Key Performance Indicator
QoS	Quality of Service
SLA	Service Level Agreements
ICT	Internet and Communication Technologies
CEP	Complex Event Processing
VUAI	Virtualized Unified Access Interfaces
API	Application Programming Interface
PPI	Platform Provider Interface
URI	Uniform Resource Identifier
DMS	Data Management Services
BPM	Business Process Model
UI	User Interface
UX/UI	User Experience/User Interface

1 INTRODUCTION

1.1 Scope

Modern cities are increasingly seeking ICT-driven ways to plan and monitor their sustainable interventions on the urban environment. Such interventions include the development, deployment and operation of IoT-based smart city services. The ability to plan, monitor and manage such IoT based services regardless of the technology platform that empowers them is a key enabler for structuring and organizing city-wide interventions. The VITAL project focuses on the development of a novel smart cities platform, which will enable the integration and semantic interoperability of multiple IoT systems that underpin smart city applications and services. One of the core characteristics of this platform (as specified in WP2 and deliverable D2.3) is its management layer, which aims at providing functionalities for managing in a unified way diverse IoT systems and services, thereby facilitating the city-wide planning and monitoring of relevant interventions. Despite the importance of such a management layer, most of the IoT platforms (including IoT platforms deployed in VITAL such as GSN, Hi and FIT) do not provide relevant functionalities and therefore VITAL aspires to be a pioneer in this respect.

The present deliverable introduces the VITAL management layer, including the drivers behind its development and its main functionalities. In particular, the deliverable presents the VITAL management layer and its role within the VITAL architecture. It also illustrates the various entities and objects that will be managed (i.e. VITAL managed objects), along with the main management functionalities to be implemented over them. Furthermore, the deliverable illustrates the prototype implementation of the management layer, which follows the early (mock-up) release presented as part of deliverable D5.1.1. The prototype implementation includes a management application for monitoring and visualizing VITAL managed objects. The second version of the application includes several updates and enhancement when compared to the initial release including:

- Functionalities for Dynamic Topology Discovery, including dynamic discovery of services within the management platform.
- Functionalities for monitoring and visualizing performance.
- Configuration management services, notably services for configuring the PPI (Platform Provider Interfaces) services.
- Functionalities for managing alerts and event management services, including CEP (Complex Event Processing) services.
- Services for managing SLAs and security, including roles and credentials.

The VITAL management platform will be further enhanced with additional management functionalities and services. Additional enhancement will be included and described in the scope of the third and final release of the present deliverable.

1.2 Audience

This deliverable is destined for researchers and engineers that are interested in the development of management applications for smart city services and more specifically on IoT based applications and services. It is also expected to be of great interest to development communities working on the OpenIoT, GSN and FIT platforms, given that it provides valuable add-on management functionalities, which are currently missing from these platforms.

The deliverable could be also of interest to operators/administrators of smart city services (including the city authorities), which are the natural end-users for the management services provided by the VITAL management layer in a smart cities context. The VITAL management platform is an independently exploitable result/solution of the project, since it can enable smart city authorities to monitor, configure and manage their infrastructures, including SLA, security and performance aspects.

Note also that the deliverable has also importance as an internal milestone document for the VITAL consortium, since it is closely linked to VITAL results in other (on-going and future) project activities, such as the activities relating to the VITAL applications integration (WP6), but also to activities associated with data/interfaces modelling and implementation (in WP3).

1.3 Summary and Structure

The second version of this deliverable starts with a discussion of the motivation and need behind the implementation of a management tool (and of an associated environment) as part of a smart cities platform. It also illustrates its positioning within the VITAL architecture, including the way it interacts with other modules of the VITAL platform. A significant part of the deliverable is devoted to the specification of the entities/objects to be managed, but also of the type of management operations (e.g., read/write) that are envisaged over them. As part of the deliverable, we also outline the early prototype implementation of the VITAL management environment, including an illustration of supported/implemented functionalities and some relevant screenshots. The deliverable is structured as follows:

- Section 2 following this introductory section outlines the drivers behind the specification and implementation of the VITAL management environment. It also outlines the positioning of the management environment within the overall VITAL architecture, along with the specified integration interfaces. Finally, this section explains the approaches selected for designing and implementing the management layer within the VITAL project.
- Section 3 provides a detailed list of the entities to be managed (and of their attributes). These entities include IoT systems, ICOs, SLAs, security components, city-wide information and more.
- Section 4 presents the set of management functionalities that are supported & implemented by the VITAL management environment as part of the second release of the deliverable D5.1. Moreover, this section provides an overview of the technical architecture and design guidelines of the management layer.

- Section 5 presents the prototype implementation of the VITAL management environment, which accompanies this document as part of the second release of the deliverable. The description of this prototype implementation is an extended version of the relevant functionalities description of the earlier release of the deliverable (i.e. D5.1.1). Details about new functionalities (e.g., dynamic topology discovery, SLA management, security management and more) are provided.
- Section 6 is the concluding section of the deliverable. Apart from concluding remarks, it also provides an outlook for the extensions that will be implemented in the final release of the deliverable.

2 INCEPTION AND SCOPING OF VITAL MANAGEMENT LAYER

2.1 Management Layer for Smart Cities

An IoT Platform is a distributed software system with multiple components and subsystems that typically integrate with external services and sensor networks. In addition to the application software systems involved, an IoT deployment requires infrastructure software (e.g. application servers, databases) and hardware (servers, network equipment) as well as peripheral protocols, APIs, and software systems that provide sensor data streams and related business services. This architectural diversity and operational complexity is the main drive behind the introduction of a management layer that will enable the efficient monitoring of the platform & supported services status and operational readiness.

The need of a management plane/layer is common to all complex ICT systems, especially when the services offered are of a critical nature; however, IoT systems and their applications in the Smart City domain are not just complex systems that support critical functionality but they also provide processed data and reports that may affect large scale urban planning, policy making, regional/national-level strategic decisions or even legislation. Consequently, a Smart City IoT system requires a management layer that:

- enables the monitoring of operational readiness of the system as a whole and the supported services,
- provides early warning on faults and glitches that affect the quality of service offered to the Smart City actors, before the problem escalates into the complete disruption of service or even worse into producing false data,
- provides the FCAPS model functionality [Raman98] of the typical management system, focusing mostly on the IoT services provided and less on the low level systems information (but also be able to drill down into some more detailed management views depending on the specific scope),
- provides friendly and ergonomic user interfaces to the operator and administrators of the Smart City IoT system as well as e-services that can be used by higher level IT Governance Systems used by the city or the regional government,

- and seamlessly integrate with underlying ICT systems that cooperate with the IoT platform in order to minimize delays and communication glitches, thus achieving near real-time information.

The above are high level requirements derived by the key goals of the Smart City IoT business goals and by taking into account the best practices and state-of-the-art functional characteristics of the modern management systems (systems and network management, application management, and integrated services management). Although standards and available technologies would allow the management of an IoT system to drill down to excruciating detail throughout the technology stack (e.g. network, system, operating system, DB/app server, application), the management layer should focus on enabling the timely monitoring, and in some cases control, of the health of the overall platform as well as the quality of service delivered to the Smart City actors. The following is a list of functionalities and functional modules that we consider important and critical in this scope:

- Sub-system / component and overall health monitoring: provide a “health map” of the IoT platform with the ability to drill-down to a level that is practical for monitoring sub-systems and component operational status parameters.
- Operations management: provide the GUI tools for supporting operations management, i.e. basic monitoring and configuration of operational parameters, visual status reporting for individual components (in list or geographical views).
- Data/Information strategic planning input: maintain and process historical data in order to provide a data store of structured monitoring data of key operational information, QoS parameters, and service KPIs that can be retrieved via querying/export functionalities and to be used in strategic planning and similar activities by the city/regional authorities.
- Critical FCAPS operations: cover the complete span of the FCAPS functional model, but focus on Smart City & IoT/VITAL platform specific requirements:
 - Fault: health map, visual representation of fault events with drill down options, alerting
 - Configuration: geographical map, configuration parameter monitoring and control for platform components and underlying systems (depending on permissions)
 - Accounting: monitor and maintain accounting information, i.e. client application usage of platform resources, for enabling the implementation of fairness control policies, etc. SLA monitoring / management is an important aspect that is associated with this area.
 - Performance: monitor the performance of the platform and individual sub-systems. The goal is not only to monitor QoS but also to provide “early warning insight” on developing faulty conditions before they escalate and/or cascade to an actual fault.
 - Security: monitor operational parameters related to the security and access control of the platform.

The VITAL platform is a complex IoT system that consists of multiple loosely coupled subcomponents (via RESTful APIs), and it by design integrates with multiple and heterogeneous underlying IoT Platforms / Products. VITAL provides to its connected applications a common view of the data & services provided by these underlying systems, unified by a common semantic data model. In a similar fashion the VITAL management layer provides a common management plane that unifies all the management information of all components and systems as well as management functionality that can be performed in a unified way to all parts of the VITAL ecosystem.

More specifically, the VITAL management layer:

- Supports the management of the core VITAL platform and all individual VITAL sub-systems (i.e. software components that are designed and developed by the VITAL project).
- Provides a common, unified management view (monitoring and control if allowed) of the different underlying IoT Systems. It will integrate with any management hooks provided by these systems and translate data and functionality into the common VITAL management model.
- Manages the relationships between VITAL components and subsystems. This is an extension of the monitoring (and control when possible) of the individual component/sub-system operational / health parameters to the monitoring of the interactions between them.

2.2 Types of VITAL Managed Objects

The VITAL management layer will handle the following types of objects (VITAL managed objects)

- ICOs and data streams: These are the actual “low-level” ICOs that are typically connected to the VITAL platform via an underlying IoT System. The management layer provides operational information of the ICO basic parameters (including position and trajectory for mobile ICOs) as well as the availability and QoS (if available by the IoT systems) parameters of the related data streams. This is information provided either by the IoT Systems’ management hooks or from the related VITAL sub-systems/components.
- IoT Systems: These are the underlying IoT Platforms and Applications that are connected to the VITAL platform via the VITAL adapters. Depending on what each system supports, the management layer retrieves, processes and uses information on the health, operational parameters, performance, accounting, security and even specific alerts emanating from the connected IoT system.
- VITAL Platform Components: These are the individual loosely coupled sub-systems/components of the VITAL architecture, i.e. the CEP module, the Service Discovery module, the Data Management layer and the VUAls, the Workflow Management module, etc. The management layer monitors and handles health information as well as data related to all the aspects of the FCAPS model (wherever applicable) for each component.

- **SLA:** This refers to the SLAs set between the VITAL deployment and the connected Platform/Data Providers. The management layer will provide monitoring services as well as historical data on these SLAs.
- **Security information:** This refers to all security related management data, i.e. data related to authenticated sessions and failed authentication or unauthorized access attempts by the applications operating on top of the VITAL platform. It could also extend to other security related information like authentication errors in the communication of VITAL with the underlying IoT Systems (if they support it).
- **City-specific configurations:** Access to this information is realized via the VITAL Governance toolkit/module (T5.4). Although the T5.4 will produce specific software modules, the management layer will integrate T5.1 and T5.4 functionality via a common GUI.

2.3 Positioning in the VITAL Architecture

The following diagram (Figure 1) provides an overview of the VITAL architecture and outlines the relationships and basic communication between the various components.

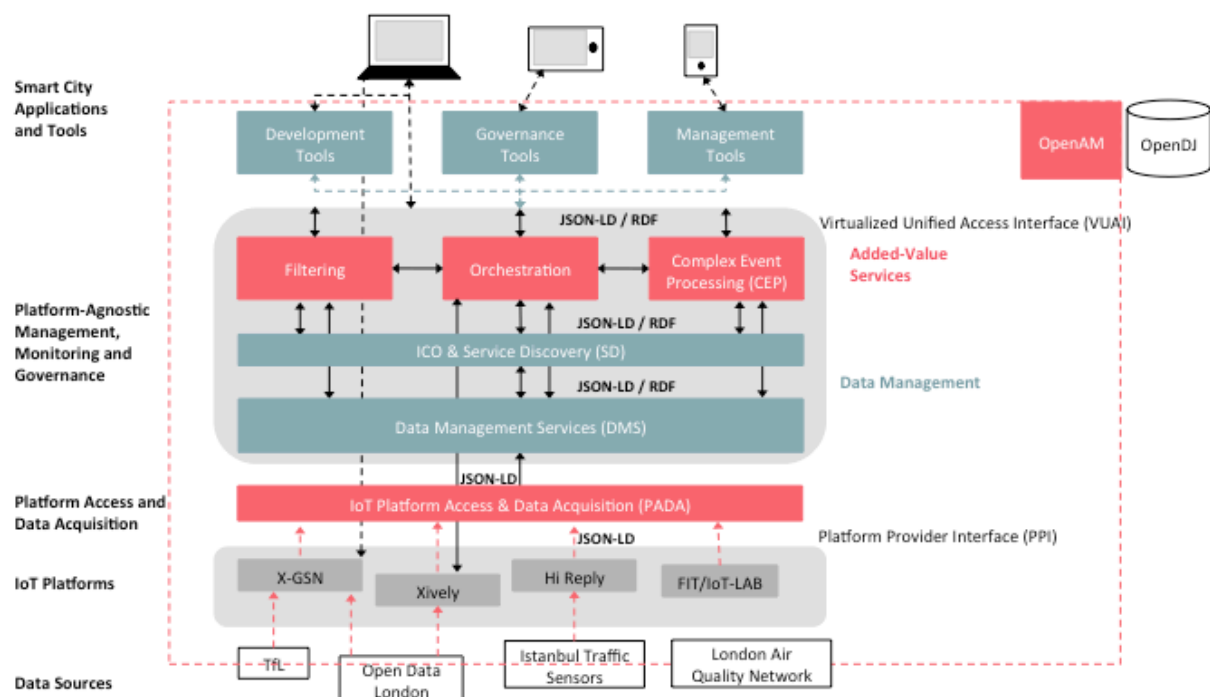


Figure 1: VITAL Architecture

The Management layer (shown as Management Tools), as well as the Governance set of tools, is a high level component in the sense that it interfaces with all other subsystems and components of the VITAL platform architecture. As shown in the diagram, the Management layer interface:

- Uses the VUAls for monitoring & retrieving information on the managed entities (or for pushing configuration changes), with a main focus on VITAL components and VITAL higher level service components. The VUAls can be also used for retrieving information on ICOs/datasources and underlying IoT Systems.
- Uses directly the PPIs via the VITAL Adapters as an alternative for accessing ICO/IoT System management functionality in use cases where performance is critical (i.e. avoid the delay of going through all the VITAL layers for a configuration change).

In both cases, the Management Layer accesses the RESTful interfaces and the VITAL ontologies and data models, conforming to the JSON-LD syntax defined by VITAL.

In the current version of the design, the Management Layer does not publish services for consumption by the applications built on top of the VITAL platform. It does provide:

- A GUI application for the platform operator / administrator.
- Possibly¹ a set of eServices that can be used by the platform administrators for integration with Network & System Management Systems.

The following VITAL components interact with the Management Layer as follows:

- Discoverer: provides service directory information (e.g. for the healthmap as well as meta-data for accessing VITAL services) as well as management hooks for monitoring its health.
- Data Services: provides access to the semantically unified structured data and underlying services.
- Adapters/PPIs: provide management hooks to underlying IoT systems (and their services) and ICOs/data streams.
- All VITAL components: provide management hooks for monitoring and configuration (if allowed by each component).

2.4 Design and Implementation Approach

An incremental top-down approach will be adopted for the implementation of the VITAL management environment. This approach is based on the segmentation of this environment to a number of middleware modules and dashboards, which are dedicated to the management of specific smart city entities (such as ICOs, smart cities platforms, smart city applications and more). Following this segmentation, the management environment will gradually incorporate middleware modules that will enable the management of different types of managed entities sets. Also, the management environment will offer a range of visualization modules (such as mashups), which will be conveniently used in order to present/visualize the values that are associated with managed entities, as well as their evolution (e.g., in time or

¹ It will be investigated before the final version of this deliverable (D.5.1.3) whether this is a useful feature with real-world applications.

space). The overall approach is incremental since more bundles of managed entities will be gradually incorporated to the VITAL management environment. This incremental approach will be also reflected in the successive releases of the present deliverable. Indeed, the present release illustrates the first (mock-up) prototype implementation of the management environment, while the latter two releases of the deliverable will elaborate on add-on functionalities associated with the management of additional sets of managed entities.

In addition to incremental, the approach taken towards implementing the prototypes accompanying this deliverable is a top-down approach. It starts with the establishment of a general environment for managing IoT resources, data, platforms and applications for smart cities, which will be gradually refined, detailed and customized to the needs of specific sets/bundles of managed entities. The general environment includes also (re)useable visualization elements for several (rather than a few) middleware modules of the VITAL platform. The following section provides an initial specification of the entities to be managed.

3 SPECIFICATION OF MANAGED ENTITIES

3.1 Generic Management Measurement

Data format for reading management data, e.g. performance throughput, error rate, component load. This is information retrieved by the management layer from the VITAL management instrumentation (modules, IoT subsystems). These are modelled as sensor observations, following the same `@context` with but with their own types of properties (vital:errors for example).

Table 1: JSON-LD Sample for the management data.

<pre>{ "@context": { "@vocab": "http://vital-iot.eu/ontology/ns/", "vital": "http://vital-iot.eu/ontology/ns/", "vital": "http://vital-iot.com/ontology#", "lsm": "http://lsm.derii.ie/ont/lsm.owl#", "ssn": "http://purl.oclc.org/NET/ssnx/ssn#", "geo": "http://www.w3.org/2003/01/geo/wgs84_pos#", "time": "http://www.w3.org/2006/time#", "qudt": "http://qudt.org/vocab/unit#", "xsd": "http://www.w3.org/2001/XMLSchema#", "id": "@id", "type": "@type", "time:inXSDDateTime": { "@type": "xsd:dateTime" } },</pre>

```
      "qudt:unit": {
        "@type": "@id"
      }
    },
    "uri": "http://example.iot.system/sensor/monitoring/observation/errors/95",
    "type": "ssn:Observation",
    "ssn:observedBy": "http://example.iot.system/sensor/monitoring",
    "ssn:observationProperty": {
      "type": "vital:errors"
    },
    "ssn:observationResultTime": {
      "time:inXSDDatetime": "2015-06-14T19:37:22Z"
    },
    "dul:hasLocation": {
      "type": "geo:Point",
      "geo:lat": "55.701",
      "geo:long": "12.552",
      "geo:alt": "4.33"
    },
    "ssn:observationQuality": {
      "ssn:hasMeasurementProperty": {
        "type": "Reliability",
        "hasValue": "HighReliability"
      }
    },
    "ssn:observationResult": {
      "type": "ssn:SensorOutput",
      "ssn:hasValue": {
        "type": "ssn:ObservationValue",
        "value": 1.0,
        "qudt:unit": "qudt:Number"
      }
    }
  }
}
```

The Management and Governance Layer require the addition of a new section to the IoT ontology, which is specific to management. This section provides information on the VITAL management instrumentation (service for monitoring and configuration) supported by the specific IoT System connected to the VITAL platform deployment. The structure is in Table2

Table 2: IoT System services management section.

```
[{
  "@context": {
    "@vocab": "http://vital-iot.eu/ontology/ns/",
    "vital": "http://vital-iot.eu/ontology/ns/",
    "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
    "msm": "http://iserve.kmi.open.ac.uk/ns/msm#",
    "hrest": "http://www.wsmo.org/ns/hrests#",
    "id": "@id",
    "type": "@type",
    "name": "rdfs:label",
    "description": "rdfs:comment",
    "hrest:hasAddress": {
      "@type": "hrest:URITemplate"
    },
    "operations": {
      "@id": "msm:hasOperation",
      "@type": "@id"
    }
  },
  "id": "http://example.iot.system/service/2",
  "type": "vital:MonitoringService",
  "msm:hasOperation": [{
    "type": "vital:GetSystemStatus",
    "hrest:hasAddress": "http://example.iot.system/system/status",
    "hrest:hasMethod": "hrest:POST"
  }, {
    "type": "vital:GetSensorStatus",
    "hrest:hasAddress": "http://example.iot.system/sensor/status",
    "hrest:hasMethod": "hrest:POST"
  }, {
    "type": "vital:GetSupportedPerformanceMetrics",
    "hrest:hasAddress": "http://example.iot.system/system/performance",
    "hrest:hasMethod": "hrest:GET"
  }, {
    "type": "vital:GetPerformanceMetrics",
    "hrest:hasAddress": "http://example.iot.system/system/performance",
    "hrest:hasMethod": "hrest:POST"
  }, {
    "type": "vital:GetSupportedSLAParameters",
    "hrest:hasAddress": "http://example.iot.system/system/sla",
    "hrest:hasMethod": "hrest:GET"
  }
}]
```

```
    }, {
      "type": "vital:GetSLAParameters",
      "hrest:hasAddress": "http://example.iot.system/system/sla",
      "hrest:hasMethod": "hrest:POST"
    }
  ]
}, {
  "@context": {
    "@vocab": "http://vital-iot.eu/ontology/ns/",
    "vital": "http://vital-iot.eu/ontology/ns/",
    "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
    "msm": "http://iserve.kmi.open.ac.uk/ns/msm#",
    "hrest": "http://www.wsmo.org/ns/hrests#",
    "id": "@id",
    "type": "@type",
    "name": "rdfs:label",
    "description": "rdfs:comment",
    "hrest:hasAddress": {
      "@type": "hrest:URITemplate"
    },
    "operations": {
      "@id": "msm:hasOperation",
      "@type": "@id"
    }
  },
  "id": "http://example.iot.system/service/1",
  "type": "vital:ConfigurationService",
  "msm:hasOperation": [{
    "type": "vital:GetConfiguration",
    "hrest:hasAddress": "http://example.iot.system/service/1",
    "hrest:hasMethod": "hrest:GET"
  }, {
    "type": "vital:SetConfiguration",
    "hrest:hasAddress": "http://example.iot.system/service/1",
    "hrest:hasMethod": "hrest:POST"
  }
]
```

The management part of any Vital system (PPI, CEP, Orchestrator, ...) that supports it, is composed of two services, the vital:MonitoringService that is responsible for measuring and reporting status and performance related information and the vital:ConfigurationService that allows changing the configuration of the system.

3.1.1 Monitoring Hooks

The Management & Governance layer will call the individual URIs of the required monitoring service, as defined by the response of the GetPerformanceMetrics call to the PPI of the IoT System (refer to **Error! Reference source not found.**).

Table 3: GetSystemStatus response data.

	vital:GetSystemStatus	
Description	Retrieve the current status of the system	
Method	POST	
Request headers		
Request body	{}	
Response headers	Content-Type	application/ld+json Or application/json
Response body	Example <pre> { "@context": { "@vocab": "http://vital-iot.eu/ontology/ns/", "vital": "http://vital-iot.eu/ontology/ns/", "vital": "http://vital-iot.com/ontology#", "lsm": "http://lsm.deri.ie/ont/lsm.owl#", "ssn": "http://purl.oclc.org/NET/ssnx/ssn#", "geo": "http://www.w3.org/2003/01/geo/wgs84_pos#", "time": "http://www.w3.org/2006/time#", "qudt": "http://qudt.org/vocab/unit#", "xsd": "http://www.w3.org/2001/XMLSchema#", "id": "@id", "type": "@type", "time:inXSDDateTime": { "@type": "xsd:dateTime" }, "qudt:unit": { "@type": "@id" } }, "id": "http://example.iot.system/sensor/monitoring/observation/status/1435652134143", "type": "ssn:Observation", "ssn:observedBy": "http://example.iot.systemsensor/monitoring", "ssn:observationProperty": { "type": "vital:OperationalState" }, "ssn:observationResultTime": { "time:inXSDDateTime": "2015-06-30T08:15:34Z" }, "ssn:featureOfInterest": "http://example.iot.system", "ssn:observationResult": { "type": "ssn:SensorOutput", "ssn:hasValue": { "type": "ssn:ObservationValue", "value": "vital:Unavailable" } } } </pre>	
Notes	<ul style="list-style-type: none"> The response contains the current status of the system 	

Table 4: GetSensorStatus response data.

	vital:GetSensorStatus	
Description	Retrieve the current status of the sensor	
Method	POST	
Request headers		
Request body	Example <pre>{ "id": ["http://104.131.128.70:8080/istanbul-traffic/sensor/5-F"] }</pre>	
Response headers	Content-Type	application/ld+json OR application/json
Response body	Example <pre>{ "@context": { "@vocab": "http://vital-iot.eu/ontology/ns/", "vital": "http://vital-iot.eu/ontology/ns/", "vital": "http://vital-iot.com/ontology#", "lsm": "http://lsm.derii.ie/ont/lsm.owl#", "ssn": "http://purl.oclc.org/NET/ssnx/ssn#", "geo": "http://www.w3.org/2003/01/geo/wgs84_pos#", "time": "http://www.w3.org/2006/time#", "qudt": "http://qudt.org/vocab/unit#", "xsd": "http://www.w3.org/2001/XMLSchema#", "id": "@id", "type": "@type", "time:inXSDDateTime": { "@type": "xsd:dateTime" }, "qudt:unit": { "@type": "@id" } }, "id": "http://example.iot.system/sensor/monitoring/observation/status/1435652134143", "type": "ssn:Observation", "ssn:observedBy": "http://example.iot.systemsensor/monitoring", "ssn:observationProperty": { "type": "vital:OperationalState" }, "ssn:observationResultTime": { "time:inXSDDateTime": "2015-06-30T08:15:34Z" }, "ssn:featureOfInterest": "http://example.iot.system", "ssn:observationResult": { "type": "ssn:SensorOutput", "ssn:hasValue": { "type": "ssn:ObservationValue", "value": "vital:Unavailable" } } }</pre>	
Notes	<ul style="list-style-type: none"> The response contains the current status of a specific sensor of a system 	

Table 5: GetSupportedPerformanceMetrics response data.

	vital:GetSupportedPerformanceMetrics	
Description	VITAL pulls from an IoT system metadata about the supported management metrics.	
Method	GET	
Request headers		
Request body		
Response headers	Content-Type	application/ld+json OR application/json
Response body	Example <pre>[{ "type": "http://vital-iot.eu/ontology/ns/SysLoad", "id": "http://example.iot.system/sensor/monitoring/sysLoad" }, { "type": "http://vital-iot.eu/ontology/ns/SysUptime", "id": "http://example.iot.system/sensor/monitoring/sysUptime" }, { "type": "http://vital-iot.eu/ontology/ns/MaxRequests", "id": "http://example.iot.system/sensor/monitoring/maxRequests" }, { "type": "http://vital-iot.eu/ontology/ns/Errors", "id": "http://example.iot.system/sensor/monitoring/errors" }, { "type": "http://vital-iot.eu/ontology/ns/ServedRequests", "id": "http://example.iot.system/sensor/monitoring/servedRequests" }, { "type": "http://vital-iot.eu/ontology/ns/AvailableMem", "id": "http://example.iot.system/sensor/monitoring/availableMem" }, { "type": "http://vital-iot.eu/ontology/ns/UsedMem", "id": "http://example.iot.system/sensor/monitoring/usedMem" }, { "type": "http://vital-iot.eu/ontology/ns/PendingRequests", "id": "http://example.iot.system/sensor/monitoring/pendingRequests" }]</pre>	
Notes	<ul style="list-style-type: none"> The response contains the hooks for retrieving the monitoring parameters supported by the specific IoT system. Obviously, these parameters are optional and use case specific. The management module is designed to handle missing data (e.g. if the IoT system does not provide a maxRequests value, utilization will not be calculated). 	

Table 6: GetPerformanceMetrics response data.

	vital:GetPerformanceMetrics	
Description	VITAL pulls from an IoT system actual performance metrics	
Method	POST	
Request headers		
Request body	Example <pre>{ "system": "http://example.iot.system/istanbul-traffic", "metric": ["http://vital-iot.eu/ontology/ns/Errors", "http://vital-iot.eu/ontology/ns/ServedRequests"] }</pre>	

	}
Response headers	Content-Type application/ld+json OR application/json
Response body	<p>Example</p> <pre>[{ "@context": { "@vocab": "http://vital-iot.eu/ontology/ns/", "vital": "http://vital-iot.eu/ontology/ns/", "vital": "http://vital-iot.com/ontology#", "lsm": "http://lsm.derii.ie/ont/lsm.owl#", "ssn": "http://purl.oclc.org/NET/ssnx/ssn#", "geo": "http://www.w3.org/2003/01/geo/wgs84_pos#", "time": "http://www.w3.org/2006/time#", "qudt": "http://qudt.org/vocab/unit#", "xsd": "http://www.w3.org/2001/XMLSchema#", "id": "@id", "type": "@type", "time:inXSDDateTime": { "@type": "xsd:dateTime" }, "qudt:unit": { "@type": "@id" } }, "uri": "http://example.iot.system/sensor/monitoring/observation/errors/95", "type": "ssn:Observation", "ssn:observedBy": "http://example.iot.system/sensor/monitoring", "ssn:observationProperty": { "type": "vital:errors" }, "ssn:observationResultTime": { "time:inXSDDateTime": "2015-06-14T19:37:22Z" }, "ssn:observationResult": { "type": "ssn:SensorOutput", "ssn:hasValue": { "type": "ssn:ObservationValue", "value": 1.0, "qudt:unit": "qudt:Number" } } }, { "@context": { "@vocab": "http://vital-iot.eu/ontology/ns/", "vital": "http://vital-iot.eu/ontology/ns/", "vital": "http://vital-iot.com/ontology#", "lsm": "http://lsm.derii.ie/ont/lsm.owl#", "ssn": "http://purl.oclc.org/NET/ssnx/ssn#", "geo": "http://www.w3.org/2003/01/geo/wgs84_pos#", "time": "http://www.w3.org/2006/time#", "qudt": "http://qudt.org/vocab/unit#", "xsd": "http://www.w3.org/2001/XMLSchema#", "id": "@id", "type": "@type", "time:inXSDDateTime": { "@type": "xsd:dateTime" }, "qudt:unit": { "@type": "@id" } }, "uri": "http://example.iot.system/sensor/monitoring/observation/servedRequests/96", "type": "ssn:Observation",</pre>

	<pre> "ssn:observedBy": "http://example.iot.system/sensor/monitoring", "ssn:observationProperty": { "type": "vital:servedRequests" }, "ssn:observationResultTime": { "time:inXSDDateTime": "2015-06-14T19:37:22Z" }, "ssn:observationResult": { "type": "ssn:SensorOutput", "ssn:hasValue": { "type": "ssn:ObservationValue", "value": 5.0, "qudt:unit": "qudt:Number" } } } } </pre>
Notes	<ul style="list-style-type: none"> The response contains the observation values of the system and for the specified types in the request

3.1.2 Configuration hooks

The Management & Governance layer will use the GetConfigurationOptions to retrieve the exposed configuration parameters of the system. UpdateConfigurationOptions operation is then used to set/update one or more configuration parameters with “rw” permission, and provided that the IoT system supports the operation (as shown in Table 2).

Table 7: GetConfigurationOptions response data.

	GetConfigurationOptions metadata	
Description	VITAL pulls from an IoT system metadata about the supported configuration options.	
Method	GET	
Request headers	Content-Type	application/json
Request body		
Response headers	Content-Type	application/json
Response body	Example <pre> { "parameters": [{ "name": "configuration_parameter_1", "value": "some value that will not change", "type": "http://www.w3.org/2001/XMLSchema#string", "permissions": "r" }, { "name": "configuration_parameter_2", "value": "value 2", "type": "http://www.w3.org/2001/XMLSchema#string", "permissions": "rw" }, { "name": "configuration_parameter_3", "value": "value 34", "type": "http://www.w3.org/2001/XMLSchema#string", "permissions": "rw" }, { "name": "configuration_parameter_4", "value": "some value that will not change", "type": "http://www.w3.org/2001/XMLSchema#string", </pre>	

	<pre> "permissions": "r" }, { "name": "configuration_parameter_5", "value": "value 5", "type": "http://www.w3.org/2001/XMLSchema#string", "permissions": "rw" }] }</pre>
Notes	<ul style="list-style-type: none"> The response contains the key, value pairs of all configuration parameters exposed to VITAL, along with permissions supported via the PPI interface (rw, r). The VITAL Management & Governance module is agnostic of the keys, but it uses the type parameter for visually rendering these values and providing an editing widget (for parameters with “rw” permission).

Table 8: UpdateConfigurationOptions response data.

	UpdateConfigurationOptions metadata	
Description	VITAL sends new values for configuration options.	
Method	POST	
Request headers	Content-Type	application/json
Request body	Example <pre> { "parameters": [{ "name": "configuration_parameter_2", "value": "value 2" }, { "name": "configuration_parameter_3", "value": "value 34" }, { "name": "configuration_parameter_5", "value": "value 5" }]</pre>	
Response headers	Content-Type	application/json
Response body	200 OK	
Notes	<ul style="list-style-type: none"> Only ‘rw’ configuration parameters are allowed Response is empty 	

3.1.3 ICO management

The management of ICOs is supported by the existing design without extra ontologies and services, provided that the Management and Governance layer is interested solely on the status of an ICO and its location data (if supported by the ICO). Configuration options will be supported via the Get/Set Configuration Options of the IoT System, as described above.

3.2 Managed Entities

This section identifies the different types of managed entities handled by the VITAL Management Layer.

3.2.1 IoT Systems

IoT Systems are managed through their VUAIs. They expose sets of metadata information not only for describing the system's capabilities and setup, but also for measuring performance. The management system utilizes the subset of the available information that is required for managing the health of the system.

The following is a list of generic parameters that the management system utilizes:

Property	Type	Description
id	Static	A unique identifier to differentiate entities
type	Static	The type of the entity, for systems this is vital:IoTSystem
name	Static	A name of the system, for displaying on screen
description	Static	A description of the system, for displaying on screen
status	Dynamic	<p>The status of the entity. This can contain info like 'vital:Running', 'vital:Unavailable', or any values that will be added in the ontology as required by the systems.</p> <p>In the monitoring screen the status will be the first indicator that the system is working properly or not.</p>
operator	Static	The system's operator's contact information
serviceArea	Static	Describes the area that the system covers, for example a geographic polygon (Basic Geo (WGS84) ontology)
providesService	Dynamic	A list of ids of the available services offered by the system. More specific information for the services are obtained through an extra call to the system, described in the next section
providesSystem	Static	A list of underlying systems, if this IoT system is a wrapper for other system in a hierarchical topology
hasNetworkConnection	Static	Information on the underlying network infrastructure (wired/wireless, bandwidth etc.)
deviceHardware	Static	Description of the hardware where this system is installed (CPU, memory, hard disk etc.)

Below, is a list of performance measurements that should be measured by the IoT system and exposed to the management platform via the **MonitoringService** of the system. This list is not complete and not all metrics are mandatory. Any system developer should review these and implement the metrics that fit their systems, expanding the list as necessary.

Property	Type	Description
SysUptime	System Metric	The time in msec since the last restart
SysLoad	System Metric	A number indicating the load of the system (similar to the unix load http://en.wikipedia.org/wiki/Load_%28computing%29).
UsedMem	System Metric	The memory in bytes used by the system
AvailableMem	System Metric	The memory in bytes available to the system
ServedRequests	Performance Metric	The total number of requests served by this system. This is an always-increasing number (can be reset when restarting). Requests can be http requests for operation on the system (eg. data retrieval)
PendingRequests	Performance Metric	The number of pending requests at the time of the measurement
MaxRequests	Performance Metric	The number of requests per a period of time (seconds, minutes) the entity can serve simultaneously (e.g. 10 requests/sec). It indicates the maximum operation capabilities of the service.
Errors	Performance Metric	The total number of errors occurred from this entity. This is always an increasing number (can be reset when restarting).

With the parameters above we can calculate higher-level performance metrics like:

- $\text{throughput} = (\text{request}(t_2) - \text{request}(t_1)) / (t_2 - t_1)$
- $\text{utilization} = \text{throughput} / \text{maxRequests}$
- $\text{error percentage} = \text{error} / \text{requests}$
- $\text{error rate} = (\text{error}(t_2) - \text{error}(t_1)) / (t_2 - t_1)$

Another option would be to integrate these higher-level metrics directly into the system and let the IoT system provide them. Either approach is viable.

3.2.2 Services

Services are provided by systems. We treat them as managed entities, as the services advertises by systems, provide the necessary information to deduct whether the system is manageable (i.e. does it measure and provide monitoring metrics, can it be configured) and how to actually perform management operations.

In addition it displays the extra functionalities of the system, useful for displaying in the topology of Vital, for example, filter systems that support the ObservationService.

Property	Type	Description
id	Static	A unique identifier to differentiate entities
type	Static	The type of the service. Important services for the management platform are described in sections Error! Reference source not found. 3.1.2
name	Static	The name of the ICO for displaying on screen
description	Static	A description of the system for displaying on screen
operations	Static	A list of operations this service supports. With this information any external application can find the proper way to actually use the service (URL, HTTP Method)

3.2.3 Sensors / ICOs

For a comprehensive list of both the topology and the current state of the overall VITAL deployment, the management system needs information on the specific ICOs / Sensors.

The following is a list of the metadata parameters that the management system utilizes for sensors:

Property	Type	Description
id	Static	A unique identifier to differentiate entities
name	Static	The name of the ICO for displaying on screen
description	Static	A description of the system for displaying on screen
status	Dynamic	The status of the ICO. This can contain info like 'active', 'disabled', 'malfunctioning', 'low battery', 'network error' etc. In the monitoring screen the status will be the

		first indicator that the system is working properly or not.
ssn:observes	Static	A list of measurements this ICO performs (e.g. temperature, average traffic speed etc.)
hasLastKnownLocation	Dynamic	The latest geographic location of the ICO on the map. This is useful to display the HealthMap of the ICOs on a geographic map.
hasMovementPattern	Dynamic	Provides information on both the type of the movement pattern (Stationary Mobile Predicted) and some dynamic information on current direction and speed. Useful to track the location of the ICO on the map.
hasNetworkConnection	Static	Information on the underlying network infrastructure (wired/wireless, bandwidth etc.)
deviceHardware	Static	Description of the hardware this system is installed (CPU, memory, hard disk etc.)

ICOs are not associated with performance metrics, since access to ICOs is via IoT/Systems and PPIs that are already monitored and managed.

3.2.4 VITAL Platform Modules

VITAL Platform modules are monitored and managed in the same manner as IoT Systems through their VUAs. All parameters mentioned in 3.2.1 are also applicable for VITAL Modules. However, each module may have additional requirements (or maybe less) and modification to the exposed management information may be necessary. These extensions are described in the following paragraphs.

In addition, some modules provide required functionality to the management platform. For example, the Service Discover module provides information on the topology of the VITAL deployment (list of IoT Systems, ICOs and VITAL modules) and the Data Management module provides access to measurements without connecting directly to PPIs.

3.2.4.1 Service Discover

The Service Discovery module provides the means for discovering ICOs, IoT services, and IoT data that are virtualized in the VITAL platform.

The other modules of the architecture (e.g. CEP, Orchestrator) will access the Service Discovery through RESTful interfaces in order to operate on the IoT resources that are needed for their particular business context.

Regarding the Management Layer, the Service Discovery exposes a set of RESTful APIs as:

Property	Type	Description
<i>connDMS</i>	Static	Gives information about the connection with

		the Data Management Service
<i>getAlloTSystems</i>	Dynamic	Provides the list of all registered system's URIs

The Discovery module, according to specifications, will support the MonitoringService required by the Management Platform. Such a component is not implemented at the time we are writing this document, but will be introduced in the next version.

Further parameters will be defined in future according to project requirements.

3.2.4.2 VITAL Data Management Services

The VITAL Data Management Services (DMS) are defined as RESTful web services. These services will be used to get data from other sensors, systems or services. Data could be metadata as well as measurements.

The management platform will monitor the DMS status and performance based on the parameters defined in 3.2.1. DMS will also be used by the management platform as a silo of data and metadata for all the other components of the VITAL deployments. For example, retrieving measurements from a specific IoT system in a time period can be executed as a request to the DMS.

Regarding the Management Layer (and other VITAL components), the DMS exposes two RESTful APIs as:

Property	Type	Description
<i>sparql</i>	Dynamic	A SPARQL endpoint to query ICO observations stored in DMS. Accepts a SPARQL query and a returnType flag. Returns information in RDF/XML, JSON-LD or JSON format.
<i>metadata</i>	Dynamic	This interface provides stored metadata of an IoT system, an ICO, or an IoT service). Returns information in RDF/XML, JSON-LD or JSON format.

3.2.4.3 CEP

The CEP (Complex Event Processing) module is responsible for managing diverse IoT events stemming from different platforms and applications. The management platform can benefit from the CEP functionality, by defining management specific event processing instructions that the CEP will apply. For example, CEP can process streams of ICO status changes from IoTs and generate aggregated notifications to the management platform (for example “N ICOs report a malfunction”).

In a wide sense, the functionality offered by the CEP is to process data streams, but the aim of this data processing depends on users intentions that are expressed by means of Dolce specifications. The CEP is able to run many Dolce specifications to fulfil several request from different users. In order to avoid collisions between the

Dolce specifications the VITAL CEP dynamically deploys and undeploys new instances of CEP.

All the instances of the VITAL CEP are internally managed and monitored but the CEP manager module and will be announced to the management platform via the VUAI's service metadata endpoint and the providesService parameter of systems

Property	Type	Description
providesService	Dynamic	A list of CEP instaces. Apart from the static ones (MonitoringService, ConfigurationService ...), this list is updated by CEP manager module.

The management module of CEP will support the MonitoringService and ConfigurationService required by the Management platform. The specification of the CEP management will be introduced in the next version. In the next iteration of VITAL CEP development, it is planned to provide the below listed performance metrics per each CEP instance;

- **Generic:** Status, UsedMem, AvailableMem, SysLoad
- **CEP Specific:** Number of Rules monitored, Number of Complex events detected in a period of time, Number of idle CEPicos

3.2.4.4 VITAL Orchestrator module

The goal of the VITAL Orchestrator /BPM module is to achieve cross-platform & cross-business-context process integration in sync to the VITAL overall goal to provide an abstract digital layer over the application silos of the modern smart city.

The Vital Orchestrator implements the VUAI specs and supports the MonitoringService and ConfigurationService, required by the Management Platform. At the time of writing this document, this module does not implement specific performance metrics or configuration parameters; these will be added in the next version. It does however provide system and services metadata, as to announce the capabilities of the system, and the services it supports.

While, a generic IoT system's services are static in nature, the Orchestrator services are dynamically deployed and undeployed by users of the system. These dynamic services are monitored internally by the Orchestrator and are announced to the management platform, via the VUAI's service metadata endpoint and the providesService parameter of systems.

Property	Type	Description
providesService	Dynamic	A list of services offered by the system. Apart from the static ones (MonitoringService, ConfigurationService ...), this list is updated with new services created by users.

3.2.5 Service Level Agreements

Service Level Agreements (SLAs) govern the relationships by defining the terms of engagement for the participating entities and VITAL Platform. These parameters will provide a fundamental ground for interactions by means of Quality of Service.

At the time of writing this document, this module does not implement specific performance metrics or configuration parameters; these will be added in the next version. The SLA monitoring module implements the VUAI specs and it will support the MonitoringService and ConfigurationService required by the Management Platform, that will be announced to the management platform via the VUAI's service metadata endpoint and the providesService parameter of systems

SLA management model is in progress as a different deliverable (D5.3.1 – Smart Governance Toolkit V1, due for month 24).

3.2.5.1 SLA Parameters

Below are the first versions of SLA parameters that will be monitored for the quality of service. For the next iterations, these parameters will be updated and new parameters will be added, if necessary.

In the first version of this functionality it will be released the module in charge of evaluating the incoming data to set the Reputation Scores. This first version will use the interface offered by the PPI in order to get the SLA values of each ICO.

The table below contains the list of parameters used in the first release to calculate the Reputation, which are based on the parameters provided by the PPI interface.

Property	Type	Description
Uptime availability	Percentage	Uptime percentage of the source IoT system/ICO
Max. Number of Request	Numeric	Max. Number of the request limit for a given period of time. (for example, max 300 request in 60 seconds)
Response time	Millisecond	Max. Response time of data request in milliseconds. This value is calculated within the max request value limit in a given period of time.
Max. Number of Requests for a specific user	Numeric	Max. Number of the request limit for a specific user in a given period of time. (for example, max 300 request in a day per user)
Mean time to restore	Seconds	The time value for service restore from an unexpected service failure

In the second release of this module we plan to add more parameters to calculate Reputation of ICO and systems, The next table show a list of parameters that will be

processed, In this second version the SLA Monitoring will fetch all the required data from DMS module.

Property	Type	Description
Uptime availability	Percentage	Uptime percentage of the source IoT system/ICO
Max. Number of Request	Numeric	Max. Number of the request limit for a given period of time. (for example, max 300 request in 60 seconds)
Response time	Millisecond	Max. Response time of data request in milliseconds. This value is calculated within the max request value limit in a given period of time.
Mean time to restore	Seconds	The time value for service restore from an unexpected service failure
Max. Number of Requests for a specific user	Numeric	Max. Number of the request limit for a specific user in a given period of time. (for example, max 300 request in a day per user)
DatasetsVocabularySyntax	Numeric	Max number of vocabulary errors into the Datasets
Data Stability	Numeric	Max number of errors regarding some aspects of data values: <ul style="list-style-type: none"> • Numeric values are into thresholds • Variance and standar deviation • Random values: to detect where a data source is generating values randomly
Data Correlation	Percentage	Deviation of the data provided by two or more ICOs,when they can be considered similar (nature of the observation, localization, etc)

3.2.6 City Specific Information

This is directly related to the Governance tools of the VITAL platform. Since the related task has not produced concrete results yet and it is an on-going work at the time of writing this deliverable no specific parameters will be mentioned here. This information is related to deployment specific data that can be configured via the management layer UI for adapting the VITAL platform to the specific needs of the deployment environment (city) derived by the different social, economic and business parameters. The management UI will provide an overview of these data and allow the administrator of the platform to modify them when the need arises.

4 FUNCTIONAL AND TECHNICAL SPECIFICATIONS

4.1 Management Functionality Supported

Scope of the management platform is to provide the necessary tools for ensuring the smooth operation of the overall VITAL architecture. To this end, it supports the following functions:

4.1.1 Topology

The management platform, with the support of the Service Discoverer, is able to gather and display information on all the entities that participate in the VITAL architecture. The administrator is able to identify on the screen (a) IoT Systems: PPIs and Vital Modules (b) ICOs/Sensors with enough administrative information on the purpose of each component.

In this version of the platform, the management platform is extended to support multiple managed entities, discovered through two options: (1) from a predefined list of VUAI URLs (system endpoints), and direct connections to each system to retrieve their metadata (2) through queries to the Service Discovery module. The second option is designed but not integrated at the time of this deliverable. It will be supported in the next version.

4.1.2 Monitoring

Apart from the discovery of each module, administrators need to detect **faults** and **performance** bottlenecks. Thus, in addition to the topology a set of metrics are **monitored** and displayed on screen. The metrics for each component are defined in chapter 3.2. Scope of the management platform is to provide health-maps and dashboards to administrators, with which they will be able to identify problems as they occur.

To this end, two functionalities are supported. The first one is a location-based health-map that displays all participating entities in a geographical map, colour coded to indicate their current status. For entities without location information, a grid-based health-map is also available. These are the first level view of the overall system where the administrator can check the overall system at a glance.

The second one is a dynamic dashboard that is tied to specific managed components. Each component is described by its ontology as a JSON-LD document. The dashboard's scope is to map each part of the ontology to specific visualizations and create dynamic and agnostic views for every component. To this end, a set of different widgets have been developed directed to the different parts of the ontology:

Widget	Description
Information	Generic information on the entity, like name, description, to differentiate each entity and describe it's high-level functions
Status	This is the current status of the managed module. A green colour is associated with a functional module but other colours indicate problems.

Services	More applicable to IoT Systems and VITAL Modules, it displays a list of the services and operations this entity supports
Area	Displays a polygon in a geographic map. This is used in IoT Systems to describe which area they cover.
Location	Displays a marker with the accurate location of the entity on the map. More focused on ICOs that define a specific location and not an area (in contrast to IoT systems that provide area coverage visualized as a polygon on the geographic map). In future iterations, it will be possible to draw a trajectory if this ICO is mobile.
Hardware	Displays information on the hardware, CPU, memory, hard disk storage etc.
Network	Displays information on network connectivity infrastructure (wired/wireless, delay, bandwidth and). Useful for systems to display infrastructure or ICOs for performance
Errors	This widget fetches the errors and requests served by a system and draw a line graph with the error-rate for a period and a gauge with the error percentage.
Load	Retrieves load information and draws a gauge chart with the latest observation and a line chart with history of the load. An increasing number indicates a performance problem.
Memory	Displays the memory used and the memory available in a pie chart
Throughput	Based on the requests served in a time period it calculates the throughput and visualizes it on a line chart
Utilization	Calculates utilization from the throughput and the maximum request capacity of the module. It displays this info in a gauge chart

4.1.3 Alerts / Events

The management platform will support dynamic alerts both by exploiting the health information of the system and by connecting to the CEP module for higher-level alerts.

Due to the capability of CEP of processing big amounts of data in real time, it can be used to monitor data stemming from ICOs in order to detect inappropriate values or the lack of data that can be a hint of malfunctioning of an ICO. Some of the aspects that can be monitored by CEP are:

- Data not received. It can indicate, for instance, problems in the communication channel of the ICO, ICO is off, etc.
- Data out of thresholds. It can indicate ICO malfunctioning.
- Trends of data not according to the expected behaviour.
- Correlation of data between similar ICOs, which monitor the same aspect or physical property, at the same geographical position.

The interface to set the required Dolce specifications will be the REST API offered by the management module of CEP as described in “D3.2.2 Specification and Implementation of Virtualized Unified Access Interfaces V2”.

Alerts generated in CEP base on this processing will be sent to DMS. It is possible to send the alarms; filtered by content, source, etc.; to the other functional modules or applications.

4.1.4 Configuration

For each component, a separate service that exposes its configuration parameters is defined. Simple examples would include enable/disable, numberOfThreads etc.

The “Configuration” widget in the management platform provides a user interface that dynamically adapts to the ontology description of each configurable property and allows the administrator to make changes to the system. Its main purpose is to display a different type of widget for editing each property, for example a simple textbox for string parameters, or a date-picker for date parameters. Currently only string parameters are required from the existing system implementations, thus only textboxes are generated in the UI.

4.1.5 Security

In terms of security, the management platform is expanded to utilize the underlying infrastructure of VITAL and allow management of users and authorizations policies and display basic statistics about user/application access to provide a monitoring of the access control system availability and possibly detect anomalous behaviours.

4.1.6 Accounting

The management layer is expected to monitor and allow the aggregation of accounting information for the platform, i.e. the usage of platform resources and underlying IoT system resources by each of the applications operating on top of the VITAL platform. The accounting module will use instrumentation provided by the VITAL modules and the PPIs for monitoring accounting information and visualizing usage per application in a hierarchical fashion (overall VITAL usage and drill down option for viewing usage at the individual component). The goal is to provide a tool for monitoring the fair use and sharing of resources among applications. The accounting functionality will support the definition of monitoring & alert processes that can gather and store historical data in the management store.

4.2 Technical Architecture and Requirements

The following high-level diagram (Figure 2) provides an overview of the technical architecture of the Management Layer of VITAL as specified in T5.1.

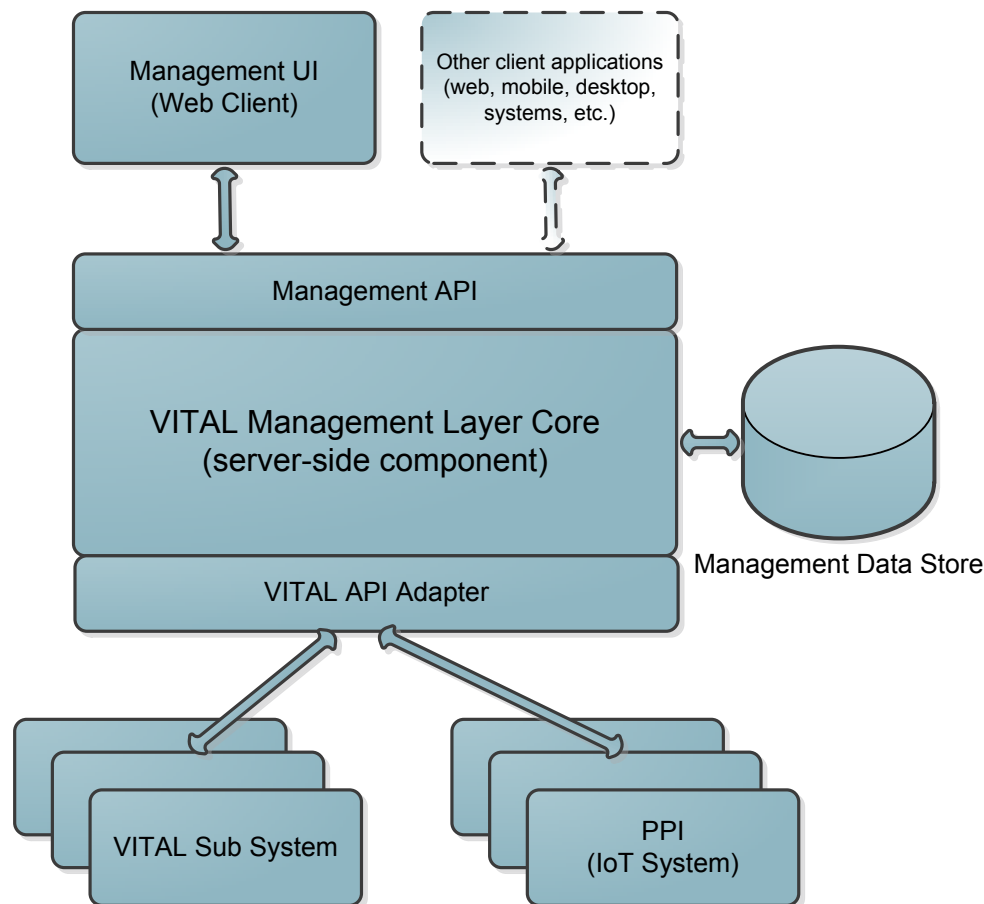


Figure 2: Management Layer technical architecture

The server side part of the Management Layer sub-system consists of the following components:

- The core server process. This is the implementation of the management business logic including scheduling of polling requests, handling of push notifications by other VITAL components, alerts setup and initiation, processing of first-hand management data and calculation of metrics, storage and retrieval of stored data (historical, accounting), event generation, etc. This is the heart of the Management Layer sub-system that controls and coordinates all other components explained below. This is being implemented in Java/JEE and it is deployed in a JEE application server; we selected the Wildfly² 8.x server, a widely used and open source JEE server.
- Management data store. This component handles storage/search/retrieval of configuration, monitoring and historical data; it can also be used for caching of information for improved performance. Since all data in VITAL are exchanged between components in JSON-LD representation, the Management Layer uses a document store that can natively handle JSON documents. Currently

² <http://wildfly.org/>

the design and implementation is based on Elasticsearch³, an open source, highly scalable JSON document store and full text index/search server.

- **Management UI.** This is the web client of the management layer that provides a rich UI to the administrator and implements the functionality specified in this deliverable. It is a Javascript/HTML5/CSS3 single page application based on the AngularJS⁴ framework; it also relies on the bootstrap⁵ CSS/JS framework for implementing a responsive design / fluid UI that supports both desktop and mobile web browsers with various screen sizes.
- **VITAL API Adapter.** This is the implementation of the VITAL API client code that the management core uses for retrieving information and communicating with all the components of the VITAL architecture (VITAL sub-systems like the CEP, Discoverer, Orchestrator, etc. and the PPI management services). This is also a server-side component implemented in Java/JEE using the JAX-RS API/specification.
- **Management API.** The server component that implements the RESTful management API through which the Management UI communicates with the client applications. Currently this is only used by the Management UI web client application, but future clients or other components of the VITAL ecosystem can use it.

The design and implementation of this architecture should comply with the following requirements:

- **Usability.** Although the user of the management layer is the platform administrator (i.e. a technical savvy user), and not the end-user of the applications or the application developer, the user experience and user interface should be friendly, concise and not-overwhelmed with too much information. The goal is to support efficiency and early warning.
- **Layered and modular approach.** Easy maintenance and extensibility is important as the management layer may evolve rapidly with additional functionality as the platform matures and the supported IoT platforms increase in number and provide more monitoring and configuration hooks. Clear separation of layers and modularity is important in the design and implementation of the management layer component.
- **RESTful APIs / JSON-LD.** As a VITAL component the management layer must be able to consume and produce REST/JSON-LD data streams and conform to the VITAL semantic data model.
- **Security.** The application must be designed in a secure way that will protect its assets from OWASP-10 attacks and unprivileged access in general.

³ <http://www.elasticsearch.org/>

⁴ <https://angularjs.org/>

⁵ <http://getbootstrap.com/>

5 PROTOTYPE IMPLEMENTATION (V2)

5.1 Supported Functionality

The prototype implementation focuses on the following functionality:

- Overall UI design. Based on the UI/UX guidelines of the first version of the prototype, the UI design has been adopted for all additional modules since the first prototype.
- Health map of a VITAL deployment. The prototype implementation implements a geographical map view based on the OpenStreetMap data feed and demonstrates the Health map approach.
- Monitoring Dashboard. This is a set of entity agnostic management widgets/tools that visualize the VITAL ontologies. These are presented in a dynamic dashboard view as explained in the previous section. Currently the prototype contains implementations of all the widgets described in section 4.1.2, integrated with real live data from existing systems.
- Configuration View. This is a dynamically adapted configuration screen. It displays and allows editing the configuration parameters of systems.
- Security Management. In the current release, the security management module allows: (1) Registering/managing users, (2) Defining/managing groups/roles, (3) Assigning roles to users or revoking roles from users, (4) Defining/managing permissions, (4) Monitoring number of sessions and policy evaluations
- Event processing as a mean to build valuable information by correlating data stemming from different sources. The Vital CEP functionality will provide access to expert programmers through the administration interface and the Dolce Language, and also to non-expert users, through predefined Dolce specifications, that will be offered as services, and internally managed by the Management layer. This management comprises not only the dolce specifications, but the on the fly creation/deletion of new instances of the CEP.
- Data model and Communication layer. The prototype understands the VITAL ontologies and processes the JSON-LD syntax as specified by the project at this time.

5.2 Migration Plan: from prototype to concrete implementation

As explained earlier in the text, we adopted a top-down approach, starting with a UI prototype that relied on mock-up data streams and tried to identify specific functional requirements through multiple iterations. The current prototype implementation has been validated within the VITAL consortium by discussing and updating iterations, thus resulting in a first functional UX/UI design as well as the identification of more detailed functional and technical requirements. In addition to that, the prototype led to the initial implementation of the VITAL data model and the VITAL API module that was integrated with the PPIs for end-to-end manipulation and visualization of actual data produced by real IoT systems.

As part of the second release we have continued working in iterations gradually implementing all layers of the management layer architecture and integrating with the management instrumentation as they become available from the implementations of the VITAL core components (e.g. Discoverer, CEP) and the PPIs of the targeted IoT systems. The first iterations (until the next version of the deliverable) have focused on implementing and validating as much as possible of the functionality specified, while the final iterations (up to the final version of the deliverable) will focus on optimizations, debugging and fine-tuning.

As part of the second release of the deliverable the following updates have been implementing:

- **Dynamic Discovery:** Discovery of systems, services and sensors is now more dynamic. Two options are supported (1) a predefined list of VUAI URLs and direct connection to each VUAI for retrieving metadata (2) A complete list of VUAI URLs from the Service Discovery module. In the next phase, we will add more features in the dynamic discovery mechanism to support complex queries like search for sensors in an area or with specific type of observations.
- **Performance Monitoring:** All widgets from the first prototype that were based on mockup data, are now implemented to support real live data from systems, as defined in the Vital ontology. As new VUAI conformant systems are implemented they will be added to the list of monitored systems seamlessly.
- **Configuration Management:** The management platform is extended with a configuration module. This module interacts with configurable systems and displays configuration options on screen with the ability to edit them as necessary. This module will be able to configure all VUAI conformant systems as they are implemented and deployed in next releases of the Vital platform.
- **Security:** A module to monitor and configure security has been added in this release. It provides a web based interface to access and manage data about users, groups and policies and includes a module interacting with the OpenAM identity provider through REST APIs. The interface is already designed with the same look and feel and technologies as the other management UI modules and it will be fully integrated. In the next phase, the security and security management modules will be tested on the integrated VITAL prototype; functionalities and UI will be evaluated and, where needed, expanded to support simple and effective security configurability and monitoring
- **Event/Alert Management:** This module allows the management admin to define and trace event definitions and alert rules that will provide alerts/information about the data sources, such as sensor health, data validity etc
- **SLA management module:** It allows defining and managing the SLA parameters for data sources, and tracking of these parameters over the values provided by the PPI implementation of the data sources

The functionalities implemented as part of the second version are described in the following paragraphs.

5.3 Dynamic Topology Discovery

As stated Dynamic Discovery is based on two approaches, (a) a predefined list of VUAI URLs with direct interaction with systems and (b) via the Service Discovery module and DMS. In what follows we will describe the algorithm of the first approach, to showcase how systems actually describe their capabilities and how this information is then consumed to fill the topology database and be displayed on the screens of administrators. Next we will focus on the alternative of exploiting the Service Discovery and DMS modules.

Each VUAI exposes the following required API calls (described in detail in deliverable D3.2.2):

<BASE_URL>/metadata	Returns the system's static information as described in 3.2.1
<BASE_URL>/service/metadata	Returns the system's supported services and operation as described in 3.2.2
<BASE_URL>/sensor/metadata	Return the system sensors that it provides access to, as described in 3.2.3

The management platform, given the list of BASE_URLs, connects to each system and performs the following steps:

1. Stores in cache the static information of all components and uses it to generate the System Grid, Sensor Grid and Geographical Health-map (for components that contain geo-location data).
2. Searches for the following services: (a) MonitoringService (b) ConfigurationService to detect if the system is manageable.
3. Interacts with the two services to generate performance graphs, configuration hooks and live status updates.

This procedure is repeated every few minutes, to detect changes in the topology and update the management cache accordingly. Actual performance metrics are not retrieved from the cache, they are obtained through the systems directly.

Alternatively, instead of connecting to each VUAI directly, the management platform can exploit the added value services of Vital, i.e. the Service Discovery and DMS.

In order to retrieve, in a dynamic way, the different systems connected to VITAL, the Discovery module provides a function findAllSystems. To trigger the Discoverer a user has to send a POST request containing a JSON object. Since the discovery operation is meant to retrieve all systems, no input parameters are needed except those regarding security aspects.

Following the reception of the above mentioned POST request, the Discovery module instantiates a connection to the DMS to query all registered systems. The SPARQL query necessary for such a purpose is defined as follows:

PREFIX vital:<http://vital-iot.eu/ontology/ns/>

```

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

SELECT ?uri
WHERE {
    ?uri rdf:type vital:lotSystem
}

```

Information retrieved from the DMS is then processed to produce an output in JSON-LD format containing the URIs of all the systems registered in VITAL.

After detection of static information, DMS can be used to obtain measurements of performance metrics and live status updates, in place of direct connections to systems. DMS keeps the latest information, as provided by systems through the embedded PADA module.

The option of utilizing the SD and DMS will be fully integrated in the next version of the platform.

Following are a few screenshots of the UI, with the dynamically generated grid of IoT Systems and Sensors:

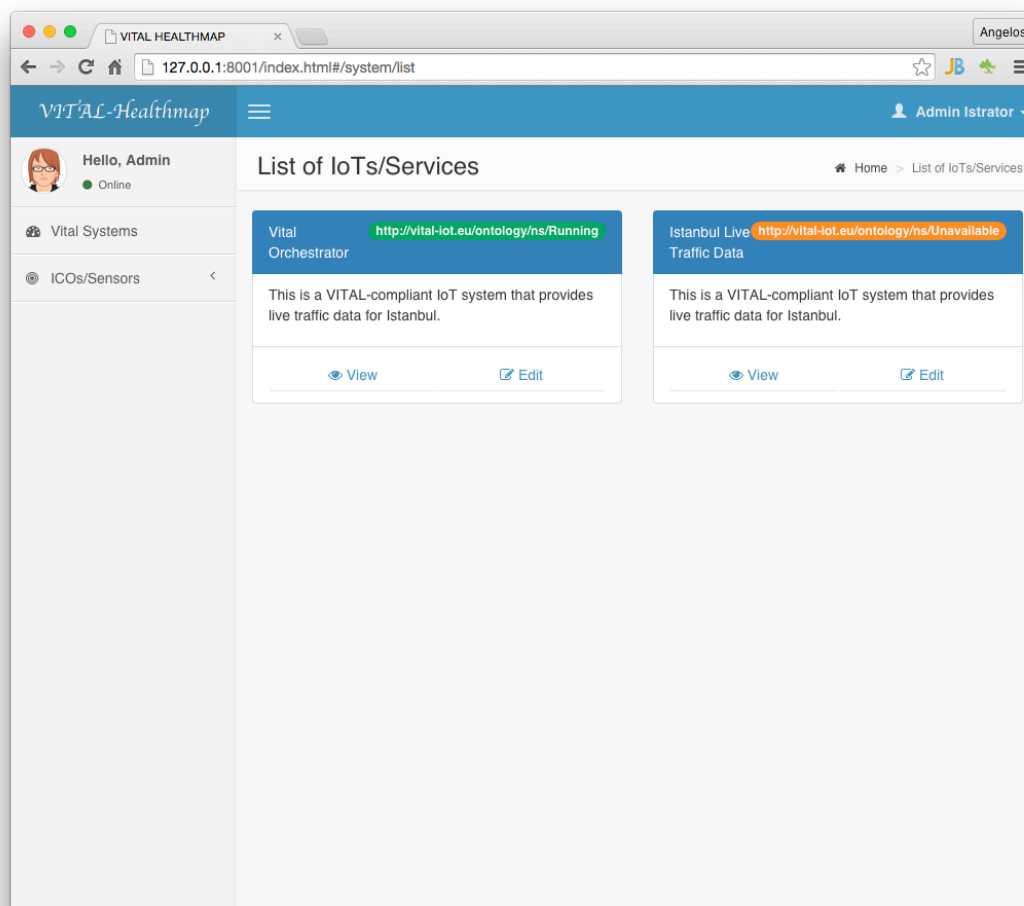


Figure 3 Management UI: List of Systems

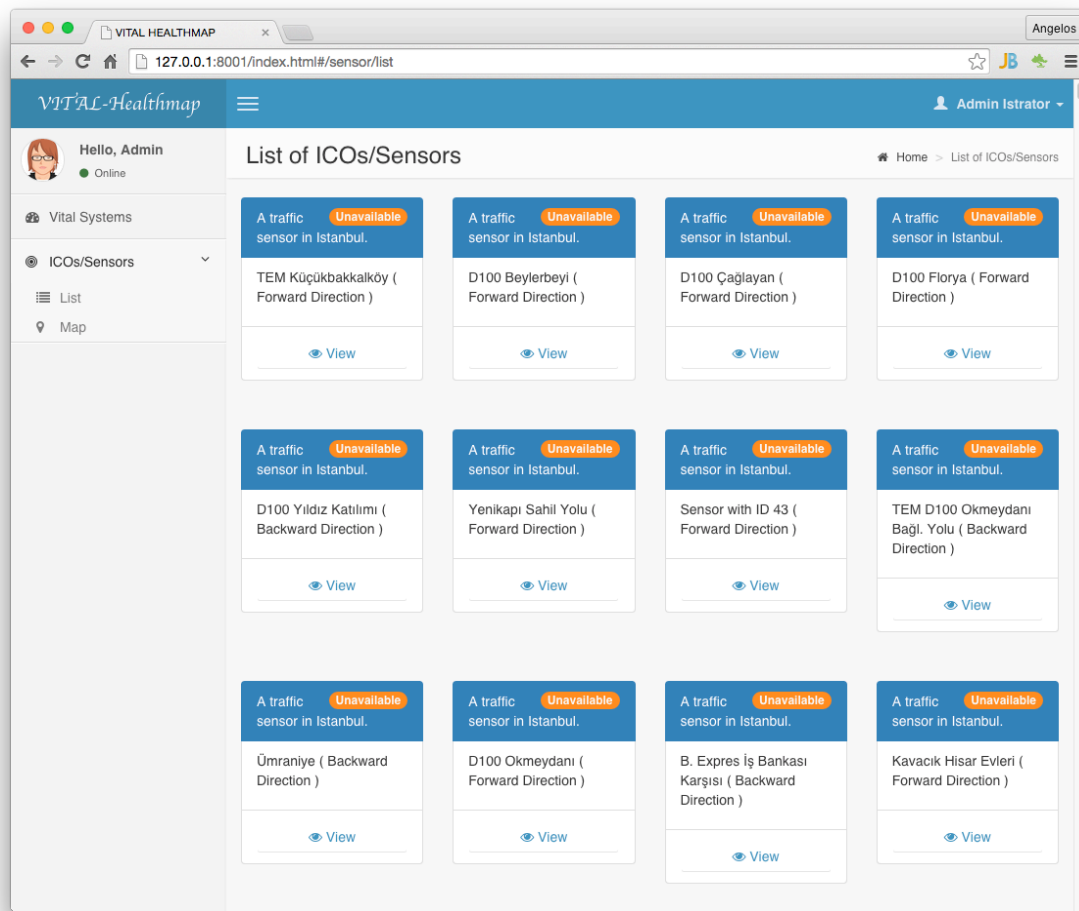


Figure 4: Management UI: List of Sensors

5.4 Performance Monitoring

The following are screenshots from the monitoring dashboard of systems and the overall health-map. These views are generated with a combination of static information and the live performance metrics as obtained by systems.

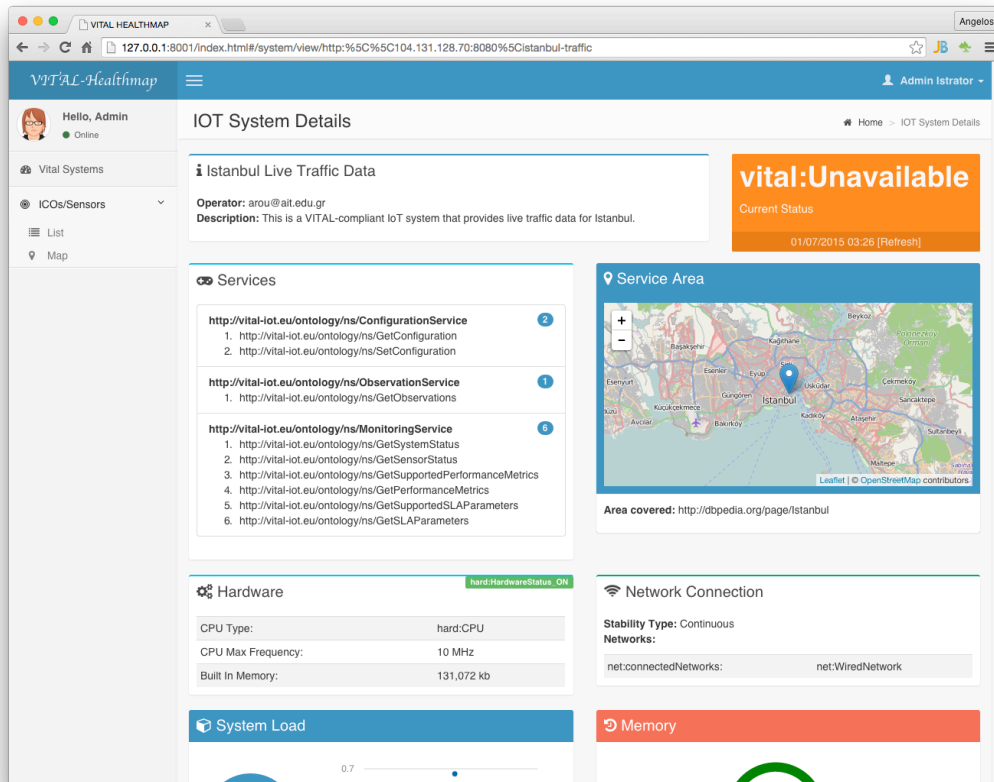


Figure 5: Management UI: System Overview – Info and Status

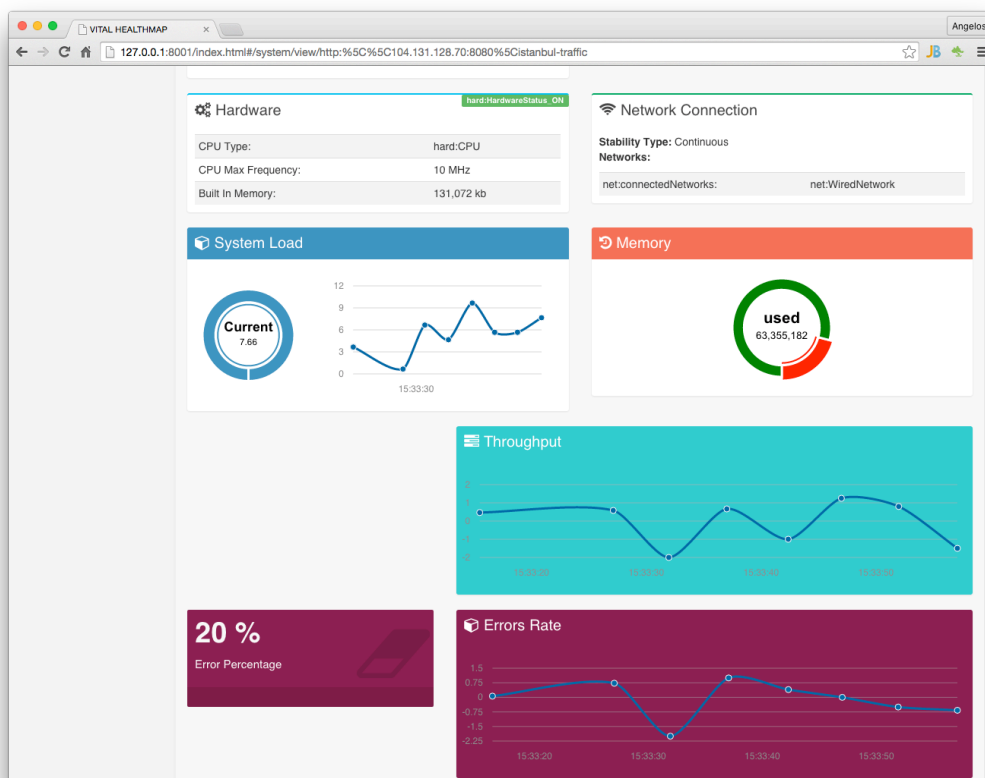


Figure 6: Management UI: System Overview - Performance Graphs

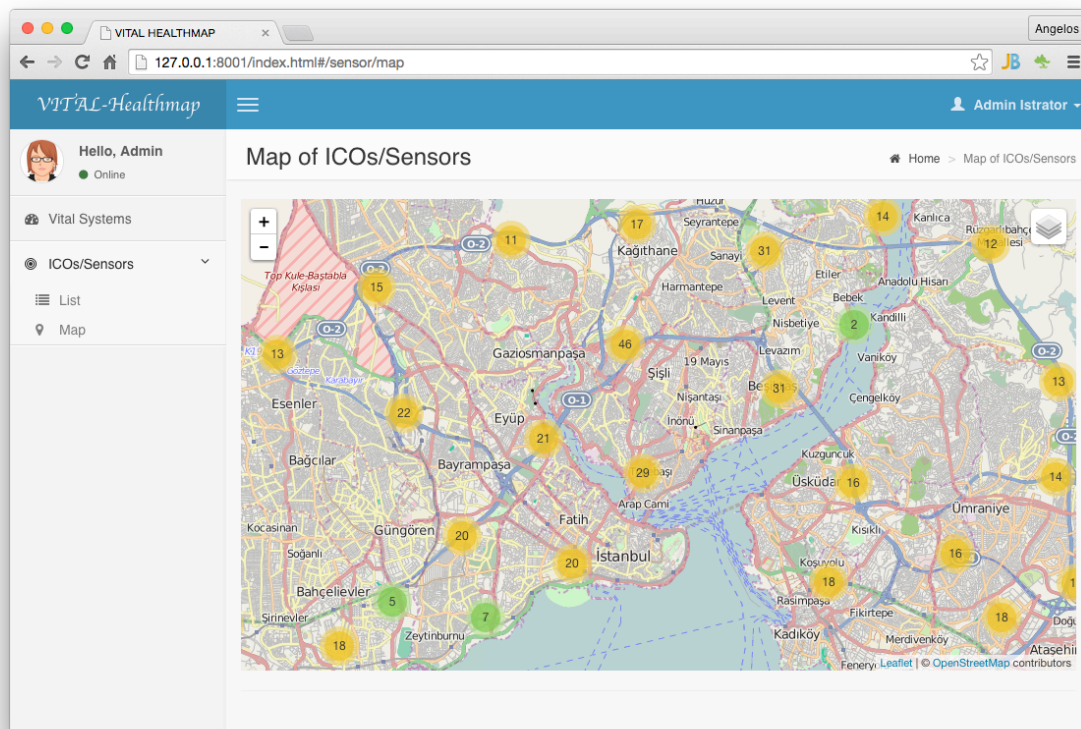


Figure 7: Management UI: Geographical Health-map

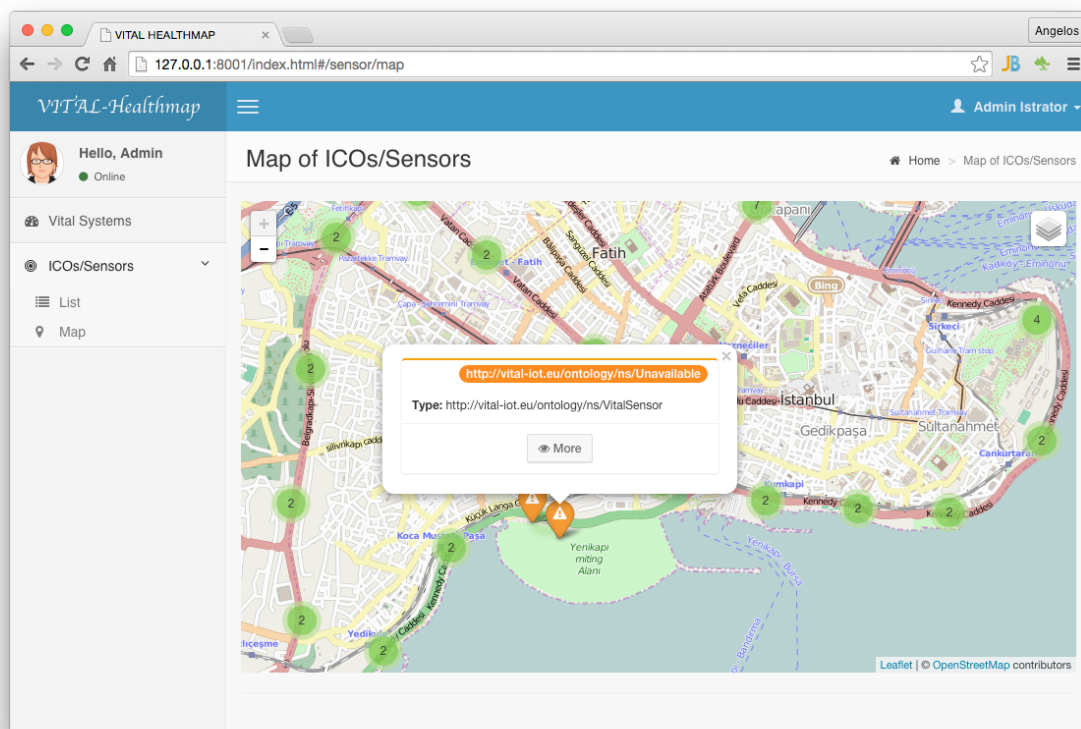


Figure 8: Management UI: Geographical Health-map - Sensor Details

5.5 Configuration Management

This is a screenshot of the configuration widget, as it is dynamically generated from the system's ConfigurationService. Some parameters are editable while others are read-only. Pressing the button "Update" will cause the management platform to directly connect to the system and send the new configuration.

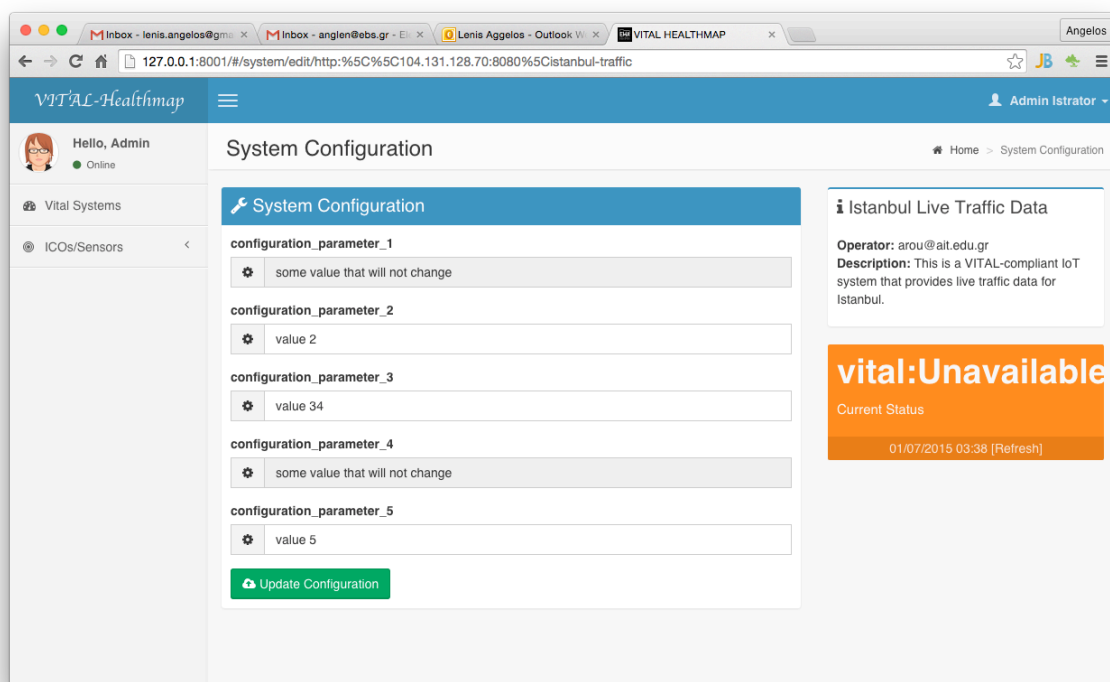


Figure 9: Management UI: Configuration

5.6 Alerting and Event Management Services

The following are the screenshots from the event management and monitoring dashboard of CEPICOs. The first one is the screenshot that shows the list of all the events and alerts generated by a specific CEPICO. Events and alerts are represented in different background colours.

Detected Events Control panel

[Home](#) > Detected Events

#	Event Name	Date	Event Data
10907	TrafficData	30.6.2015 14:06:46	[{"name":"TrafficData","Speed":"0.000000","Py":"29.045630","Px":"41.070557","Sensorid":"232"}]
10906	TrafficProblem	30.6.2015 14:06:46	[{"name":"TrafficProblem","Speed":"0.000000","Py":"29.045630","Px":"41.070557","Sensorid":"232"}]
10905	TrafficData	30.6.2015 14:06:45	[{"name":"TrafficData","Speed":"26.000000","Py":"29.048182","Px":"41.095734","Sensorid":"231"}]
10904	TrafficData	30.6.2015 14:06:44	[{"name":"TrafficData","Speed":"0.000000","Py":"29.052469","Px":"41.098129","Sensorid":"230"}]
10903	TrafficProblem	30.6.2015 14:06:44	[{"name":"TrafficProblem","Speed":"0.000000","Py":"29.052469","Px":"41.098129","Sensorid":"230"}]
10902	TrafficData	30.6.2015 14:06:42	[{"name":"TrafficData","Speed":"31.000000","Py":"29.036055","Px":"41.155113","Sensorid":"229"}]
10901	TrafficData	30.6.2015 14:06:41	[{"name":"TrafficData","Speed":"83.000000","Py":"28.881060","Px":"41.066868","Sensorid":"228"}]
10900	TrafficData	30.6.2015 14:06:41	[{"name":"TrafficData","Speed":"73.000000","Py":"28.881737","Px":"41.066963","Sensorid":"227"}]
10899	TrafficData	30.6.2015 14:06:40	[{"name":"TrafficData","Speed":"73.000000","Py":"28.885525","Px":"41.050488","Sensorid":"226"}]
10898	TrafficData	30.6.2015 14:06:39	[{"name":"TrafficData","Speed":"72.000000","Py":"28.855013","Px":"41.054737","Sensorid":"225"}]
10897	TrafficData	30.6.2015 14:06:38	[{"name":"TrafficData","Speed":"77.000000","Py":"28.891943","Px":"41.044365","Sensorid":"224"}]
10896	TrafficData	30.6.2015 14:06:36	[{"name":"TrafficData","Speed":"79.000000","Py":"28.917564","Px":"41.037674","Sensorid":"223"}]
10895	TrafficData	30.6.2015 14:06:35	[{"name":"TrafficData","Speed":"67.000000","Py":"28.922306","Px":"41.029831","Sensorid":"222"}]
10894	TrafficData	30.6.2015 14:06:34	[{"name":"TrafficData","Speed":"18.000000","Py":"28.930857","Px":"41.023201","Sensorid":"220"}]
10893	TrafficProblem	30.6.2015 14:06:34	[{"name":"TrafficProblem","Speed":"18.000000","Py":"28.930857","Px":"41.023201","Sensorid":"220"}]
10892	TrafficData	30.6.2015 14:06:34	[{"name":"TrafficData","Speed":"40.000000","Py":"28.931339","Px":"41.023468","Sensorid":"219"}]
10891	TrafficData	30.6.2015 14:06:33	[{"name":"TrafficData","Speed":"0.000000","Py":"28.977236","Px":"41.064781","Sensorid":"218"}]
10890	TrafficProblem	30.6.2015 14:06:33	[{"name":"TrafficProblem","Speed":"0.000000","Py":"28.977236","Px":"41.064781","Sensorid":"218"}]
10889	TrafficData	30.6.2015 14:06:32	[{"name":"TrafficData","Speed":"99.000000","Py":"28.977133","Px":"41.065277","Sensorid":"217"}]
10888	TrafficData	30.6.2015 14:06:30	[{"name":"TrafficData","Speed":"65.000000","Py":"28.968121","Px":"41.049583","Sensorid":"216"}]

The second screenshot represents the update of the DOLCE rule for a specific CEPICO. When user clicks the “Submit” button, management platform connect to the VITAL CEP and update the DOLCE rule.

DOLCE Specification <small>Home</small> > DOLCE Specification	
<div> DOLCE Specification used for Detection <pre> event ImmEvent { use { int sensorId, //Id for sensor float pointX, // Sensor Latitude float pointY, // Sensor Longitude float speed //Average Speed Km/h } } # Traffic Problem Detection complex TrafficProblem { payload { int SensorId = sensorId, float Px = pointX, float Py = pointY, float Speed = speed }; detect ImmEvent where speed < 20; } # Traffic Data Streaming to UI complex TrafficData { payload { int SensorId = sensorId, float Px=pointX, float Py=pointY, float Speed= speed }; detect ImmEvent; } </pre> </div>	
Submit	Reset

5.7 Security Management Functionalities

The Vital management layer includes the functionalities needed by an administrator to monitor and manage security. In the current release, the security management module allows:

- Registering/managing users
- Defining/managing groups/roles
- Assigning roles to users or revoking roles from users
- Defining/managing permissions
- Monitoring number of sessions and policy evaluations

Permissions are applied at the group level, CRUD (Create, Read, Update, Delete) actions are possible on users, groups, policies.

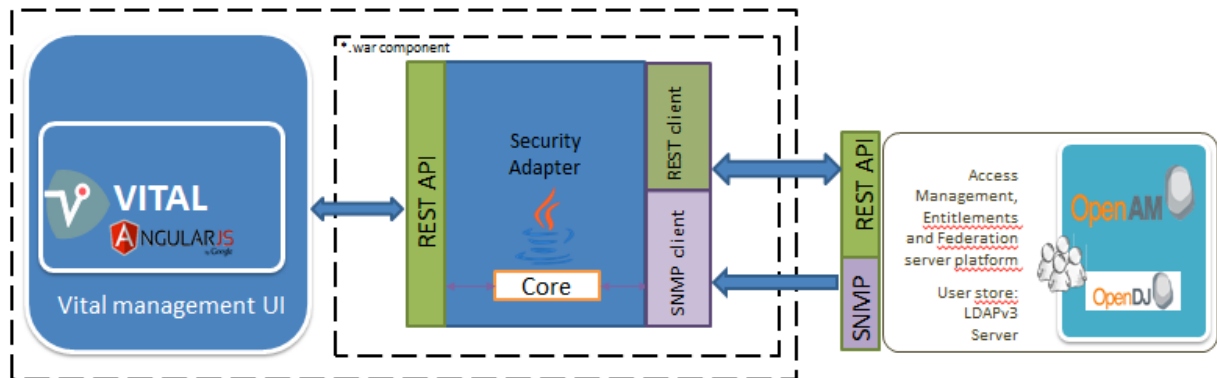


Figure 10 – Security management architecture

The Vital security management component interacts with the OpenAM identity provider through a security adapter component that accesses data about users, groups and policies through the REST APIs exposed by OpenAM. The security adapter component is a Java Web Archive file (*.war), meant to be deployed in a WildFly container.

It exposes a REST API used by Vital Management user interface. This interface is the 'Security' section of the web (Javascript/HTML5/CSS3) application based on the AngularJS framework.

It also retrieves information about the access control activity useful to monitor and ensure appropriate performance and service availability; these statistics are retrieved through the OpenAM SNMP (Simple Network Management Protocol) interface over UDP, using Object Identifiers defined in a Management Information Base (MIB) file. Currently reported figures include:

- Total number of current sessions
- Cumulative number of policies evaluations
- Average rate of policy evaluations

It is noteworthy that besides the information shown in the Monitor section, detailed trails can be found in the log files. OpenAM servers generate two types of log files: audit logs and debug logs. While debug logs are unstructured and mainly intended for debug and troubleshooting purposes, audit logs capture normal operational information about OpenAM usage. Audit file records are structured: they adhere to a consistent, documented file format, are by default logged to flat files but could be sent to relational database tables or a syslog server.

OpenAM log files are named after the service logging the message, with two types of log files per service: .access and .error. For instance the audit log files for the authentication service are named amAuthentication.access and amAuthentication.error.

Similarly, OpenAM policy agents also produce agent audit logs, too.

Section 5.7.1 describes the functionalities showing user interface snapshots, while Section 5.7.2 details the REST interfaces provided by the security adapter.

5.7.1 Security management user interface

As shown in Figure 11 the Security management functionalities are grouped in four subsections:

- Users
- Groups
- Policies
- Monitor

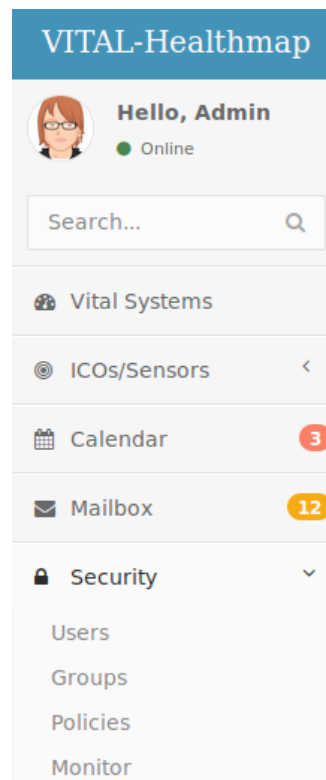


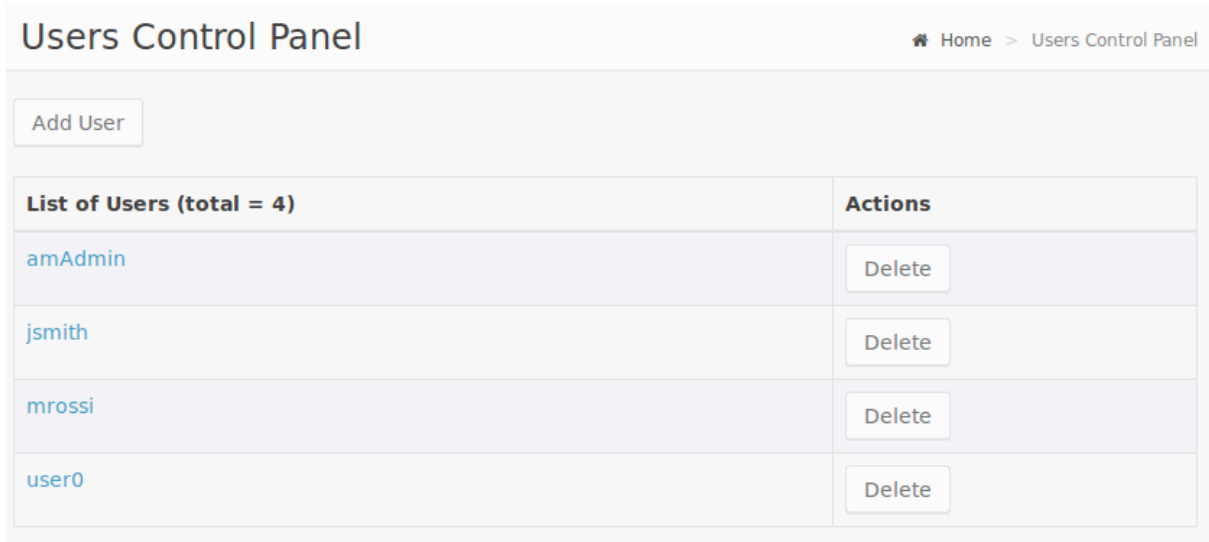
Figure 11 – Management menu including security section

The Users Control Panel, allows to list users (Figure 12), edit modifiable users attributes (e.g. email) (Figure 14), delete users or add new users (Figure 13).

From the Users panels it's also possible to manage the membership of a user.

Groups can be specifically managed from the Groups panel, which allows performing of CRUD actions.

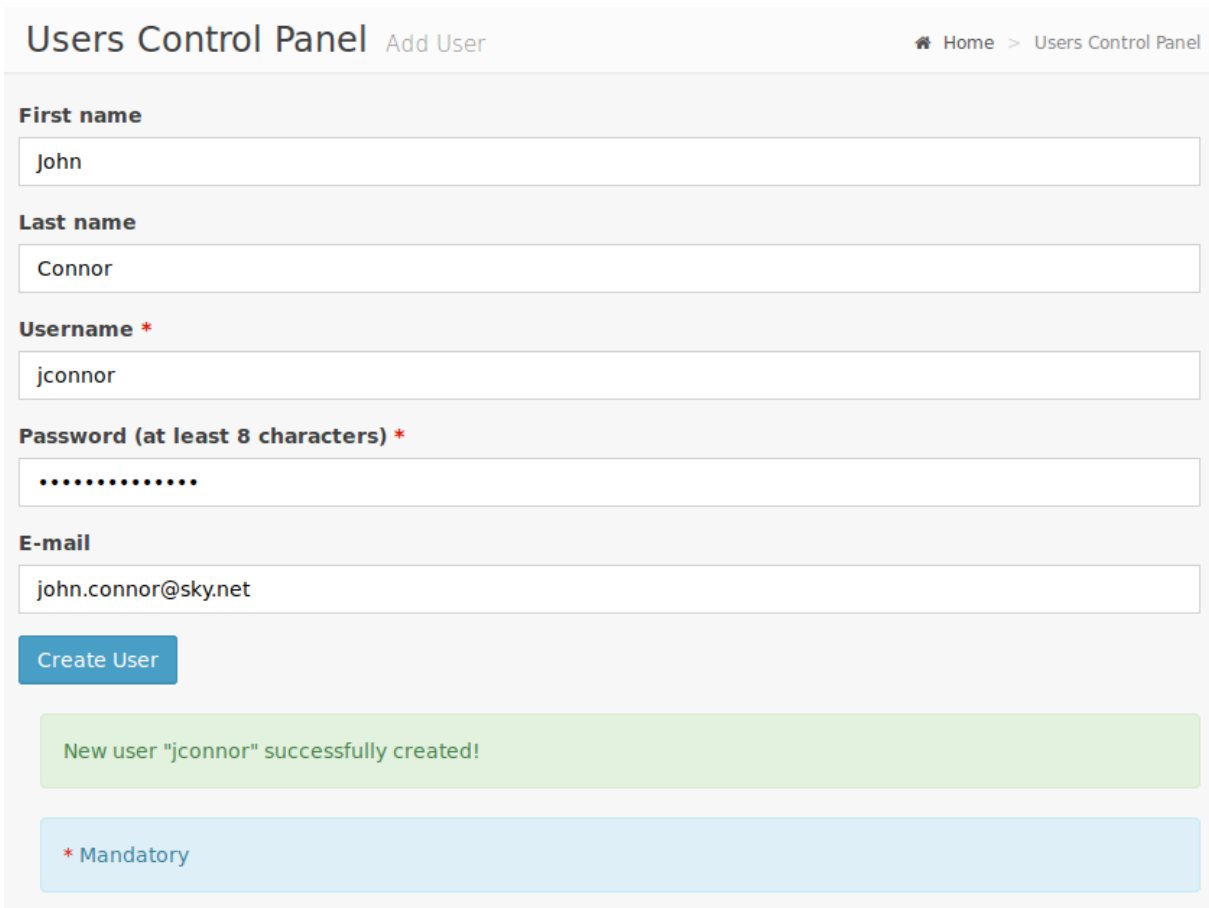
Groups are intended to represent roles and to simplify permissions assignment. It has been chosen that permissions cannot be directly assigned to users but must be assigned to a group/role. In this solution, roles and permissions are mapped to the group memberships of users (Figure 15).



The screenshot shows the 'Users Control Panel' interface. At the top right is a breadcrumb trail: Home > Users Control Panel. On the left, there is an 'Add User' button. The main content area contains a table with two columns: 'List of Users (total = 4)' and 'Actions'.

List of Users (total = 4)	Actions
amAdmin	Delete
jsmith	Delete
mrossi	Delete
user0	Delete

Figure 12 – Users listing



The screenshot shows the 'Users Control Panel' interface with the 'Add User' button highlighted. The form contains the following fields:

- First name**: John
- Last name**: Connor
- Username ***: jconnor
- Password (at least 8 characters) ***: (masked with dots)
- E-mail**: john.connor@sky.net

Below the form is a 'Create User' button. A green message box states: 'New user "jconnor" successfully created!'. A blue box at the bottom indicates: '* Mandatory'.

Figure 13 – User creation

Users Control Panel User Details [Home](#) > [Users Control Panel](#)

[Edit](#)

Successfully updated user details!

Username	Realm	First name	Last name	E-mail	Status
jconnor	/	John	Connor	john.connor@sky.net	Inactive

The user is part of no groups.

Add "jconnor" to group ▾

- Advanced_Users
- Base_Users
- Dev_Users

Figure 14 – User editing

Groups Control Panel Group Details [Home](#) > [Groups Control Panel](#)

Name	Realm
Advanced_Users	/

Add user to "Advanced_Users" ▾

Successfully added "jconnor" user to the group!

Group Users	Actions
user0	Remove from group
jconnor	Remove from group
amAdmin	Remove from group

Figure 15 – Group control

The Policies panel, which allows performing CRUD actions on policies, used to manage permissions, is shown in Figure 16.

Policies Control Panel [Add Policy](#) Home > Policies Control Panel

Policy name *

judgementday

Resource(s) *

http://example.com/resource/*

Add

http://terminator.net/* Remove

Group(s)

Authorize group ▾

Advanced_Users Remove

Dev_Users Remove

Create Policy

New policy "judgementday" successfully created!

* Mandatory

Figure 16 – Policy creation

For specifying the resources in the policy definition, the URL or resource name to protect can be specified individually or by using patterns with the wildcards * and -*-. These wildcards can be used throughout resource patterns to match URLs or resource names (they can be used to match protocols, host names, and port numbers too).

The wildcard * matches multiple levels in a path, while the wildcard -* - matches a single level in a path.

For example,

`http://www.example.com/-*-` matches `http://www.example.com/index.html` but does not match `http://www.example.com/company/images/logo.png`.

Instead `http://www.example.com/*` matches both of the above.

It shall be considered also that:

- Wildcards do not match ?. Instead, when used at the end of a pattern after a ? character, * matches one or more characters, not zero or more characters.

- Duplicate slashes (/) are not considered part of the resource name to match. A trailing slash is considered by OpenAM as part of the resource name.
- Wildcards cannot be escaped.
- It is recommended not to mix * and -* in the same pattern.
- Comparisons are not case sensitive (by default).

The Monitor panel, which allows monitoring of the load and performance of the access control system, is shown in Figure 17.

5.7.2 Security management REST interfaces

The security adapter exposes a RESTful interface allowing to perform the needed operations described above. The response bodies are coded in Json.

The list of APIs is detailed in Section 5.7.2.1, along with responses and status codes. Security management REST APIs respond to successful requests with HTTP status codes in the 2xx range, to error conditions with HTTP status codes in the 4xx and 5xx range.

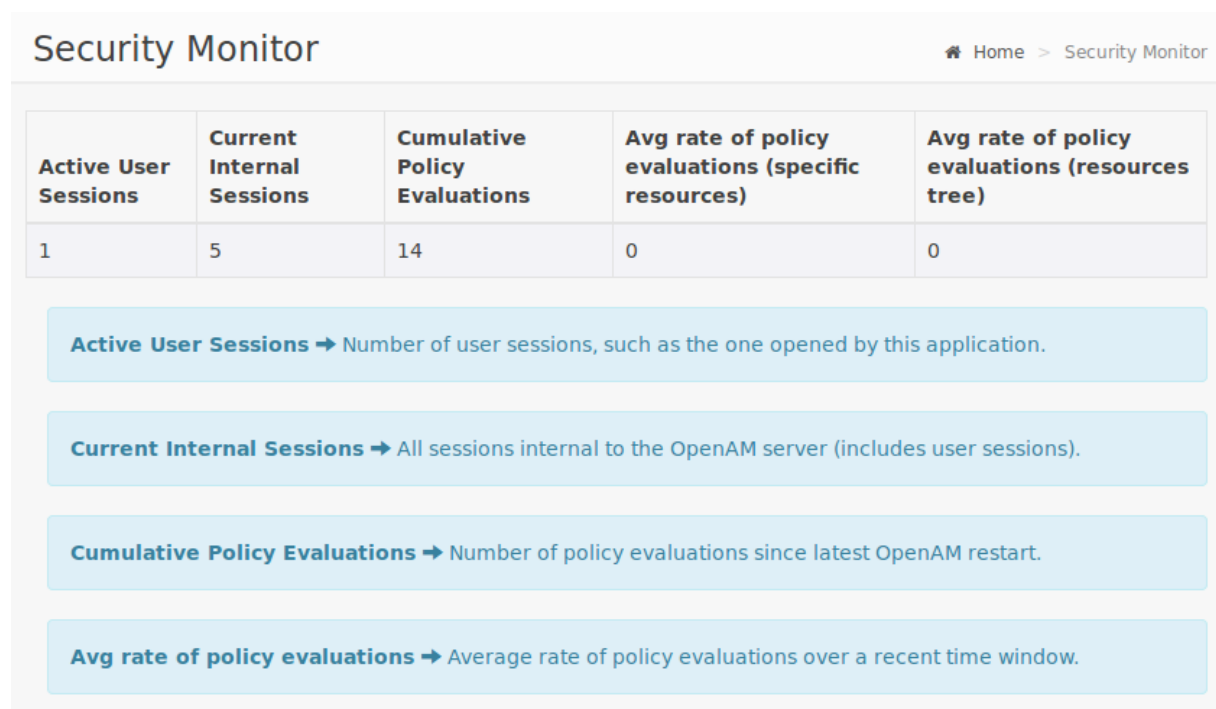


Figure 17 – Monitor interface

5.7.2.1 Security management REST API

The following APIs are exposed by the security adapter. In the current prototype the response bodies are the Json representations of the requested resources or of the errors coming from OpenAM.

DESCRIPTION	Returns the user information
TYPE	GET
ENDPOINT	/user/{id}
REQUEST PARAM	{id}: user id
RESPONSE BODY	Returns the info or error description

Table 9: getUserInfo

DESCRIPTION	Returns the group information
TYPE	GET
ENDPOINT	/group/{id}
REQUEST PARAM	{id}: group id
RESPONSE BODY	Returns the info or error description

Table 10: getGroupInfo

DESCRIPTION	Returns the policy information
TYPE	GET
ENDPOINT	/policy/{id}
REQUEST PARAM	{id}: policy id
RESPONSE BODY	Returns the info or error description

Table 11: getPolicyInfo

DESCRIPTION	Returns the users list
TYPE	GET
ENDPOINT	/users
REQUEST PARAM	//
RESPONSE BODY	Returns the list or error description

Table 12: getUsers

DESCRIPTION	Returns the groups list
TYPE	<i>GET</i>
ENDPOINT	/groups
REQUEST PARAM	//
RESPONSE BODY	Returns the list or error description

Table 13: getGroups

DESCRIPTION	Returns the policies list
TYPE	<i>GET</i>
ENDPOINT	/policies
REQUEST PARAM	//
RESPONSE BODY	Returns the list or error description

Table 14: getPolicies

DESCRIPTION	Returns the user groups list
TYPE	<i>GET</i>
ENDPOINT	/user/{id}/groups
REQUEST PARAM	{id}: user id
RESPONSE BODY	Returns the list or error description

Table 15: getUserGroups

DESCRIPTION	Returns the monitoring information
TYPE	<i>GET</i>
ENDPOINT	/stats
REQUEST PARAM	//
RESPONSE BODY	Returns the info or error description

Table 16: getStats

DESCRIPTION	Create a user
TYPE	POST
ENDPOINT	/user/create
REQUEST BODY	Format: application/x-www-form-urlencoded Parameters: <ul style="list-style-type: none"> ▪ <i>Given name (optional)</i> ▪ <i>Surname (optional)</i> ▪ <i>Username</i> ▪ <i>Password</i> ▪ <i>Email (optional)</i>
RESPONSE BODY	Returns the representation of the created user or error

Table 17: createUser

DESCRIPTION	Delete the specified user
TYPE	POST
ENDPOINT	/user/delete
REQUEST BODY	Format: application/x-www-form-urlencoded Parameters: <ul style="list-style-type: none"> ▪ <i>Username</i>
RESPONSE BODY	Returns error representation on failure

Table 18: deleteUser

DESCRIPTION	Create a group
TYPE	POST
ENDPOINT	/group/create
REQUEST BODY	Format: application/x-www-form-urlencoded Parameters: <ul style="list-style-type: none"> ▪ <i>Groupname</i>
RESPONSE BODY	Returns the representation of the created group or error

Table 19: createGroup

DESCRIPTION	Delete the specified group
TYPE	POST
ENDPOINT	/group/delete
REQUEST BODY	Format: application/x-www-form-urlencoded

	Parameters: <ul style="list-style-type: none"> ▪ <i>Groupname</i>
RESPONSE BODY	Returns error representation on failure

Table 20: deleteGroup

DESCRIPTION	Add the specified user to the specified group
TYPE	POST
ENDPOINT	/group/{id}/addUser
REQUEST PARAM	{id}: group id
REQUEST BODY	Format: application/x-www-form-urlencoded Parameters: <ul style="list-style-type: none"> ▪ <i>User</i>
RESPONSE BODY	Returns the updated representation or error

Table 21: addUserToGroup

DESCRIPTION	Delete the specified user from the specified group
TYPE	POST
ENDPOINT	/group/{id}/delUser
REQUEST PARAM	{id}: group id
REQUEST BODY	Format: application/x-www-form-urlencoded Parameters: <i>User</i>
RESPONSE BODY	Returns the updated representation or error

Table 22: deleteUserFromGroup

DESCRIPTION	Delete the specified policy
TYPE	POST
ENDPOINT	/policy/delete
REQUEST BODY	Format: application/x-www-form-urlencoded <i>Policyname</i>
RESPONSE BODY	Returns error representation on failure

Table 23: deletePolicy

DESCRIPTION	Create a “GroupsIdentity” policy
TYPE	POST
ENDPOINT	/policy/create
REQUEST BODY	Format: application/x-www-form-urlencoded <i>Policyname</i> <i>Group/s</i> <i>Resource/s</i>
RESPONSE BODY	Returns the representation of the created policy or error

Table 24: createGroupIdentityPolicy

DESCRIPTION	Update user info
TYPE	POST
ENDPOINT	/user/{id}
REQUEST PARAM	{id}: user id
REQUEST BODY	Format: application/x-www-form-urlencoded Parameters: <ul style="list-style-type: none"> ▪ <i>Given name</i> ▪ <i>Surname</i> ▪ <i>E-mail</i> ▪ <i>Status</i>
RESPONSE BODY	Returns the updated representation or error

Table 25: updateUser

DESCRIPTION	Update policy info
TYPE	POST
ENDPOINT	/policy/{id}
REQUEST PARAM	{id}: policy id
REQUEST BODY	Format: application/x-www-form-urlencoded Parameters: <ul style="list-style-type: none"> ▪ <i>Name</i> ▪ <i>Description</i> ▪ <i>Active (boolean)</i> ▪ <i>Group(s)</i> ▪ <i>No groups flag</i>

	<ul style="list-style-type: none"> ▪ <i>Resource(s)</i> ▪ <i>No resources flag</i>
RESPONSE BODY	Returns the updated representation or error

Table 26: updatePolicy

DESCRIPTION	Evaluate a user's authorization to access resources
TYPE	POST
ENDPOINT	/evaluate
REQUEST BODY	Format: application/x-www-form-urlencoded Parameters: <ul style="list-style-type: none"> ▪ <i>Token ID</i> ▪ <i>Resource(s)</i>
RESPONSE BODY	Returns a list of allowed or denied operations (GET, POST, PUT, etc.) for each resource

Table 27: evaluate

The security adapter may return three different status codes: 200, 400 or 500, depending on OpenAM errors or internal errors. If the error comes from OpenAM its description is forwarded to the management UI in the response body so that it can display the detailed info to the user.

Possible status codes used are described in the following list.

200 OK	The request was successful and a resource returned, depending on the request. For example, a successful HTTP GET on /users/myUser returns a user profile and status code 200, whereas a successful HTTP DELETE returns {"success","true"} and status code 200.
201 Created	The request succeeded and the resource was created.
400 Bad Request	The request was malformed. Either parameters required by the action were missing, or incorrect data was sent in the payload for the action.
401 Unauthorized	The request requires user authentication. It can be reported if e.g. the required SSO Token value is missing.
403 Forbidden	Access was forbidden during an operation on a resource, for instance if a regular user tries to read the OpenAM administrator profile.
404 Not Found	The specified resource could not be found.
405 Method Not Allowed	The HTTP method is not allowed for the requested resource.

406 Not Acceptable	The request contains parameters that are not acceptable.
409 Conflict	The request would have resulted in a conflict with the current state of the resource.
410 Gone	The requested resource is no longer available, and will not become available again.
415 Unsupported Media Type	The request is in a format not supported by the requested resource for the requested.
500 Internal Server Error	The server encountered an unexpected condition which prevented it from fulfilling the request.
501 Not Implemented	The resource does not support the functionality required to fulfill the request.
503 Service Unavailable	The requested resource was temporarily unavailable. The service may have been disabled.

5.8 SLA Monitoring & Management Functionalities

The following are the screenshots from the SLA Management module. The first one is the screenshot that shows the list of SLA parameters defined for a specific data source.

SLA Parameter Name	Parameter Value	Add New
Uptime availability	90%	Edit Delete
Max. Number of Request	300	Edit Delete
Response time	20.0	Edit Delete
Mean time to restore	100	Edit Delete

SLA Admin can use “Add New” button to define new SLA parameter for the data source. SLA Admin can also use “Edit” or “Delete” buttons to modify a parameter or remove the parameter.

The second screenshot shows what happens when SLA Admin clicks “Add New” button on the previous screenshot. “New SLA Parameter” pop-up will Show, and Admin can define a new SLA parameter with the form.

SLA Management service provides the following APIs to manage the SLA parameters for data sources. For current prototype, it uses Json request/response bodies.

DESCRIPTION	Returns the defined SLA Parameters
TYPE	GET
ENDPOINT	/Sla/{id}
REQUEST PARAM	{id}: data source id
RESPONSE BODY	Returns the info or error description

Table 27:GET SLA parameters

DESCRIPTION	Add / update SLA Parameters to/of a source
TYPE	POST
ENDPOINT	/Sla/{id}
REQUEST PARAM	{id}: data source id {id}: sla param id SLA paramemeters
RESPONSE BODY	Returns error representation on failure

Table 28:Add/update Sla paramenter

DESCRIPTION	Remove SLA Parameters from a source
--------------------	-------------------------------------

TYPE	<i>DELETE</i>
ENDPOINT	<i>/Sla/{id}</i>
REQUEST PARAM	{id}: data source id {id}: sla param id
RESPONSE BODY	Returns error representation on failure

Table 29: Remove SLA parameter from a source

6 OUTLOOK AND CONCLUSIONS

The present document represents the second release of the VITAL management services for smart city environments over multiple IoT platforms, which has significantly extended the first version/release of the deliverable. In particular, the first version specified the main entities to be managed by the VITAL management platform and provided an initial description of its functionalities. Furthermore, it outlined a first prototype implementation on the basis of Istanbul sensor/traffic data, which had been integrated in the Hi REPLY platform. Also, the first version of the deliverable had established a modular approach for the pluggable integration of additional middleware modules, which were in charge of monitoring specific components of the VITAL architecture (such as the CEP and the service discovery modules). Building on the first version of the deliverable, this second version has advanced in the following directions:

- The finalization of the functional specifications of the VITAL management platform, including specifications for IoT platform agnostic management operations in a smart city environment.
- The implementation of middleware modules for managing components of the VITAL architecture such as the CEP, the service discoverer and the security management components. These middleware modules ensured the VITAL components' readiness to become integrated in the management platform i.e. to become managed entities accessible through the VITAL platform.
- The integration of these modules in the VITAL management platform on the basis of the "pluggable" approach outlined in the initial release of the deliverable.
- The delivery of integrated FCAPS functionalities for data/entities stemming from multiple IoT platforms.
-

On the basis of the above-listed advancements, this version of the deliverable has resulted in the first functional prototype implementation of the VITAL management platform, which provides a wide range of functionalities/features including:

- A health map of a VITAL deployment, which provides a geographical representation of sensors and data feeds.

- Entity agnostic management widgets and dashboard, which visualize the VITAL ontologies.
- A set of functionalities for dynamic visual configuration of the main parameters of the platform.
- Security management functionalities, including management of users, groups, roles, permissions, sessions and more.
- Event processing functionalities based on the combination and correlation of data streams from different sources.

Overall, this 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 of the VITAL management platform, 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:

- Implementing new VUAs (Virtualized Unified Access Interfaces) within the dashboards of the platform such as VUAs for monitoring and/or configuring, modules that are not yet fully integrated in the platform. Such modules include the Orchestrator, the CEP and the Service Discovery modules.
- Leveraging the dynamic discovery capabilities provided by the Service Discoverer 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. It will therefore be one of the main exploitable assets of the project, which has already been identified as part of deliverable D7.4.1. In particular, it will serve as a basis for building and commercializing semantically interoperable solutions for smart cities.

7 REFERENCES

[Jin14] Jiong Jin, Gubbi, J. ; Marusic, S. ; Palaniswami, M., « An Information Framework for Creating a Smart City Through Internet of Things», IEEE Internet of Things Journal, Volume:1 Issue:2, 2014.

[Raman98] L. Raman, “OSI Systems and Network Management”, IEEE Communications Magazine, vol. 36, no. 3, IEEE Press, New York, 1998, pp. 46-53.

[Vital-D2.3-2014] J. Soldatos, J. Kaldis, C. Georgoulis et al. “Virtualization Architecture and Technical Specifications”, June 2014.

[Vital-D3.2.1-2014] J. Soldatos, K. Roukounaki et al. Vital Project, Deliverable D3.2.1, “Specification and Implementation of Virtualized Unified Access Interfaces V1”, December 2014.

[RiRu07] L. Richardson, S. Ruby. “RESTful Web Services”, O'Reilly May 2007.

[KuRo14] Rafał Kuć, Marek Rogoziński, “Mastering Elasticsearch”, Packt, October 2014.

[Burke13] Bill Burke, “RESTful Java with JAX-RS 2.0, 2nd Edition: Designing and Developing Distributed Web Services”, O'Reilly, November 2013.