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**

D3.2.1 Specification and Implementation of Virtualized Unified Access Interfaces V1

Document Id:	VITAL-D321-141205-Draft
File Name:	VITAL-D321-141205-Draft.doc
Document reference:	Deliverable 3.2.1
Version :	Draft
Editor :	John Soldatos, Katerina Roukounaki
Organisation :	AIT
Date :	05 / 12 / 2014
Document type:	Deliverable
Security:	PU (Public)

DOCUMENT HISTORY

Rev.	Author(s)	Organisation(s)	Date	Comments
V01	John Kaldis, Katerina Roukounaki, John Soldatos	AIT	25/09/2014	Initial Table of Contents proposal and list of contributors.
	Katerina Roukounaki, John Soldatos	AIT	16/10/2014	Specification of PPI (Section 3).
	Katerina Roukounaki	AIT	08/11/2014	Revision of PPI (Section 3) based on comments from SANTER REPLY.
	John Soldatos	AIT	10/11/2014	Introductory section.
V02	Thomas Geraghty	NUIG	12/11/2014	Comments as well as changes to 3.1.
V03	Andrea Martelli	REPLY	13/11/2014	Comments.
V04	Katerina Roukounaki	AIT	17/11/2014	Changes to PPI specification based on decisions made during the Skype call on 12.11.2014.
	John Soldatos	AIT	19/11/2014	Inputs to Chapter 2; Various Edits
V05	Riccardo Petrolo	INRIA	19/11/2014	Updates to Section 4 on VUAIs for Service Discovery
V06	Fotis Stamatelopoulos, Angelos Lenis	SiLO	20/11/2014	Updates to Section 4 on VUAIs for management access
V07	Katerina Roukounaki	AIT	21/11/2014	Updates to Section 5 (interfaces for data processing)
	John Soldatos	AIT	21/11/2014	Updates to Chapter 2 (relating to architecture)
V08	Umut YILDIRIM	ATOS	24/11/2014	Inputs to Section 4 on VUAIs for CEP
V09	Riccardo Petrolo	INRIA	24/11/2014	Updates to Section 4 on VUAIs for Filtering
V10	John Soldatos	AIT	25/11/2014	Editing and Consolidation
	Katerina Roukounaki	AIT	27/11/2014	Descriptions for data processing web services and various edits
V11	Thomas Geraghty	NUIG	29/11/2014	Inputs to Section 4 on VUAIs for Data Management
V12	John Soldatos	AIT	30/11/2014	Edits and Formatting
V13	Thomas Geraghty	NUIG	02/12/2014	Added examples to data management
V14	John Soldatos	AIT	02/12/2014	Preparation of Version for T/Q Review
V15	Sema Oktug	ITU	05/12/2014	Technical review
V16	Nathalie Mitton	INRIA	05/12/2014	Quality review
V17	John Soldatos	AIT	10/12/2014	Circulated for SB Approval
Draft	Martin Serrano	NUIG	10/15/2014	EC Submitted Draft

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TERMS AND ACRONYMS

API	Application Programming Interface
ARIMA	AutoRegressive Integrated Moving Average
CEP	Complex Event Processing
FCAPS	Fault, Configuration, Accounting, Performance, Security
GSN	Global Sensor Networks
ICO	Internet Connected Object
IoT	Internet-of-Things
JSON-LD	JavaScript Object Notation for Linked Data
OIDC	OpenID Connect
PPI	Platform Provider Interface
SAML	Security Assertion Markup Language
TFL	Transport For London
VUAI	Virtualized Unified Access Interface

1 INTRODUCTION

1.1 Scope

The third work package of the VITAL project (WP3) deals with the specification and implementation of models and interfaces that could enable virtualization of diverse IoT systems. This virtualization is a key to implement IoT platform agnostic functionalities as part of the VITAL platform, thereby enabling development of integrated IoT applications for smart cities i.e. applications leveraging data and services from multiple underlying IoT systems. Key elements of the VITAL virtualization infrastructure include:

- A range of platform-agnostic data models (data schemas, ontologies and databases), along with systems for managing their data elements. At the heart of these data models lies the VITAL ontology, which provides a model for representing data for IoT applications in smart cities regardless of the IoT systems used to capture and/or process these data. The VITAL ontology is already specified as part of deliverable D3.1 of the project.
- A range of virtualized interfaces, which enable platform agnostic access to the IoT services of diverse heterogeneous IoT systems. These interfaces boost VITAL's integrated virtualized application development paradigm, which promotes a «learn-once and use-across-IoT-systems» approach to IoT application development in smart cities.

The purpose of the present deliverable is to provide the specification of the virtualized interfaces of the VITAL platform. In particular, the following types of virtualized interfaces are specified:

- Interfaces to the value-added functionalities of the VITAL platform, such as CEP and Service Discovery functionalities. These functionalities typically use the VITAL ontology as a means for accessing IoT data elements stemming from the underlying platforms.
- Abstract, virtualized interfaces to the functionalities of the underlying IoT systems/platform, which are classified as PPIs (Platform Provider Interfaces). The notion and the role of PPIs have already been introduced as part of the VITAL architecture specification (described in deliverable D2.3).
- Virtualized data processing interfaces over data stemming from the VITAL ontology, which enable the implementation of basic data mining functionalities, notably functionalities that are common in the scope of smart city applications.

All the above interfaces can be classified as VUAIs (Virtualized Unified Access Interfaces). As part of this deliverable, WP3 partners have also commenced the implementation/realization of these interfaces over the four IoT platforms that have been selected (as part of WP2) towards practically showcasing and validating the VITAL platform-agnostic concept. The final implementation of the VUAIs will be realized as part of the next (and final) release of the present deliverable, which may also contain revisions/updates to the current specification of the VUAIs.

1.2 Audience

This deliverable is addressed to the following audiences:

- **IoT applications developers and solution providers**, notably solution providers emphasizing on smart city applications using the IoT paradigm. Application developers and solution providers are expected to be interested into the project's general-purpose interfaces for accessing IoT systems, especially given the fact that the VITAL VUAIs specifications attempt to virtually address any IoT system. Furthermore, VUAIs provide a first approach to implement the very topical target of IoT/BigData convergence, since they include a range of abstract data processing functions over platform agnostic IoT services.
- **IoT researchers,** notably researchers working on abstract data models and service models for IoT applications. To these researchers, VUAIs may serve as a source of information for their research.
- **VITAL developers,** notably individuals engaging in the development of the VITAL added-value functionalities and of the VITAL applications. The former will need to understand the VUAIs in order to make sure that their components/modules support them, while the latter need to gain insights on the VUAIs in order to use them properly as part of their smart city application development tasks.

1.3 Summary

As already outlined, the VITAL virtualized interfaces can be classified in three different types, namely: (a) Abstract interfaces to IoT platforms/systems; (b) Abstract Interfaces to the added-value functionalities of the VITAL platform and (c) Abstract interfaces to the data processing function. This deliverable includes dedicated parts to the specification/description of each of the above types of interfaces. Each dedicated part describes the interfaces and their use in the scope of the VITAL platform i.e. when use by «client» applications accessing the VITAL platform. The specification of each interface (regardless of its type) involves two parts:

- An API (Application Programming Interface) for accessing the functionalities that are wrapped under each abstract interface.
- The data models that drive the exchange of information between the caller (i.e. the invoker of an abstract virtualized interface) and the providers (i.e. the implementers of the abstract virtualized interface) during the invocation of each interface call. The latter data models are prescribed in appropriate formats, notably the JSON-LD format in a way that complies with the VITAL ontology (prescribed in deliverable D3.1).

In addition to providing the interfaces' specification, the deliverable elaborates on their positioning and use in the scope of the VITAL architecture. It therefore identifies the consumers of the interface's functionalities. For instance, PPIs provide a low-latency interface for accessing data streams and capabilities of the underlying IoT platforms. Hence, PPIs are very handy for VITAL application developers, notably developers that are engaging in the integration of real-time or semi real-time applications. As another example, data processing interfaces are handy for solution developers focusing on data-intensive applications within the smart city.

Note that several aspects, such as security are handled within the specification of each interface, rather than horizontally i.e. in a way that transcends all specified interfaces. This is intentional and due to the different scopes and time-scales of operations of the various interface types.

As already outlined, the present version of the deliverable constitutes its first release. The second and final releases may contain updates and revisions to both specifications and implementation issues. An outlook for these revisions is also provided at the end of this document.

1.4 Structure

The deliverable is structured as follows:

- Section 2 following this introductory section, illustrates the scope and purpose of the various types of VUAIs. It also discusses their positioning within the VITAL architecture.
- Section 3 is devoted to the specification of PPIs.
- Section 4 focuses on the specifications of VUAIs for accessing the added-value functionalities of the VITAL platform.
- Section 5 includes the specification of a range of abstract data processing functionalities.
- Section 6 is the final and concluding section of this deliverable, which provides also an outlook for the development of the next (and final) release of the deliverable.

2 VIRTUALIZED UNIFIED ACCESS INTERFACES

2.1 Purpose and Positioning within the VITAL Architecture

The purpose of VUAIs is to facilitate virtualized platform-agnostic access to ICO data and IoT services for smart cities. They are abstract interfaces enabling developers to access data streams and services provided by multiple IoT platforms, without the need to deal with the low-level details of the individual IoT platforms. The concept of VUAIs has already been introduced in deliverable D2.3 as part of the presentation of the VITAL architecture.

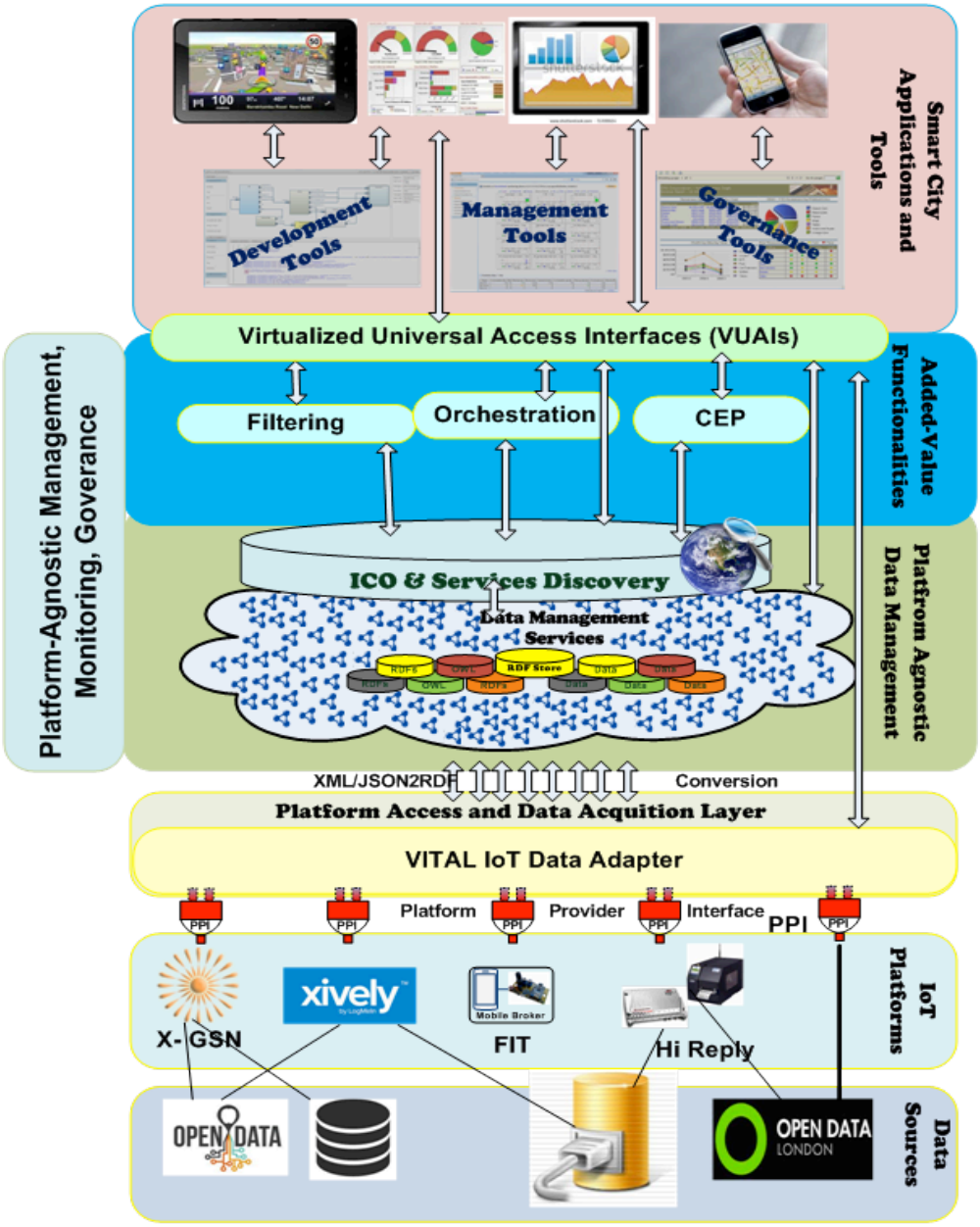

Figure 1: Overview of VITAL Architecture and Positioning of the VUAIs

Figure 1 illustrates a slightly enhanced version of the (high-level) illustration of the VITAL architecture provided in deliverable D2.3. It depicts the role of VUAIs as abstract interfaces residing at the top layer of the VITAL architecture thereby enabling access to added-value data processing and process management functionalities, including CEP functionalities, service discovery functionalities, filtering functionalities and more. As already outlined it also illustrates additional concepts regarding high performance access to platforms and data sources using VUAIs. These concepts are implied, but not detailed in deliverable D2.3. In particular:

- **PPI as a high performance VUAI**: The architecture specifies the possibility of accessing IoT data both through VUAIs accessing the VITAL ontology management services and through direct access to PPIs. The latter option is required in cases where high-performance, low-overhead access to data provided by IoT systems is required (e.g., (near) real-time applications). In this context, the PPI i.e. the platform agnostic interface for accessing IoT systems can be considered as a high-performance VUAI for data access. Overall, the two virtualized platform-agnostic options for accessing IoT system and IoT data through VUAIs are explained in Figure 2.
- **PPI as an interface for accessing individual data sources**: VITAL enables the federation of IoT platforms (such as the four platforms selected in deliverable D2.2. i.e. FIT, X-GSN, Xively.com and Hi REPLY). Nevertheless, it also acknowledges the possibility of exploiting individual (IoT-related) data sources and datasets (e.g., live feeds stemming from TFL Open Data set), based on direct access to their data. Access to individual data sources can be also carried out through a PPI implementation, and more specifically based on a light implementation that does not implement optional functionalities. This concept is also illustrated in Figure 1.

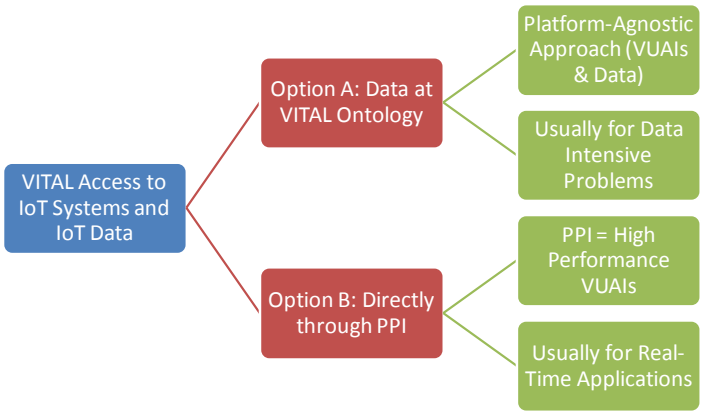

Figure 2: Two main options offered by VITAL Virtualization Layer (VUAIs) regarding IoT Systems and IoT Data access

In addition to virtualized access to IoT data and services, as well as to added-value functionalities (like discovery, filtering and CEP), the high-level VITAL architecture (depicted in the previous figure) prescribes that VUAIs will be accessed by the VITAL development environments and tools. The latter tools are expected to facilitate the development of IoT applications that are based on data processing functionalities (notably data intensive problems). It is therefore envisaged that VUAIs should include interfaces for executing data processing / data mining methods (such as classification and clustering). Such methods (and the respective VUAIs) could be

directly accessed and used by the development tools of the project, towards addressing the needs of the development environment in WP5, which are also driven by the data intensive nature of some of the use cases of the project.

2.2 Types of VUAIs and Functional Scope

The previous paragraph has introduced the concept of VUAIs and their positioning in the VITAL architecture. It has also illustrated the fact that PPIs can be considered VUAIs, as well as the need to provide a data mining toolkit over the VITAL platform-agnostic (multi-platform) datasets. As a result, VUAIs can be classified in three main categories:

- **VUAIs for data access leveraging PPIs**: These are interfaces for platform access, which enable a «learn-once and use-across IoT systems» discipline for accessing IoT data. They facilitate developers and solution providers to access any VITAL compliant platform via a single interface. PPIs can also be used to access individual data sources. As illustrated in the following section, the PPI specification includes a range of both mandatory and optional methods, which correspond to the different spectrum of functionalities that are required to access individual data sources or integrated fully fledged IoT platforms.
- **VUAIs accessing VITAL modules**: These are interfaces for accessing the modules residing at the added-value layer of the VITAL architecture, notably the filtering, CEP and service discovery modules as well as the data from the VITAL ontology management service (comprising the VITAL ontology initially specified in deliverable D3.1.1).
- **VUAIs added-value data processing functionalities**: These are the interfaces enabling data mining functions over data from multiple platforms. They will be used for the implementation of data intensive applications and will be integrated within the VITAL development tools in WP5.

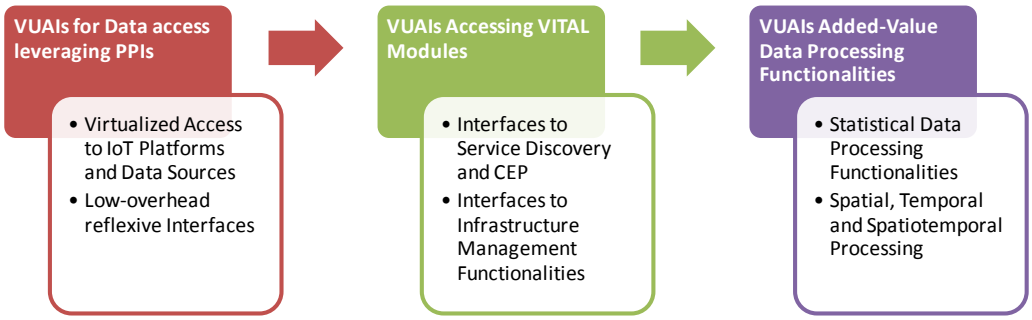

Figure 3: Overview of VUAIs of the VITAL Platform

The three types of VUAIs outlined above are specified in the following paragraphs. The specifications are driven by other VITAL results produced as part of earlier deliverables in particular:

- The VITAL ontology and related data models, which drive the specification of the data elements that are exchanged via the VUAIs. Indeed, we do not specify new data models in this deliverable; rather we rely on the data models prescribed in the first version of the VITAL data modeling deliverable (i.e. D3.1.1). Note that

subsequent versions/releases of this document will be based on the final release of the VITAL ontology (i.e. D3.1.2), which is currently in progress.

- The VITAL architecture introduced in deliverable D2.3. The same deliverable has also illustrated requirements associated with the data elements that should be exchanged over VUAIs.

Overall, the present deliverable is in-line with background results contained in deliverables D3.1.1 and D2.3. The readership is advised to consult these documents for more details on data models and architectural concepts that underpin the VUAIs specifications in the present document.

It should also be noted that the interfaces (VUAIs) specified in this document will provide valuable inputs to other technical activities of the project, notably activities that will produce technical modules that will consume/use the VUAIs. Several such modules (e.g., CEP, discovery, filtering, data access for FCAPS management) are currently being developed in WP4 and WP5 of the project.

3 PLATFORM PROVIDER INTERFACE SPECIFICATION AND PROTOTYPING

3.1 Overview

The Platform Provider Interface (PPI) is a set of RESTful web services, marked as either mandatory or optional, that enable access to IoT systems, as well as to ICOs managed and IoT services provided by those systems. All VITAL compliant IoT systems are expected to implement and expose at least the web services that are designated as mandatory.

3.2 PPI Specification

PPI is defined as a set of RESTful web services that IoT systems offer to be integrated into VITAL should expose and that VITAL can then use in order to retrieve:

- Information about the IoT systems (e.g. their status).
- Information about the IoT services that an IoT system exposes (e.g. how to access them)
- Information about the ICOs that an IoT system manages (e.g. what they observe)
- Observations made by the ICOs that an IoT system manages.

A detailed description of these web services is given in the following paragraphs and is in-line with information contained in earlier deliverable D2.3. Specifically, in D2.3 we have already provided an overview of data and services that the PPI is expected to provide/expose, and we have also classified these data and services as mandatory or optional. Overall, the detailed PPI specifications contained in the following paragraphs build on earlier specifications and requirements specified as part of WP2 of the project.

3.2.1 Access to IoT system and IoT service metadata

This section describes the PPI services for obtaining metadata and lifecycle information about IoT systems. IoT system metadata include also metadata about the IoT services provided by the corresponding IoT system. Lifecycle information about an IoT system is essentially a real-time version of the IoT system metadata.

Access to IoT system metadata is described as mandatory in D2.3, whereas access to real-time information about the status and operation of the underlying IoT system is described as optional. Thus all VITAL compliant IoT systems should implement and expose at least the first of the two web services that are described in the following tables, namely Table 1 and Table 2 respectively.

Table 1: Access to the metadata of an IoT system.

	Get IoT system metadata	
Description	VITAL pulls from an IoT system its metadata.	
URL	`BASE_URL/external/metadata`	
Method	`POST`	
Request headers	`Content-Type`	`application/ld+json` or `application/json`

Request body	**Example**
	`{`
	`  "@context": "http://vital-iot.org/contexts/query.jsonld",`
	`  "type": "vital:iotSystem"`
	`}`	

Response headers	`Content-Type`	`application/ld+json` or `application/json`

Response body	**Example**
	`{`
	`  "@context": "http://vital-iot.org/contexts/system.jsonld",`
	`  "uri": "http://www.example.com",`
	`  "name": "Sample IoT system",`
	`  "description": "This is a VITAL compliant IoT system.",`
	`  "operator": "http://www.example.com",`
	`  "serviceArea": "http://dbpedia.org/page/Camden_Town",`
	`  "status": "vital:Running",`
	`  "providesService":`
	`  [`
	`  {`
	`  "@context":`
	`  "http://vital-iot.org/contexts/service.jsonld",`
	`  "type": "ICOManager",`
	`  "msm:hasOperation":`
	`  [`
	`  {`
	`  "type": "GetMetadata",`
	`  "hrest:hasAddress": "http://www.example.com/ico/metadata",`
	`  "hrest:hasMethod": "hrest:POST"`
	`  }`
	`  ]`
	`  },`
	`  {`
	`  "@context":`
	`  "http://vital-iot.org/contexts/service.jsonld",`
	`  "type": "ObservationManager",`
	`  "msm:hasOperation":`
	`  [`
	`  {`
	`  "type": "GetObservations",`
	`  "hrest:hasAddress":`
	`  "http://www.example.com/observation",`	

	<pre>"hrest:hasMethod": "hrest:POST"
	}
	]
	}
	]
	}</pre>

Mandatory	Yes
Notes	• The request body can be used to filter the metadata to return. • The context of the request body is the context for queries that has not yet been defined. Thus, the request body should have the exact format shown in the example. • The context of the response body is the context for systems described in Section 5.1 of D3.1.1.

Table 2: Access to lifecycle information about an IoT system.

	Get IoT system lifecycle information	
Description	VITAL pulls from an IoT system its lifecycle information.	
URL	`BASE_URL/external/lifecycle_information`	
Method	`POST`	
Request headers	`Content-Type`	`application/ld+json` or `application/json`

Request body	**Example**
	<pre>{
	"@context": "http://vital-iot.org/contexts/query.jsonld",
	"type": "vital:iotSystem"
	}</pre>	

Response headers	`Content-Type`	`application/ld+json` or `application/json`

Output	**Example**
	<pre>{
	"@context": "http://vital-iot.org/contexts/system.jsonld",
	"uri": "http://www.example.com",
	"status": "vital:Running"
	}</pre>	

Mandatory	No	
Notes	• The request body can be used to filter the lifecycle information to return. • The context of the request body is the context for queries that has not yet been defined. Thus, the request body should have the exact format shown in the example. • The context of the response body is the context for systems described in Section 5.1 of D3.1.1.	

3.2.2 Access to ICO metadata

IoT systems should expose to VITAL information about the ICOs they manage (e.g. their type or their location). In order to do that, VITAL compliant IoT systems are expected to provide an IoT service of type **ICOManager** with (at least) an operation of type **GetMetadata**. Table 3 illustrates an example for the description of such an IoT service.

Table 3: ICOManager IoT service description.

```
{
  "@context": "http://vital-iot.org/contexts/service.jsonld",
  "type": "ICOManager",
  "msm:hasOperation":
  [
```

```
    {
      "type": "GetMetadata",
      "hrest:hasAddress": "http://www.example.com/ico/metadata",
      "hrest:hasMethod": "hrest:POST"
    }
  ]
}
```

As described in Table 4, the `GetMetadata` operation is expected to return metadata about the managed ICOs that satisfy certain criteria in JSON-LD format based on the JSON-LD context for sensors described in deliverable D3.1.1 of the project (see Section 3.3), where data models used in VITAL are specified.

Table 4: Access to the metadata of managed ICOs.

	Get ICO metadata	
Description	VITAL pulls from an IoT system metadata about the managed ICOs.	
Method	`POST`	
Request headers	`Content-Type`	`application/ld+json` or `application/json`

Request body	**Example**
	
	`{`
	`  "@context": "http://vital-iot.org/contexts/query.jsonld",`
	`  "icos":`
	`  [`
	`  "http://www.example.com/ico/123/"`
	`  ]`
	`}`	

Response headers	`Content-Type`	`application/ld+json` or `application/json`

Response body	**Example**
	
	`[`
	`  {`
	`  "@context": "http://vital-iot.org/contexts/sensor.jsonld",`
	`  "uri": "http://www.example.com/ico/123",`
	`  "name": "A sensor.",`
	`  "type": "VitalSensor",`
	`  "description": "A sensor.",`
	`  "status": "vital:Running",`
	`  "hasLastKnownLocation":`
	`  {`
	`  "type": "geo:Point",`
	`  "geo:lat": 53.2719,`
	`  "geo:long": -9.0849`
	`  },`
	`  "hasMovementPattern":`
	`  {`
	`  "type": "Stationary",`
	`  "hasPredictedSpeed":`
	`  {`
	`  "value": "3.1",`
	`  "qudt:unit": "qudt:KilometerPerHour"`
	`  },`
	`  "hasPredictedDirection":`
	`  {`
	`  "type": "NormalVector",`
	`  "geo:lat": "53.2719",`
	`  "geo:long": "-9.0489"`
	`  }`
	`  },`
	`  "hasLocalizer":`
	`  {`
	`  "type": "GpsService",`
	`  "msm:hasOperation":`	

<table>
<tr><td></td><td>

<pre>
      {
        "type": "GetLocation",
        "hrest:hasMethod": "hrest:GET",
        "hrest:hasAddress":
          "http://www.example.com/ico/123/location/"
      }
    },
    "hasNetworkConnection":
    {
      "hasStability":
      {
        "type": "Continuous"
      },
      "hasNetworkSupport":
      {
        "net:connectedNetworks":
        {
          "type": "net:WiredNetwork"
        }
      }
    },
    "deviceHardware":
    {
      "hard:status": "hard:HardwareStatus_ON",
      "hard:builtInMemory":
      {
        "size": 131072
      },
      "hard:cpu":
      {
        "type": "hard:CPU",
        "maxCpuFrequency": 10
      }
    },
    "ssn:observes":
    [
      {
        "type": "http://lsm.deri.ie/OpenIot/Temperature",
        "uri": "http://www.example.com/ico/123/temperature"
      }
    ]
  }
]
</pre>

</td></tr>
<tr><td>Notes</td><td>

- The request body can be used to filter the ICOs to return metadata for.
- The context of the request body is the context for queries that has not yet been defined. Thus, the request body has the format shown in the example. **icos** field is optional and determines the ICOs to return metadata for. If icos is omitted, then metadata are returned for all managed ICOs.
- The context of the response is the context for sensors described in Section 3.3 of D3.1.1.

</td></tr>
</table>

Since access to ICO metadata is marked as mandatory in Deliverable D2.3, all VITAL compliant IoT systems are required to implement and expose a RESTful web service that acts as an implementation for the `GetMetadata` operation of the ICOManager IoT service and therefore satisfies all requirements outlined in Table 4.

3.2.3 Access to ICO observations

It is prescribed that VITAL can use both a pull- and a push-based mechanisms in order to obtain observations made by an ICO. An IoT system can support one or both of these mechanisms by providing an IoT service of type **ObservationManager** with the necessary operations.

Since access to the ICO data of an IoT system is mandatory, all VITAL compliant IoT systems should support at least one of these two mechanisms. This essentially means that IoT providers should implement and expose one RESTful web service for each operation that is required by the mechanism (or the mechanisms) they want to support, as described in the following paragraphs.

3.2.3.1 Pull-based mechanism

In order to support the pull-based mechanism (i.e. the VITAL pulls observations from the IoT system), the **ObservationManager** service should have (at least) an operation of type **GetObservations**. An example for the description of an IoT service of type ObservationManager that supports the pull-based mechanism is shown in Table 5.

Table 5: Description for an ObservationManager IoT service that supports the pull-based mechanism.

```
{
  "@context": "http://vital-iot.org/contexts/service.jsonld",
  "type": "ObservationManager",
  "msm:hasOperation":
  [
    {
      "type": "GetObservations",
      "hrest:hasAddress": "http://www.example.com/observation",
      "hrest:hasMethod": "hrest:POST"
    }
  ]
}
```

As described in Table 6, the `GetObservations` operation is expected to return observations that satisfy certain criteria in JSON-LD format based on the JSON-LD context for measurements described in deliverable D3.1.1 (see Section 3.4).

Table 6: Access to observations.

	Get observations	
Description	VITAL pulls from an IoT system observations.	
Method	`POST`	
Request headers	`Content-Type`	`application/ld+json` or `application/json`

Request body	**Example**
	
	`{`
	`  "@context": "http://vital-iot.org/contexts/query.jsonld",`
	`  "ico": "http://www.example.com/ico/123",`
	`  "property": "http://lsm.deri.ie/OpenIot/Temperature",`
	`  "from": "2014-11-17T09:00:00+02:00",`
	`  "to": "2014-11-17T11:00:00+02:00"`
	`}`	

Response headers	`Content-Type`	`application/ld+json` or `application/json`

Response body	**Example**
	
	`[`
	`  {`
	`  "@context": "http://vital-iot.org/contexts/measurement.jsonld",`
	`  "uri": "http://www.example.com/ico/123/observation/1",`
	`  "type": "ssn:Observation",`	

	<pre>"ssn:observationProperty":
	{
	"type": "http://lsm.deri.ie/OpenIoT/Temperature"
	},
	"ssn:observationResultTime":
	{
	"inXSDDateTime": "2014-08-20T16:47:32+01:00"
	},
	"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": "21.0",
	"qudt:unit": "qudt:DegreeCelsius"
	}
	}
	}
	]</pre>

Notes	• The request body can be used to filter the observations to return. • The context of the request is the context for queries that has not yet been defined. Thus, the request body has the format shown in the example. **ico** and **property** fields are mandatory and determine the ICO and the property, respectively, to return observations for. **from** and **to** determine the time interval, when the observations to return were taken. Both to and from are optional. If to is omitted, then all observations taken after from are returned. If both from and to are omitted, then the last observation taken from the specified ICO for the specified property is returned. • The context of the response is the context for sensors described in Section 3.3 of D3.1.1.

3.2.3.2 Push-based mechanism

In order to support the push-based mechanism (i.e. the IoT system pushes observations to VITAL), the **ObservationManager** service should have (at least) the following operations:

- an operation of type **SubscribeToObservationStream** that creates a subscription to a specific stream of observations
- an operation of type **UnsubscribeFromObservationStream** that cancels a subscription with a specific ID.

An example for the description of an IoT service of type ObservationManager that supports the push-based mechanism is shown in Table 7.

Table 7: Description for an ObservationManager IoT service that supports the push-based mechanism.

```
{
  "@context": "http://vital-iot.org/contexts/service.jsonld",
  "type": "ObservationManager",
  "msm:hasOperation":
  [
    {
      "type": "SubscribeToObservationStream",
      "hrest:hasAddress": "http://www.example.com/observation/stream/subscribe",
      "hrest:hasMethod": "hrest:POST"
    },
    {
      "type": "UnsubscribeFromObservationStream",
      "hrest:hasAddress": "http://www.example.com/observation/stream/unsubscribe",
      "hrest:hasMethod": "hrest:POST"
    }
  ]
}
```

As described in Table 8, the `SubscribeToObservationStream` operation is expected to return a subscription ID. As a result of that operation, VITAL periodically receives at the specified URL observations that match the specified criteria (i.e. observations that were obtained by the specified ICO for the specified property). The URL essentially indicates another RESTful web service - one that is implemented and exposed by the VITAL platform - that supports the HTTP POST method, and expects a set of observations in JSON-LD format based on the JSON-LD context for measurements described in Section 3.4 of D3.1.1.

Table 8: Subscribe to observation stream.

	Subscribe to observation stream	
Description	VITAL subscribes to an observation stream.	
Method	`POST`	
Request headers	`Content-Type`	`application/ld+json` or `application/json`

Request body	**Example**
	`{`
	`  "@context": "http://vital-iot.org/contexts/query.jsonld",`
	`  "ico": "http://www.example.com/ico/123",`
	`  "property": "http://lsm.deri.ie/OpenIot/Temperature",`
	`  "url": "http://www.example.com/vital/observation/push"`
	`}`	

Response headers	`Content-Type`	`application/json`

Response body	**Example**
	`{`
	`  "subscriptionId" : "d670460b4b4aece5915caf5c68d12f560a9fe3e4"`
	`}`	

Notes	• The request body can be used to filter the observations to push, as well as to specify the URL where the observations are to be pushed. • The context of the request is the context for queries that has not yet been defined. Thus, the request body has the format shown in the example. all fields are mandatory. **ico** and **property** fields determine the ICO and the property, respectively, to push observations for. **url** determines the URL where observations are to be pushed.	

As described in Table 9, the `UnsubscribeFromObservationStream` operation gives back no response, and results in the cancellation of the subscription with the specified ID.

Table 9: Unsubscribe from observation stream.

	Unsubscribe from observation stream	
Description	VITAL unsubscribes from an observation stream.	
Method	`POST`	
Request headers	`Content-Type`	`application/json`

Request body	**Example**
	`{`
	`  "subscriptionId": "d670460b4b4aece5915caf5c68d12f560a9fe3e4"`
	`}`	

Notes	• The request body is used to specify the ID of the subscription to cancel.	

3.3 Middleware Infrastructure for Accessing the PPIs from VITAL

VITAL provides a web-accessible user interface that can be used to register and de-register IoT systems. As part of the registration and deregistration processes, the IoT system provider is expected to provide:

- The **URI** of the IoT system, which acts as its unique identifier.
- The **base URL**, through which its PPI implementation is accessible
- Any information that is necessary for the **authenticated** and **authorized** access of the PPI implementation (e.g. the authentication scheme to use).

As the SLA modeling process is still on-going, authenticated access to the PPI implementation provided by an IoT system is at the moment considered to be system-specific. This essentially implies that each IoT system can choose to provide authenticated and authorized access to its PPI implementation in a different way. The different authentication schemes that are currently under consideration are:

- **OpenID Connect** (OIDC)[1], which is an identity layer on top of the OAuth 2.0 protocol
- **Security Assertion Markup Language** (SAML)[2], which is an XML-based framework for communicating user authentication, entitlement and attribute information.

This process is likely to be subsumed as part of the finalization of the SLA modeling and management implementation.

Based on the PPI services outlined above, VITAL is expected to periodically pull information from all registered IoT systems. More specifically, VITAL pulls the following information (using the respective PPI primitives):

- Information and metadata about the IoT system.
- Lifecycle information about the IoT system.

[1] http://openid.net/connect

[2] https://www.oasis-open.org/committees/tc_home.php

- Information and metadata about the ICOs that the IoT system manages.
- Information and metadata about the IoT services that the IoT system provides.
- Observations made by ICOs that are managed by the IoT system.

The frequency with which each one of the above pieces of information is pulled from an IoT system is configurable and might vary from system to system, depending also on business requirements of the targeted/supported smart cities applications.

Alternatively to the above pull-based mechanism, registered IoT systems can also push observations to VITAL . VITAL stores all data and metadata, which it pulls from registered IoT systems or that IoT systems push to it, into a cloud database, so that VITAL can reply to a request without having to collect the necessary data from one or more IoT platforms at the time the request is made.

4 VIRTUALIZED ACCESS TO VITAL MODULES

4.1 Interfaces to Data Management Service

The Data Management service is responsible for allowing access to ICO metadata as well as historical measurement data. This is provided through a RESTful interface that uses GET request as well POST requests with attached JSON or JSON-LD bodies that allow for simple filtering of requested data by using key-value pairs or attached SPARQL queries for more complex requests.

The REST API offers methods to retrieve both metadata and measurements from connected ICOs. After some discussion the decision to use POST requests with attached JSON in the body of the request over the more common GET using URL encoded parameters was made. The primary reasons for this were:

- By using JSON-LD the request can utilize VITAL's linked data capabilities if required, which enabled more powerful contextual information to be offered to the management service.
- For complex requests SPARQL queries can be attached in the body of the JSON-LD document. If the key-value pair used for SPARQL queries is present in the body other key-value pairs are ignored and only the query is used for filtering. Attaching a query via URL encoding would lead to large unwieldy URLs.
- Because of the two aforementioned reasons JSON-LD via the body must be supported. Supporting GET request via URL encoded parameters and POST requests with JSON-LD bodies for every single request would lead to a large amount of code duplication and increase chances of human error, leading to a more complex, less secure system. For a system like VITAL any design decisions which may impact security must be carefully considered.

The REST interface supports both the application/json and application/ld_json content types. It decides if the body should be treated as linked data by checking for the existence of the @content key that's value resolves to a valid JSON-LD schema.

	Get all measurements from child above provided temperature in date range	
Method	`POST`	
URL	`BASE_URL/dms/children/g57skd/measurements`	
Request headers	`Content-Type`	`application/ld+json` or `application/json`

Request body	**Example**
	`{`
	`  "@context": "http://vital-iot.org/contexts/query.jsonld",`
	`  "temperature": {`
	`  "gt":60.0`
	`  },`
	`  "time":{`
	`  "start":"2014-11-10T00:00:00+01:00",`
	`  "end":"2014-11-11T10:00:00+00:00"`
	`  }`
	`}`	

Response headers	`Content-Type`	`application/json`

Response body	**Example**
	`[`
	`  {`
	`  "url":"http://vital.example.com/ico/g57skd/measurements/0f7s3j",`
	`  "time":"2014-11-10T06:45:03+01:00",`
	`  "temperture":"62.5"`
	`  },`
	`  {`
	`  "url":"http://vital.example.com/ico/g57skd/measurements/jd921s",`
	`  "time":"2014-11-10T19:12:54+01:00",`
	`  "temperture":"62.5"`
	`  },`
	`  {`
	`  "url":"http://vital.example.com/ico/g57skd/measurements/8sl33f",`
	`  "time":"2014-11-10T06:45:09+01:00",`
	`  "temperture":"61.1"`
	`  }`
	`]`	

	Get specific measurement from child	
Method	`GET`	
URL	`BASE_URL/dms/children/h9sh33/measurements/0f7s3j`	
Request headers	`Content-Type`	`application/ld+json` or `application/json`
Input	-	
Response headers	`Content-Type`	`application/json`

Response body	**Example**
	`{`
	`  "url":"http://vital.example.com/ico/d72h9d/measurements/0f7s3j",`
	`  "time":"2014-11-10T06:45:03+01:00",`
	`  "temperture":"62.5"`
	`}`	

4.2 Interfaces to Service Discovery Module

The Service Discovery module is in charge of discovering ICOs, data streams, and services horizontally integrated in the VITAL platform. It operates on the IoT

resources that are stored by the Data Management Service and are requested by other modules in the VITAL Architecture (e.g. CEP, Filtering and Orchestration).

The Service Discovery module is accessible through a RESTful web service designed and implemented as part of WP4 and based on JavaEE technologies.

Table 10 briefs the description of this module, presenting a list of the operations that can be executed and how they can be accessed.

In the first version, the module exposes general operations, as the `ConnDMS` operation used to retrieve information on the status of the connection with the Data Management Service (DMS), and other operations that focus on the ICO Discovery as:

- the `NICOs` operation that is used to provide the number of ICOs that are available in the DMS.
- the `GetICOs` operation accepts input parameters like `latitude`, `longitude`, `radius`, `observedProperty` and returns the available ICOs in the area.
- the `GetICO` operation gives information regarding a specific ICO that is identified by the `URI`.

Table 10: Service Discovery description

```
{
  "@context":"http://vital-iot.org/contexts/service.jsonld",
  "type":"ServiceDiscovery",
  "description":"This is the VITAL Service Discovery module.",
  "status":"running",
  "msm:hasOperation":
  [
    {
      "type":"ConnDMS",
      "hrest:hasAddress":"BASE_URL/discoverer/ConnDMS",
      "hrest:hasMethod":"hrest:GET"
    },
    {
      "type":"nICOs",
      "hrest:hasAddress":"BASE_URL/discoverer/nICOs",
      "hrest:hasMethod":"hrest:GET"
    },
    {
      "type":"getICOs",
      "hrest:hasAddress":"BASE_URL/discoverer/getICOs",
      "hrest:hasMethod":"hrest:GET",
      "hrest:hasParameters":"double:longitude, double:latitude, double:radius,
string:ObservationProperty"
    },
    {
      "type":"getICO",
      "hrest:hasAddress":"BASE_URL/discoverer/getICO",
      "hrest:hasMethod":"hrest:GET",
      "hrest:hasParameters":"string:uri"
    },
    {
      "type":"getICOsMobility",
      "hrest:hasAddress":"BASE_URL/discoverer/getICOsMobility",
      "hrest:hasMethod":"hrest:GET",
      "hrest:hasParameters":"string:mobilityType"
    },
    {
```



```
      "type":"getICOsConnectionStability",
      "hrest:hasAddress":"BASE_URL/discoverer/getICOsConnectionStability",
      "hrest:hasMethod":"hrest:GET",
      "hrest:hasParameters":"string:stabilityType"
    },
    {
      "type":"getICOsLocalizerService",
      "hrest:hasAddress":"BASE_URL/discoverer/getICOsLocalizerService",
      "hrest:hasMethod":"hrest:GET"
    }
  ]
}
```

The above table (Table 10) provides the description of the services exposed by the Service Discovery. The following tables illustrate additional interfaces for the use of these services.

Table 11: Connection to DMS

	Connection to DMS
Description	It gives information about the status of the connection with the DMS.
URL	BASE_URL/discoverer/ConnDMS
Method	GET
Input	-

Output	{
	"@context": "http://vital-iot.org/contexts/service.jsonld",
	"type": "ServiceDiscovery/ConnDMS",
	"hrest:hasAddress": "BASE_URL/ConnDMS",
	"hrest:hasMethod": "hrest:GET",
	"hrest:status": "OFF "
	}

Table 12: Number of ICOs available

	Number of ICOs available
Description	This interface gives information about the number of ICOs available in the DMS
URL	BASE_URL/discoverer/nICOs
Method	GET
Input	-
Output	Number of ICOs available

Table 13: Get ICOs

	Get ICOs
Description	The Discoverer queries the DMS in order to get ICOs available in a defined area. It is also possible to filter the ICOs in function of the observed properties.
URL	BASE_URL/discoverer/getICOs
Method	GET

Input	Filtering parameters.
	
	Example format:
	{
	double:"latitude","longitude,"radius", string:"observedProperty"
	}

Output	List of ICOs with capabilities metadata in JSON-LD format

Table 14: Get ICO

	Get ICO
Description	The Discoverer queries the DMS in order to get 1 specified ICO characterized by its URI
URL	BASE_URL/discoverer/getICO
Method	GET

Input	URI.
	
	Example format:
	{
	string:"URI"
	}

Output	Capabilities and information about ICO in JSON-LD format

All the operations return information formatted in JSON, according to the data models and ontologies illustrated in deliverable D3.1.1.

In the next version, the Service Discovery module will offer services that would also take into account mobile ICOs, and will extend the discovery to services and data streams.

Furthermore, details about the interface to Service Discovery module are available in deliverable D4.1.1 of the VITAL project.

4.3 Interface to Filtering Module

The Filtering module offers mechanisms that enable the filtering of data and metadata stemming from different sources. It uses the Service Discovery in order to retrieve the data to which it applies filtering mechanisms.

Table 1 summarizes a preliminary description of the filtering. It presents `ConnSD`, an interface that checks the status of the connection with the Service Discovery.

The second interface is called `DataElaboration`; it queries the Discoverer in order to get data from a certain area, and then makes elaboration on it. The type of elaboration is defined trough a mathematical operator (`string:operation`) i.e., AVG, MIN, MAX and trough a value (`double:value`).

Table 15: Description of Interfaces to Filtering Functionality

```
{
  "@context": "http://vital-iot.org/contexts/service.jsonld",
  "type": "Filtering",
  "description": "This is the VITAL Filtering module."
  "status": "running"
  "msm:hasOperation":
  [
    {
      "type": "ConnSD",
      "hrest:hasAddress": "BASE_URL/ConnSD",
      "hrest:hasMethod": "hrest:GET"
    },
```



```
    {
      "type": "DataElaboration",
      "hrest:hasAddress": "BASE_URL/DataElaboration",
      "hrest:hasMethod": "hrest:GET"
      "hrest:hasParameters":"double:longitude,      double:latitude,
double:radius,    string:ObservationProperty",    string:operation,
double:value"
    }
  ]
}
```

Since the filtering functionalities have not yet been implemented yet (i.e. the respective work is still ongoing), additional details will be available in the Deliverable 4.2.1 of the VITAL Project.

4.4 Interfaces to CEP Module

The Complex Event Processing Module is an added-value mechanism to VITAL Project. It is responsible for managing event processing over observation streams.

The Complex Event Processing is accessible through a RESTful web service researched and implemented as part of WP4 and based on JavaEE technologies.

In the first version, the module exposes interfaces for managing listener, event & rule specifications. Below are the list of exposed interfaces and how they can be accessed.

4.4.1 Coordinator:

4.4.1.1 GetLogFileName

	Get Log filename
Description	Gets log file name from CEP
URL	BASE_URL/`solcep/coordinator/logger/name`
Method	GET
Input	-
Output	**Example** `{ "file": "solcep.log" }`

4.4.1.2 SetLogFileName

	Get Log filename
Description	Sets log file name on CEP
URL	BASE_URL/`solcep/coordinator/logger/name` /logfile.log
Method	POST
Input	-
Output	**Example** `{`

	<pre>{
	  "data": {
	  "success": "true",
	  "file": " logfile.log"
	  }
	}</pre>

4.4.1.3 GetLogLevel

	Get Log Level
Description	Gets the log level value from logger in CEP
URL	BASE_URL/solcep/coordinator/logger/level
Method	GET
Input	-

Output	**Example**
	<pre>{
	  "data": { "logLevel": "9" }
	}</pre>

4.4.1.4 SetLogLevel

	Set Log Level
Description	Sets the log level for logger in CEP
URL	BASE_URL/solcep/coordinator/logger/level/6
Method	POST
Input	-

Output	**Example**
	<pre>{
	  "data": {
	  "success": "true",
	  "level": "6"
	  }
	}</pre>

4.4.2 Event Collector

4.4.2.1 GetEventCollectors:

	Get Event Collectors
Description	Gets the list of Event Collectors
URL	BASE_URL/solcep/eventCollectors
Method	GET
Input	-

Output	**Example**
	<pre>{
	  "125": {
	  "time": "2014-11-10T10:07:02+01:00",
	  "temperature": 18.4
	  }
	}</pre>

4.4.2.2 GetEventCollector:

	Get Event Collector
Description	Gets the details about a specific (eventCollector-id) event collector
URL	BASE_URL/solcep/eventCollector/{eventCollector-id}
Method	GET
Input	-

Output	**Example**
	
	{
	"125": {
	"time": "2014-11-10T10:07:02+01:00",
	"temperature": 18.4
	}
	}

4.4.2.3 CreateEventCollector

	Create Event Collector
Description	Create a new event collector on CEP
URL	BASE_URL/solcep/eventCollector/{eventCollector-id}
Method	PUT
Input	-

Output	**Example**
	
	{
	"data": {
	"success": "true",
	"eventCollector": "6"
	}
	}

4.4.2.4 DeleteEventCollector

	Delete Event Collector
Description	Deletes the specified Event collector on CEP
URL	BASE_URL/solcep/eventCollector/{eventCollector-id}
Method	DELETE
Input	-

Output	**Example**
	
	{
	"data": {
	"success": "true"
	}
	}

4.4.2.5 GetListeners

	Get Listeners
Description	Gets the list of available listeners of a specific event collector
URL	BASE_URL/solcep/eventCollector/{eventCollector-id}/listeners
Method	GET
Input	-

Output	Example


```
{
  "listeners": {
    "listener": {
      "-id": "udpListener",
      "enabled": "1",
      "plugin": {
        "name": "libUDPListenerPlugin.so.1.0",
        "comment": "Atos provided UDP listener plugin",
        "params": {
          "param": [
            {
              "-id": "port",
              "#text": "29201"
            },
            {
              "-id": "maxPacketSize",
              "#text": "512"
            }
          ]
        }
      },
      "decoderRef": "simpleEventFormatDecoder"
    }
  }
}
```


4.4.2.6 Create or Update Listener

	Create or Update Listener
Description	Creates or updates a listener in a specified Event Collector
URL	BASE_URL`/solcep/eventCollector/{eventCollector-id}/listener/{listener-id}`
Method	PUT
Input	-
Output	**Example**


```
{
  "data": {
    "success": "true",
    "listener": "udpListener"
  }
}
```


4.4.2.7 Delete Listener

	Delete Listener
Description	Deletes a specified listener in an event collector
URL	BASE_URL`/solcep/eventCollector/{eventCollector-id}/listener/{listener-id}`
Method	DELETE
Input	-
Output	**Example**


```
{
  "data": {
    "success": "true"
  }
}
```


4.4.2.8 Assign Decoder To Listener

	Assign Decoder To Listener
Description	Assigns decoder to specified listener
URL	BASE_URL/solcep/eventCollector/{eventCollector-id}/listeners/{listener-id}/decoderRef/{decoder-id}
Method	POST
Input	-

Output	**Example**
	
	{
	"data": {
	"success": "true"
	}
	}

4.4.2.9 Disassociate Decoder From Listener

	Disassociate Decoder from Listener
Description	Disassociates a decoder from specified listener
URL	BASE_URL/solcep/eventCollector/{eventCollector-id}/listeners/{listener-id}/decoderRef/{decoder-id}
Method	DELETE
Input	-

Output	**Example**
	
	{
	"data": {
	"success": "true"
	}
	}

4.4.2.10 Get Decoders

	Get Decoders
Description	Gets the list of decoders of a specified event collector
URL	BASE_URL/solcep/eventCollector/{eventCollector-id}/decoders
Method	GET
Input	-

Output	**Example**
	
	{
	"decoders": {
	"decoder": {
	"-id": "simpleEventFormatDecoder",
	"plugin": {
	"name": "libSimpleDecoderPlugin.so.1.0",
	"comment": "Atos provided Simple Event decoder",
	"enabled": "1"
	}
	}
	}
	}

4.4.2.11 Create or Update Decoder

	Create or Update Decoder
Description	Creates or updates a decoder in a specified event collector
URL	BASE_URL/solcep/eventCollector/{eventCollector-id}/decoder/{decoder-id}
Method	PUT
Input	-

Output	**Example**
	
	{
	"data": {
	"success": "true",
	"decoder": "simpleEventFormatDecoder"
	}
	}

4.4.2.12 Delete Decoder

	Delete Decoder
Description	Deletes a specified decoder in event collector
URL	BASE_URL/solcep/eventCollector/{eventCollector-id}/decoder/{decoder-id}
Method	DELETE
Input	-

Output	**Example**
	
	{
	"data": {
	"success": "true"
	}
	}

4.4.3 ComplexEventDetector

4.4.3.1 Get Detection Specifications

	Get Detection Specifications
Description	Gets the detection specifications from CEP
URL	BASE_URL/solcep/complexEventDetector/detectionSpecification
Method	GET
Input	-

Output	**Example**
	
	{
	"detectionSpecifications": {
	"detectionSpecification": {
	"-id": "detect",
	"enabled": "1",
	"port": "5555",
	"acceptedListeners": {
	"listenerRef": [
	{ "-id": "udpListener" },
	{

```
            "-id": "remote",
            "params": {
              "param": [
                {
                  "-id": "port",
                  "#text": "50100"
                },
                {
                  "-id": "address",
                  "#text": "127.0.0.1"
                }
              ]
            }
          }
        ]
      },
      "requiredEmitters": {
        "emitterRef": { "-id": "udpEmitter" }
      }
    }
  }
}
```


4.4.3.2 Create or Update Detection Specification

	Create or Update Detection Specification
Description	Creates or updates a detection specification
URL	BASE_URL`/solcep/complexEventDetector/detectionSpecifications/{dectspec-id}`
Method	PUT
Input	-
Output	**Example** `{ "data": { "success": "true" } }`

4.4.3.3 Delete Detection Specification

	Delete Detection Specification
Description	Deletes a specified decoder in event collector
URL	BASE_URL`/solcep/complexEventDetector/detectionSpecifications/{dectspec-id}`
Method	DELETE
Input	-
Output	**Example** `{ "data": { "success": "true" } }`

4.4.3.4 Get Accepted Listeners

	Get Accepted Listeners
Description	Gets the list of accepted listeners of a specified detection specification
URL	BASE_URL/solcep/complexEventDetector/ detectionSpecifications/ {dectspec-id}/acceptedListeners
Method	GET
Input	-

Output	**Example**
	
	{
	"acceptedListeners": {
	"listenerRef": [
	{ "-id": "udpListener" },
	{
	"-id": "remote",
	"params": {
	"param": [
	{
	"-id": "port",
	"#text": "50100"
	},
	{
	"-id": "address",
	"#text": "127.0.0.1"
	}
	]
	}
	}
	]
	}
	}

4.4.3.5 Add Accepted Listener

	Add Accepted Listener
Description	Adds a listener to Accepted Listeners of a detection specification
URL	BASE_URL/solcep/complexEventDetector/detectionSpecifications/{dectspec-id} acceptedListeners/{listener-id}
Method	POST
Input	-

Output	**Example**
	
	{
	"data": {
	"success": "true"
	}
	}

4.4.3.6 Delete Accepted Listener

	Delete Accepted Listener
Description	Deletes a specified listener from detection specification
URL	BASE_URL/solcep/ complexEventDetector/ detectionSpecifications/{dectspec id}/acceptedListeners/{listener id}

Method	DELETE
Input	-

Output	**Example**
	
	{
	"data": {
	"success": "true"
	}
	}

4.4.3.7 Get Required Emitters

	Get Required Emitters
Description	Gets the list of required emitters of a detection specification
URL	BASE_URL/solcep/ complexEventDetector/detectionSpecifications/ detectionSpecification/{dectspec id}/requiredEmitters
Method	GET
Input	-

Output	**Example**
	
	{
	"requiredEmitters": {
	"emitterRef": { "-id": "udpEmitter" }
	}
	}

4.4.3.8 Add Required Emitter

	Add Required Emitter
Description	Adds an emitter to required emitter list of a detection specification
URL	BASE_URL/solcep/ complexEventDetector/detectionSpecifications/ detectionSpecification/requiredEmitters/{emitter id}
Method	POST
Input	-

Output	**Example**
	
	{
	"data": {
	"success": "true"
	}
	}

4.4.3.9 Delete Required Emitter

	Delete Required Emitter
Description	Deletes an emitter from required emitters list of a detection specification
URL	BASE_URL/solcep/ complexEventDetector/ detectionSpecifications/{dectspec id}/ requiredEmitters/{emitter id}
Method	DELETE
Input	-
Output	**Example**

	{
	"data": {
	"success": "true"
	}
	}

4.4.4 Complex Event Publisher

4.4.4.1 GetComplexEvent plublishers:

	Get ComplexEvent Publishers
Description	Gets a list of complex event publishers from CEP
URL	BASE_URL/solcep/ complexEventPublishers
Method	GET
Input	-

Output	**Example**
	{
	"eventPublishers": {
	"eventPublisher": {
	"-id": "publish",
	"enabled": "1",
	"port": "5555",
	"encoders": {
	"encoder": {
	"-id": "simpleEncoder",
	"enabled": "1",
	"plugin": {
	"name": "libSimpleEncoderPlugin.so.1.0",
	"comment": "Atos provided encoder plugin"
	}
	}
	},
	"emitters": {
	"emitter": {
	"-id": "udpEmitter",
	"enabled": "1",
	"plugin": {
	"name": "libUDPEmitterPlugin.so.1.0",
	"comment": "Atos provided emitter plugin",
	"params": {
	"param": [
	{
	"-id": "port",
	"#text": "50000"
	},
	{
	"-id": "address",
	"#text": "127.0.0.1"
	}
	]
	}
	},
	"encoderRef": "simpleEncoder"
	}
	}
	}
	}

4.4.4.2 GetComplexEvent plublisher:

	Get ComplexEvent publisher
Description	Gets the definition of a specified complex event publisher
URL	BASE_URL/solcep/ complexEventPublisher/{publisher-id}/
Method	DELETE
Input	-

Output	**Example**
	
	"eventPublisher": {
	"-id": "publisher",
	"enabled": "1",
	"port": "5555",
	"encoders": {
	"encoder": {
	"-id": "simpleEncoder",
	"enabled": "1",
	"plugin": {
	"name": "libSimpleEncoderPlugin.so.1.0",
	"comment": "Atos provided encoder plugin"
	}
	}
	},
	"emitters": {
	"emitter": {
	"-id": "udpEmitter",
	"enabled": "1",
	"plugin": {
	"name": "libUDPEmitterPlugin.so.1.0",
	"comment": "Atos provided emitter plugin",
	"params": {
	"param": [
	{
	"-id": "port",
	"#text": "50000"
	},
	{
	"-id": "address",
	"#text": "127.0.0.1"
	}
	]
	}
	},
	"encoderRef": "simpleEncoder"
	}
	}

4.4.4.3 CreateComplexEventPublisher

	Create Complex Event Publisher
Description	Creates a complex event publisher
URL	BASE_URL/solcep/ complexEventPublisher/{publisher-id}/
Method	PUT
Input	-

Output	**Example**
	
	{
	"data": {
	"success": "true"
	}

	}

4.4.4.4 DeleteComplexEventPublisher

	Delete Complex Event Publisher
Description	Deletes the specified complex event publisher
URL	BASE_URL/solcep/ complexEventPublisher/{publisher-id}/
Method	DELETE
Input	-

Output	**Example**
	
	{
	"data": {
	"success": "true"
	}
	}

4.4.4.5 GetEncoders

	Get Encoders
Description	Gets the list of encoders of a specified complex event publisher
URL	BASE_URL/solcep/ complexEventPublisher/{publisher-id}/encoders
Method	GET
Input	-

Output	**Example**
	
	{
	"encoders": {
	"encoder": {
	"-id": "simpleEncoder",
	"enabled": "1",
	"plugin": {
	"name": "libSimpleEncoderPlugin.so.1.0",
	"comment": "Atos provided encoder plugin"
	}
	}
	}
	}

4.4.4.6 Create or update Encoders

	Create or Update Encoders
Description	Creates or updates an encoder of a specified complex event publisher
URL	BASE_URL/solcep/ complexEventPublisher/{publisher-id}/encoders/{encoder id}
Method	PUT
Input	-

Output	**Example**
	
	{
	"data": {
	"success": "true"
	}

	}

4.4.4.7 Delete encoder

	Delete Encoder
Description	Deletes an encoder from encoders list of a complex event publisher
URL	BASE_URL/solcep/ complexEventPublisher/{publisher-id}/encoders/{encoder id}
Method	DELETE
Input	-

Output	**Example**
	
	{
	"data": {
	"success": "true"
	}
	}

4.4.4.8 GetEmitters

	Get Emitters
Description	Gets the list of emitters of a specified complex event publisher
URL	BASE_URL/solcep/ complexEventPublisher/{publisher-id}/emitters
Method	GET
Input	-

Output	**Example**
	
	{
	"emitters": {
	"emitter": {
	"-id": "udpEmitter",
	"enabled": "1",
	"plugin": {
	"name": "libUDPEmitterPlugin.so.1.0",
	"comment": "Atos provided emitter plugin",
	"params": {
	"param": [
	{
	"-id": "port",
	"#text": "50000"
	},
	{
	"-id": "address",
	"#text": "127.0.0.1"
	}
	]
	}
	},
	"encoderRef": "simpleEncoder"
	}
	}
	}

4.4.4.9 Create or update Emitter

	Create or Update Emitter
Description	Creates or updates an emitter from emitters list of a complex event publisher
URL	BASE_URL/solcep/ complexEventPublisher/{publisher-id}/emitter/{emitter id}
Method	PUT
Input	-

Output	**Example**
	
	{
	"data": {
	"success": "true"
	}
	}

4.4.4.10 DeleteEmiter

	Delete Emitter
Description	Deletes an emitter from emitters list of a complex event publisher
URL	BASE_URL/solcep complexEventPublisher/{publisher-id}/emitters/{emitter id}
Method	DELETE
Input	-

Output	**Example**
	
	{
	"data": {
	"success": "true"
	}
	}

4.4.4.11 Assign Encoder to Emiter

	Assign Encoder to Emitter
Description	Assigns an encoder to an emitter on a specified complex event publisher
URL	BASE_URL/solcep/ complexEventPublisher/{publisher-id}/emitters/{emitter id} /encoderRef/{encoder id}
Method	POST
Input	-

Output	**Example**
	
	{
	"data": {
	"success": "true"
	}
	}

4.4.4.12 Disassociate Encoder from Emitter

	Disassociate Encoder from Emitter
Description	Disassociates an encoder from an emitter on a specified complex event publisher

URL	BASE_URL/solcep/complexEventPublisher/{publisher-id}/emitters/{emitter id} /encoderRef/{encoder id}
Method	DELETE
Input	-

Output	**Example**
	
	{
	"data": {
	"success": "true"
	}
	}

4.4.5 Dolce specification

4.4.5.1 Get Dolce Specifications

	Get Dolce Specifications
Description	Gets the list of Dolce Specifications
URL	BASE_URL/solcep/ dolceRuleSpecification
Method	GET
Input	-

Output	**Example**
	
	{
	"dolceSpecifications": {
	"dolceSpecification": [
	{ "-id": "trafficIncidents" },
	{ "-id": "weatherIncidents" }
	]
	}
	}

4.4.5.2 Get dolce specification

	Get Dolce Specification
Description	Gets details of a specified dolce rule
URL	BASE_URL/solcep/ dolceRuleSpecification/{dolceSpec-id}
Method	GET
Input	-

Output	**Example**
	
	{
	"dolceSpecification": {
	"-id": "trafficIncidents",
	"event": {
	"-id": "TrafficReading",
	"definition": "use
	{
	int SensorId; //Id for sensor
	int Hour; // hour of the reading time
	int Minute; // minute of the reading time
	int Speed; // average speed of the cars
	int Intensity; // traffic intensity range 0-4, the smaller the faster

	pos Position;// geo position of the sensor
	}"
	},
	"complexEvent": {
	"-id": "TrafficAlert",
	"definition": "
	payload
	{
	int Id = SensorId,
	int Sped = Speed@
	}
	
	detect TrafficReading
	where TrafficReading@.Speed - TrafficReading.Speed > 30
	in [3] TrafficReading"
	}
	}
	}

4.4.5.3 Add/Modify dolce specification

	Add or Update Dolce Specification
Description	Adds or updates the specified dolce specification
URL	BASE_URL/solcep/ dolceRuleSpecification/{dolceSpec-id}
Method	PUT
Input	-

Output	**Example**
	
	{
	"data": {
	"success": "true"
	}
	}

4.4.5.4 Delete dolce specification

	Delete Dolce Specification
Description	Deletes a specified dolce specification
URL	BASE_URL/solcep/ dolceRuleSpecification/{dolceSpec-id}
Method	DELETE
Input	-

Output	**Example**
	
	{
	"data": {
	"success": "true"
	}
	}

4.5 Interfaces to Workflow Management

The Workflow Management / Orchestrator module, as designed at the time of the preparation of this deliverable, does not access directly the PPIs; the ICO Discovery

services are used instead. If, however, we decide to change this decision at a later stage, then the requirements can be covered by current PPI specification.

4.6 Interfaces required by the Management and Governance Layer

4.6.1 Management functionality supported by the IoT system

The Management and Governance layer requires the addition of a new section to the IoT ontology (http://vital-iot.org/contexts/system.jsonld), 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. This new section will be reflected in the second version/release of the ontology as part of deliverable D3.1.2.

This section has the structure presented in the table 16.

Table 16: IoT System management section.

```
{
      "@context":
        "http://vital-iot.org/contexts/service.jsonld",
      "type": "SystemManager",
      "msm:hasOperation":
      [
        {
          "type": "GetPerformanceMetrics",
          "hrest:hasAddress":
            "http://www.example.com/performance",
          "hrest:hasMethod": "hrest:GET"
        },
        {
          "type": "GetConfigurationOptions",
          "hrest:hasAddress":
            "http://www.example.com/configurationOptions",
          "hrest:hasMethod": "hrest:GET"
        },
        {
          "type": "SetConfigurationOptions",
          "hrest:hasAddress":
            "http://www.example.com/configurationOptions",
          "hrest:hasMethod": "hrest:POST"
        }
      ]
    }
```

The responses are in JSON-LD and structured as measurement data. Their values are defined by a specific taxonomy that the VITAL platform defines and the PPI implementation conforms to (in essence the PPI transforms the IoT system data to this format).

Table 17: Access to performance metrics metadata

	Get performance metrics metadata	
Description	VITAL pulls from an IoT system metadata about the supported management metrics.	
Method	GET	
Request headers		

Request body		
Response headers	Content-Type	application/json

Response body	**Example**
	
	{
	"ssn:observes":
	[
	{
	"type": "<ontology definition URI>/upTime",
	"uri": "http://www.example.com/iot/{id}/perf/upTime"
	},
	{
	"type": "<ontology definition URI>/systemLoad",
	"uri": "http://www.example.com/ iot/{id}/perf/systemLoad"
	},
	{
	"type": "<ontology definition URI>/memUsed",
	"uri": "http://www.example.com/iot/{id}/perf/memUsed"
	},
	{
	"type": "<ontology definition URI>/memAvailable",
	"uri": "http://www.example.com/iot/{id}/perf/memAvailable"
	},
	{
	"type": "<ontology definition URI>/servedRequests",
	"uri": "http://www.example.com/iot/{id}/perf/servedRequests"
	},
	{
	"type": "<ontology definition URI>/pendingRequests",
	"uri": "http://www.example.com/iot/{id}/perf/pendingRequests"
	},
	{
	"type": "<ontology definition URI>/maxRequests",
	"uri": "http://www.example.com/iot/{id}/perf/maxRequests"
	},
	{
	"type": "<ontology definition URI>/errors",
	"uri": "http://www.example.com/iot/{id}/perf/errors"
	},
	
	…
	]
	}	

Notes	• 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 18: Access to configuration option metadata

	Get configuration option metadata	
Description	VITAL pulls from an IoT system metadata about the supported configuration options.	
Method	GET	
Request headers		
Request body		
Response headers	Content-Type	application/json
Response body	**Example**	

	<pre>{
	"configurationOptions":
	[
	{
	"name": "c1",
	"value": "v1",
	"type": "<ontology URI>/string | number | complex",
	"permissions": "rw"
	},
	{
	"name": "c2",
	"value": "v2",
	"type": "<ontology URI>/string | number | complex",
	"permissions": "r"
	},
	
	…
	]
	}</pre>

Notes	• 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).

4.6.2 Monitoring hooks

The Management & Governance layer will call the individual URIs of the required monitoring service, as defined in the result of the GetPerformanceMetrics operation (refer to Table 17).

4.6.3 Configuration hooks

The Management & Governance layer will use the SetConfigurationOptions to set/update one or more configuration parameters with "rw" permission, and provided that the IoT system supports the operation (as shown in Table 16). The request format is outlined in the table 19.

Table 19: APIs for setting configuration options

	Set configuration options	
Description	VITAL sets one or more configuration options of an IoT system.	
Method	GET	
Request headers	Content-Type	application/json

Request body	**Example**
	<pre>{
	"configurationOptions":
	[
	{
	"name": "c1",
	"value": "new value v1",
	},
	{
	"name": "c2",
	"value": "new value v2",</pre>	

	<pre>},
	…
	]
	}</pre>

Response headers	
Response body	
Notes	• The status code of the request can be: ◦ 200 (success) ◦ 403 (forbidden, if the permission is read-only) ◦ 404 (if the name is not found or not supported) • Note that the request body must contain at least one parameter to be updated otherwise the PPI API will return 404[3]. It may contain up to the total number of parameters supported by the PPI.

4.6.4 ICO management

The management of ICOs can be supported by the existing design without any extra ontologies and services, provided that the Management and Governance layer is interested solely in the status of an ICO and its location data (if supported by the ICO). Configuration options will be supported via the GetConfigurationOptions and SetConfigurationOptions operations exposed by the IoT system, as described above.

5 VIRTUALIZED DATA PROCESSING FUNCTIONS

5.1 Purpose and Scope

The VITAL virtualized data access and processing functions aim at providing access to VITAL data in a platform-agnostic way in order to facilitate the development of solutions for data intensive problems. Based on these interfaces, solution developers will be able to implement solutions to data intensive problems, through appropriately embedding VUAI invocations in their programs. At the same time, the VUAIs associated with data processing functionalities will provide a basis for implementing a data processing toolbox, which will be integrated into the VITAL development tools (designed/developed as part of WP5). The main purpose of these development tools is to enable developers, who are not experts in the IoT field, to build IoT applications and services with minimum effort. In order to achieve that, VITAL intends to offer to developers, among others, a data analysis and mining toolbox that will allow them to perform various common data processing tasks on data stemming from VITAL. The data might be either raw observations or complex events.

The following sections roughly describe the functionalities that will be available in the data analysis and mining toolbox of the project and that will be supported by the VUAIs. The implementation of all these tools will be based on popular open-source libraries for data processing (such as R), which also include functionalities for statistical computing and graphics.

[3] Alternatively «400» (bad request)

Note that the specification of VUAIs for processing data sets, along with the subsequent implementation of the data processing toolkit can be considered a first step to the integration of BigData functionalities [Marz14] over the VITAL smart city platform. While VITAL will not be implementing a fully fledged BigData toolbox (e.g. it will not be dealing with knowledge harvesting from unstructured data), the offered functionalities when applied over large data sets [Di Ciaccio12] will be proven extremely useful for a wide range of smart city applications such as the proliferating smart energy applications [Sioshansi11].

5.2 Data Analysis

VITAL data and events might carry temporal information (when they were collected or when they occurred), spatial information (where they were collected or where they occurred) or both. Based on that, we might be able to apply well-known models, algorithms or functions from the fields of econometrics, statistics, time series analysis and spatial analysis to these data, in order to better understand them and even make some estimation.

5.2.1 Temporal Analysis

VITAL shall include tools that, given a temporal data set, will predict the next N values in the future on an hourly, weekly, monthly or yearly basis. This tool will give the answer to questions like: "*What will be the expected temperature in the next 5 weeks?*".

Under the hood, this tool will first convert the timestamped data into a time series with the appropriate frequency, and fit the time series data using an AutoRegressive Integrated Moving Average (ARIMA) model. The result of the tool will be a plot that will show the observed and the predicted values, as well as the error bounds for the latter.

Table 20 describes a RESTful web service that has been designed for the support of the temporal analysis tool.

Table 20: A RESTful web service for the support of the time analysis tool

	Analyze temporal data	
Description	Receives a temporal data set (e.g. a set of observations or events) and produces a plot that depicts both the received value and predictions and error bounds for the next N values.	
Method	`POST`	
Request headers	`Content-Type`	`application/json`

Request body	**Example**
	
	`{`
	`  "N": 5,`
	`  "data":`
	`  [`
	`  {`
	`  "time": "2014-08-20T16:47:32+01:00"`
	`  },`	

	<pre>{
	"time": "2014-08-20T20:47:32+01:00"
	}
	]
	}</pre>

Response headers	Content-Type image/png
Response body	-
Notes	0The body of the request is a JSON chunk. Both **N** and **data** fields are mandatory. **N** denotes the number of values to predict, and **data** contains the values of the data set to analyze. 1The data set can contain observations or any timestamped events.

5.2.2 Spatial Analysis

The spatial analysis tool will receive a set of points that indicate where certain events occur in a specific area, and will give a prediction about the intensity of the events in this area. Questions like "*How often do car accidents occur in the streets of this area?*" will be answered by this tool.

The implementation of the tool is based on Ordinary Kriging process regression, and it is expected to produce as a result an density map of the area of interest.Table 21 describes a RESTful web service that has been designed for the support of the spatial analysis tool.

Table 21: A RESTful web service for the support of the spatial analysis tool

	Analyze spatial data
Description	Receives a spatial data set (e.g. a set of observations or events) and produces a density map.
Method	POST
Request headers	Content-Type application/json

Request body	**Example**
	<pre>{
	"data":
	[
	{
	"location":
	{
	"type": "Point",
	"lat": "55.701",
	"long": "12.552",
	"alt": "4.33"
	}
	},
	{
	"location":
	{
	"type": "Point",
	"lat": "55.801",
	"long": "12.552",
	"alt": "4.33"
	}
	}
	]
	}</pre>

Response headers	Content-Type	image/png
Response body	–	
Notes	0The body of the request is a JSON chunk. The **data** field is mandatory and contains the values of the data set to analyze. 1The data set can contain observations or any events that took place at a specific location.	

5.2.3 Spatio-temporal Analysis

In order to support spatio-temporal data analysis, we will implement a tool that will basically receive event sets that contain both temporal and spatial information, and will produce density plots per day, week, month or year. The implementation of this tool will be based on the implementation of the temporal and spatial analysis tools that were described in the previous sections. The tool is expected to give answers to questions that involve both “when” and “where”.

Table 22 describes a RESTful web service that has been designed for the support of the spatio-temporal analysis tool.

Table 22: A RESTful web service for the support of the spatio-temporal analysis tool

	Analyze spatio-temporal data	
Description	Receives a spatio-temporal data set (e.g. a set of observations or events) and produces a set of maps that show the density of the data in the area over time.	
Method	POST	
Request headers	Content-Type	application/json

Request body	**Example**
	
	{
	"data":
	[
	{
	"time": "2014-08-20T16:47:32+01:00",
	"location":
	{
	"type": "Point",
	"lat": "55.701",
	"long": "12.552",
	"alt": "4.33"
	}
	},
	{
	"time": "2014-08-20T17:47:32+01:00",
	"location":
	{
	"type": "Point",
	"lat": "55.801",
	"long": "12.552",
	"alt": "4.33"
	}
	}
	]
	}	

Response headers	Content-Type	image/png
Response body	–	

Notes	0The body of the request is a JSON chunk. The **data** field is mandatory and contains the values of the data set to analyze. 1The data set can contain observations or any events that are accompanied by a timestamp that shows when and a location that shows where they occurred.

5.3 Data Mining

As part of the VITAL toolbox for IoT developers, we intend to provide a tool that will expose some basic data mining functionality. The tool will perform a mere application of the Apriori algorithm to a data set, and will give back a set of association rules that it will have discovered. Our main intention is to experiment with variations of the Apriori algorithm that work with data streams instead of stored data sets.

Since we are still exploring the related work and, thus, we will (at this stage) refrain from giving any descriptions of web services needed to support the data-mining tool. Such descriptions will be provided in the second release of the deliverable.

6 OUTLOOK AND CONCLUSIONS

VITAL relies on a number of abstract virtualized interfaces to access (IoT-based) data streams and services in smart cities regardless of the IoT system (platform, data sources) that enables these functionalities. Earlier sections have provided an initial specification of VITAL's abstract interfaces for accessing IoT data and services in a platform-agnostic way. These interfaces have been classified in three categories: namely 1) interfaces to platforms, 2) interfaces to added-value functionalities of the VITAL platform and 3) interfaces to data processing functions. For each of the categories we have presented the interfaces. In the case of platform and added-value functionality access, the interfaces have been described at a very fine level of detail, i.e. down-to-implementation detail. This is not the case with the specification of the data processing functions, which is still in less mature state, both in terms of completeness and in terms of detail.

In parallel with the specification of the above-listed abstract interfaces, the partners have undertaken a significant step towards the realization of the interfaces. This is for example the case with the implementation of the PPIs over the four IoT platforms selected (in WP2): PPI implementations have been already realized (over X-GSN, FIT, Hi REPLY and Xively.com) and they are under testing and validation. Moreover, the implementation of the interfaces for the added-value functionalities is also in progress, as part of the implementation of the respective modules in WP4 and WP5 of the project. As expected, the implementation of the data processing interfaces is still in its infancy, given the need to finalize the selection of the open-source tools that will empower the realization of the VITAL statistical and BigData processing functionalities.

The present deliverable represents the first version/release of the VITAL abstract interfaces. A second and (final) release is planned six months later according to the project's work plan. This final release will report on updates to the specification of VITAL VUAIs, while also comprising their final implementation. In particular, the final release of the deliverable will be incremental to the present one and it will (additionally) contain:

- Updates to and fine-tuning of the PPI specifications, including any revisions that will be required in order to be in-line with the final version of the VITAL ontology.
- Updates to and fine-tuning of the interfaces to added-value functionalities, based on feedback from the actual implementation of these functionalities in WP4 and WP5 of the project.
- A more thorough (down to implementation detail) specification of the abstract interfaces to data processing, statistical processing and BigData functionalities of the VITAL platform.
- The implementation of the various abstract interfaces. Note that the deliverable will focus on the final implementation of the PPIs, given that the implementation of the rest of the interfaces (e.g. interfaces to service discovery, interfaces to management functionalities, interfaces to data mining etc.) will be implemented as part of the work packages where the respective modules are being implemented.

Overall, we envisaged that the conclusion of the present deliverable will form a sound basis for VITAL solution developers to access diverse platforms and data sources in a well-defined, versatile and effective way.

7 REFERENCES

[Di Ciaccio12] Agostino Di Ciaccio, Mauro Coli, Jose Miguel Angulo Ibanez Eds.) «Advanced Statistical Methods for the Analysis of Large Data-Sets», Series: Studies in Theoretical and Applied Statistics, Subseries: Selected Papers of the Statistical Societies 2012, XIII, 484p.

[Marz14] Nathan Marz, James Warren, «Big Data: Principles and Best Practices of Scalable Real-time Data Systems», Feb 2014, Paperback ISBN13: 9781617290343, ISBN10: 1617290343

[Sioshansi11] Fereidoon P. Sioshansi, «Smart Grid: Integrating Renewable, Distributed & Efficient Energy», November 2011, ISBN-10: 0123864526 | ISBN-13: 978-0123864529.

