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**

D4.1.1 Intelligent virtualized discovery of resources V1

Document Id:	VITAL-D411-141220-draft
File Name:	VITAL-D411-141220-draft.docx
Document reference:	Deliverable 4.1.1
Version:	Draft
Editors:	Nathalie Mitton, Valeria Loscrí, Riccardo Petrolo
Organisation:	INRIA
Date:	20/12/2014
Document type:	Deliverable
Security:	PU (Public)

DOCUMENT HISTORY

Rev.	Author(s)	Organisation(s)	Date	Comments
V01	Riccardo Petrolo	INRIA	29/09/2014	Initial version of the document
V02	Riccardo Petrolo	INRIA	12/11/2014	Assignment of Partners Responsibilities and minor fixes
V03	Riccardo Petrolo	INRIA	18/11/2014	Adding REST, JAVA EE, Introduction JSON-LD
V04	Riccardo Petrolo	INRIA	19/11/2014	Intro Service Discovery, ICOs Discovery
V05	Anne Helmreich	NUIG	21/11/2014	Adding Data Management Service
V06	Tomas Geraghty	NUIG	21/11/2014	Contributions on DMS
V07	Fotis Stamatelopoulos, Angelos Lenis	SiLo	21/11/2014	Added Orchestration/BPM module
V08	Riccardo Petrolo	INRIA	22/11/2014	Adding details Service Discovery, ICOs Discovery
V09	Valeria Loscrí	INRIA	24/11/2014	Adding Intelligent Discovery, update Architecture
V10	Riccardo Petrolo	INRIA	24/11/2014	Polishing, Adding Filtering, Adding Tables in Intelligent Discovery and ICO Discoverer
V11	Umut Yıldırım	Atos	25/11/2014	Added Interfaces to CEP Module
V12	Riccardo Petrolo	INRIA	25/11/2014	Minor fixes, polishing
V13	Valeria Loscrí	INRIA	26/11/2014	Adding details in Intelligent Discovery and ICO Discoverer
V14	Nathalie Mitton, Riccardo Petrolo	INRIA	01/12/2014	Fixing document structure
V15	Riccardo Petrolo	INRIA	02/12/2014	Minor fixes
V16	Kasper de Graaf, Malcolm Garrett	Images&Co	04/12/2014	Quality Review
V17	Andrea Martelli	Santer Reply	04/12/2014	Technical Review
V18	Riccardo Petrolo	INRIA	04/12/2014	Changed based on T/Q reviews
V19	Valeria Loscrí	INRIA	15/12/2014	Circulated for SB Approval
Draft	Martin Serrano	NUIG	20/12/2014	EC Submitted Draft

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TERMS AND ACRONYMS

API	Application Programming Interface
BPM	Business Process Management
CEP	Complex Event Processing
DMS	Data Management Service
HTTP	Hypertext Transfer Protocol
ICO	Internet-Connected Object
IoT	Internet of Things
JAVA EE	Java Enterprise Edition
JSON	JavaScript Object Notation
JSON-LD	JSON for Linked-Data
OWL	Web Ontology Language
RDF	Resource Description Framework
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
SPARQL	SPARWL Protocol and RDF Query Language
URI	Uniform Resource Identifier
URN	Uniform Resource Name
VOB	VITAL Orchestration / BPM Module
W3C	World Wide Web Consortium

1 INTRODUCTION

This deliverable specifies the architecture and features of the Service Discovery module and positions it in the VITAL framework. It also gives details about its implementation.

The main task of this component is to provide the means for discovering ICOs, IoT services and IoT data that are virtualized in the VITAL platform. The VITAL Service Discovery module is additionally enhanced through the use of smart predictive mechanisms to allow the largest services. Other VITAL modules (e.g., CEP, Filtering, Orchestration) will use the functionalities of the Service Discovery to operate on the IoT resources that are needed for their particular business context.

For example, the Filtering module will access the Service Discovery module in order to access a list of ICOs, to which it will apply its filtering techniques to reduce unwanted information.

A key element of the Service Discovery module will be the ICOs Discoverer. It will provide tools for discovering ICOs and data streams, using inputs such as location and type. The ICOs Discoverer plays an important part in ensuring that the data sets which are accessed, processed and visualized are restricted to those that are relevant to the application context, in compliance with the policies of every physical network.

Please note this document is a first version. As such it will present the main architecture, the main directions to be taken and the advances so far achieved within the project with regard to ICOs discovery and how this relates to other modules of the VITAL architecture. It provides the architecture of the ICOs Discoverer module, leaving the implementation of the other components for the final version, deliverable at a later stage. This document will be continuously developed to reflect the advances and outcomes achieved as the project proceeds. Following the project advances, The second and final releases of this document will reflect these advances with updates and revisions covering specification as well as implementation issues.

The deliverable is produced with to distinct audiences in mind: first, VITAL consortium members, notably researchers and engineers, who require the functionalities of the Service Discovery Module in order to develop their components; and second, external researchers and developers who want to use VITAL IoT resources in the development of their own applications.

The document is structured as follows. We first introduce the Service Discovery and its features; we present the technologies we are using in development; we show how the ICO Discoverer is implemented, designed, finalised and connected to the other modules of the VITAL architecture. We conclude the deliverable with a summary and an outline of the next steps.

2 SERVICE DISCOVERY

The Service Discovery (SD) module is responsible for discovering ICOs, data streams, and services and is horizontally integrated in the VITAL platform as shown in Fig. 1. Furthermore, it will deal with IoT resources requested by other modules without regard to the underlying platforms, focusing only on the meaning of the information.

Fig. 1 also shows how the Service Discovery interacts directly with the Data Management Service (DMS) which has storage capabilities, and where the various data streams (and their metadata) are modelled and formatted according to the VITAL ontology – see [Vital-D3.1.1-2014].

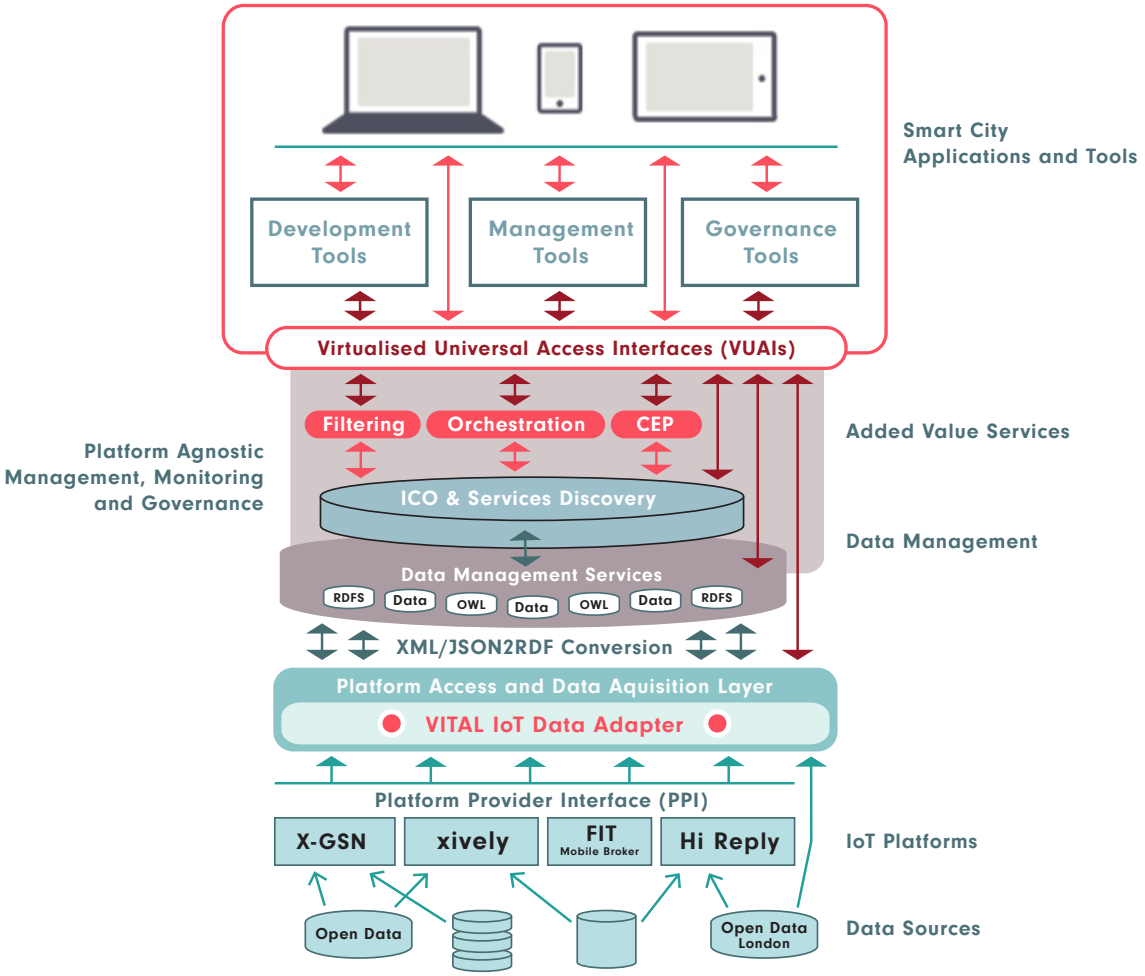

Figure 1. VITAL Architecture

The other modules in the architecture (CEP, Filtering, Orchestration) will use the functionalities of the Service Discovery to access IoT resources according to their particular business context. For example, to validate the Use Case (UC07) described in [Vital-2.2-2014], the CEP Module will access the Discoverer in order to retrieve the ICOs available in a defined location.

2.1 ICO classification and Intelligent Discovery

The Service Discovery has been implemented incrementally, starting with a basic version. It relies on a classification of the ICO such as that shown in

Table 1. In this example, an ICO is classified by reference to its motion and further qualified with information about the connectivity and localization capabilities of the device itself. This will allow the Service Discovery to include in its results information regarding the data availability of mobile devices in a defined area.

Table 1. Events Categorization

ICOs	Localization	Connection
Fixed	continue/ intermittent	continue/ intermittent
Mobile - predictive pattern	continue/ intermittent	continue/ intermittent
Mobile - random pattern	continue/ intermittent	continue/ intermittent

To enhance the service and make the discovery intelligent, the VITAL module will be equipped with smart predictive mechanisms by which it will be able to forecast the connectivity or data availability of a specific device based on its classification. We discuss these models in Section 6.

3 TECHNOLOGIES

In this chapter, we offer a short review of the technologies that could potentially have been used for the implementation of the Service Discovery. The details (heuristic details on data model and data format for Linked Data) are in deliverable Vital-D3.1.1-2014. We introduce the REST architecture and its basic concept. Then we briefly discuss JAVA EE as a programming technology for developing the REST interfaces and manipulating JSON data.

3.1 REST

Representational State Transfer (REST) is a term coined by Roy Fielding in 2000 [RFielding00] to refer a software architectural style.

Before the introduction of REST, the first generation web services relied on exchanging XML packets conforming to SOAP (Simple Object Access Protocol) specification using HTTP protocol.

SOAP and XML are now considered too heavy, and HTTP itself has sufficient capabilities to allow applications to communicate over the network.

REST ignores the details of the components' implementation and protocol syntax in favour of focusing on the roles of components, the constraints on their interaction with other components, and their interpretation of significant data elements. It encompasses the fundamental constraints on *components*, *connectors*, and *data* that define the basis of the Web architecture, and thus the essence of its behaviour as a network-based application.

REST components communicate by transferring a representation of a resource in a format matching one of an evolving set of standard data types, selected dynamically based on the capabilities or desires of the recipient and the nature of the resource. REST's data elements are summarized in Table 2.

Table 2. REST Data Elements

Data Element	Modern Web Examples
resource	the intended conceptual target of a hypertext reference
resource identifier	url, urn
representation	html document, jpeg image
representation metadata	media type, last-modified time
resource metadata	source link, alternates, vary
control data	if-modified-since, cache-control

A resource represents the key abstraction of information in REST. Any information that can be named can be a resource: a document or image, a temporal service, a collection of other resources, and so on.

Web services APIs that adhere to the architectural constraints are called RESTful. HTTP-based RESTful APIs are defined with these aspects:

- base URI, such as http://example.com/resources/
- an Internet media type for the data. This is often JSON but can be any other valid Internet media type (e.g. XML, Atom, microformats, images, etc.)
- standard HTTP methods (e.g., GET, PUT, POST, or DELETE)
- hypertext links to reference state
- hypertext links to reference related resources

3.2 Java EE

Originally specified by Sun in 1999, Java Enterprise Edition (formerly J2EE) is currently the most widely used web programming technology.

Java EE offers many advantages: independence, portability, multi-layer structure system, efficient development, scalability and stable usability to name but a few, making it a platform of first choice for building web-based application systems. Many

implementations are available, both commercial (IBM, HP, Sun, Oracle, etc.) and open-source (WildFly, Apache Geronimo, etc.).

For the development of the Service Discovery module, we have used WildFly (http://www.wildfly.org), formerly known as JBoss AS, not least because it supports the latest standards for REST-based data access including JSON.

3.3 Conclusion

The ability of REST to fully leverage the protocols and standards that power the World Wide Web, combined with its simplicity relative to SOAP-based approaches, have contributed to the position where it is now the preferred technology for building web services, including services from large vendors such as Google, Yahoo, Amazon and Microsoft.

In addition, the considerations dealt with in Deliverable [Vital-D3.1.1-2014] (Virtual Models, Data and Metadata for ICOs, V1), also point to REST as the technology that better suits the communications between the different VITAL modules.

Within the VITAL context, we therefore decided to adopt REST for the communication between the different modules of the architecture.

4 SERVICE DISCOVERY IMPLEMENTATION

4.1 Global implementation

The Service Discovery is accessible through a RESTful web service, which exposes information like the context, a description, its status and the operations it offers. Table 3 shows a résumé of the operations available so far. The `ConnDMS`, for example, gives information about the status of the connection with the Data Management Service.

Table 3. 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"
    }
  ]
}
```

In this first version of the Deliverable, we present the features of the ICOs Discoverer, leaving implementation of the services and the Data Streams Discoverer to the next version.

4.2 Implementation of enhanced mechanisms

In order to support the above functionalities, the device description is enriched with some properties as defined in Deliverable [Vital-D3.1.1-2014] as follows:

- `hasMovementPattern`, a mandatory property that links to an instance of MovementPattern;
- `hasLocalizer`, an optional property that links to an IoT service specification that provides access to the current location of the sensor.
- `hasNetworkConnection`, an optional property that links to an instance of NetworkConnection.

The property of the ICO MovementPattern gives information related to the motion of a device. In the basic version of the Discovery module, the ICO movement pattern is defined as `Stationary`, indicating that the ICO is not moving at all and the device can be characterized with information about its location (`hasLastKnownLocation`). If this property is not defined, there is no information about the sensor location.

In a more "advanced" version of the Discovery module, the mobility of a device can be defined by two possible types of `MovementPattern`: Predictive and Random. The first provides additional information that allows future movement and locations to be "predicted" (e.g. by defining the direction and the velocity),

as shown in Table 4. In Deliverable [Vital-D3.1.1-2014], we specified that a mobile sensor in a system that supports a localization service should include the property `hasLocalizer`, as defined in

Table 5. The localizer service can then rely on information transmitted by the sensor itself (e.g. through a GPS receiver), or alternatively it can use an external tracking system. In either case, the Discovery service will provide a list of objects that support the localizer service.

Table 4. Get ICOs Mobility

	Get ICOs Mobility
Description	Used to retrieve information about the Mobility of the ICOs
URL	BASE_URL/discoverer/getICOsMobility
Method	GET
Input	`mobilityType`
Output	Information about the ICOs mobility, i.e., Fixed, Mobile (in this case it provides additional details about the kind of mobility, predicted or random). The data follows the JSON-LD format.

Table 5. Get ICOs Localizer Service

	Get ICOs Localizer Service
Description	Used to search the ICOs that provide a *localizer service,* e.g. by giving access to its local GPS receiver
URL	BASE_URL/discoverer/getICOsLocalizerService
Method	GET
Input	-
Output	List of ICOs with localizer service

The information regarding connection stability (

Table 6) indicates whether the sensor is connected continuously or intermittently to a communication network.

Table 6. Get ICOs Connection Stability

	Get ICOs Connection Stability
Description	This interface is used to get information about the type of connection of the ICOs
URL	BASE_URL/discoverer/getICOsConnectionStability
Method	GET
Input	`connectionType`
Output	It returns connectivity capabilities about ICOs in JSON-LD format

Opportunistic filtering, orchestration and processing of complex events from multiple heterogeneous Cloud resources will allow diverse silos to be integrated horizontally. In this respect, the VITAL concept moves beyond the Internet of Things by referencing Cloud computing associated with the Internet of Things; we refer to this as “the Cloud of Things” (CoT) [PLM+14] [PLM2+14].

5 ICOs DISCOVERER

Deliverable [Vital-D3.1.1-2014] provides information on the data models and ontologies used by VITAL to describe the sensor/ICO.

In the short example in Table 7, we can observe the parameters used to represent a `VitalSensor`, i.e. name, type, VITAL URI and description. Meaningful information used by our ICOs Discoverer, such as the location and the observed properties, is stored in this last container.

Table 7. Example VitalSensor

```
{
  "@context": "http://vital-iot.org/contexts/sensor.jsonld",

  "name": "TemperatureSensor No.123",
  "type": "VitalSensor",
  "description": "This is an example sensor",
  "uri": "http://www.example.com/vital/sensor/123",
  "hasLastKnownLocation": {
    "type": "geo:Point",
    "geo:lat": "53.2719",
    "geo:long": "-9.0489"
  },
  "ssn:observes": [
    {
      "type": "http://lsm.deri.ie/OpenIoT/Light",
      "uri": "http://www.example.com/vital/sensor/123/light"
    },
    {
      "type": "http://lsm.deri.ie/OpenIoT/Temperature",
      "uri": "http://www.example.com/vital/sensor/123/temperature"
    }
  ],
  "ssn:madeObservation":
    "http://www.example.com/vital/sensor/123/obsvn/1"
}
```

We present below a synopsis of the ideal interfaces that should be supported by the Discovery Service in order to satisfy requests from other modules of the architecture (e.g. CEP). Note that the list below does not give the exhaustive details of them but states the basis. The on-going works on Tasks 4.2, 4.3, and 4.4 may enrich and in case mutualize them. Final release of this document (D4.1.2) will detail the actual interfaces to be implemented in VITAL.

To retrieve ICO information, the Discoverer presents different functions, such as:

- `ConnDMS` (Table 8).
- `nICOs` (Table 9) is an interface that accepts the `GET` method, and returns the number of ICOs available in the DMS.
- `getICOs` (Table 10) is a function that outputs the ICOs available in a specified area identified by the parameters in the input: `latitude`, `longitude`, `radius`. The kind of information that has to be retrieved using the `observationProperty` field can also be specified.

`getICO (`

- Table 11) is used to obtain information regarding a specific ICO, characterized by its URI in the VITAL platform.

Table 8. 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 9. Number of ICOs

	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 10. 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 11. 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

5.1 INTERFACES TO DATA MANAGEMENT SERVICE (NUIG)

Discovery uses Data Management Service (DMS) to GET data by use of parameters. It is possible to get metadata, measurements and children (all or specific ones). New children can be added to an existing child with POST.

Table 12. DMS - Get all metadata

	Get all metadata
Description	VITAL pulls from an IoT system / service / sensor its metadata.
URL	BASE_URL/metadata
Method	GET
Input	-

Output	**Example**
	
	{
	"name": "temperature-test-1",
	"description": "Reports current temperature",
	"uri": "sensors.test.com/125",
	"location": {
	"long": -1,
	"lat": 1
	}
	}

Table 13. DMS - Get specific measurement

	Get specific measurement from specific child
Description	VITAL pulls from child its measurements
URL	BASE_URL/children/:childID/measurements/125
Method	GET
Input	-

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

Table 14. DMS - Post children

	Post children to specific child
Description	VITAL pushes children to child
URL	BASE_URL/children/:childID/children
Method	GET

Input	**Example**
	
	{
	"123":{
	"metadata": {
	"name": "temperature-test-1",
	"description": "Reports current temperature",

	<pre>"uri": "sensors.test.com/125",
	"location": {
	"long": -1,
	"lat": 1
	}
	},
	"measurements": {
	"125": {
	"time": "2014-11-10T10:07:02+01:00",
	"temperature": 18.4
	}
	},
	"children":{
	"valid": true
	}
	}
	}</pre>

Output	-

Additional details regarding the concept of children, parameters and the above interfaces can be found in the on-going deliverables D3.1.2 and D3.2.1 of the VITAL consortium.

5.2 INTERFACES TO FILTERING

The Filtering module offers mechanisms for filtering data and metadata which are obtained from different sources and stored in the VITAL DMS. It communicates directly with the Service Discovery to retrieve data, to which filtering mechanisms are then applied.

The module uses an interface (`ConnSD`) to confirm the status of the connection with the Service Discovery (

Table 15).

Table 15. Filtering - Connection to SD

	Connection to Service Discovery
Description	It gives information about the status of the connection with the SD.
URL	BASE_URL/ConnSD
Method	GET
Input	-

Output	<pre>{
	"@context": "http://vital-iot.org/contexts/service.jsonld",
	"type": "Filtering/ConnSD",
	"hrest:hasAddress": "BASE_URL/ConnDMS",
	"hrest:hasMethod": "hrest:GET",
	"hrest:status": "ON"
	}</pre>

Another interface request by the Filtering module is `Get ICOs` (see Table 10). It indeed 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`).

5.3 INTERFACES TO CEP MODULE (ATOS)

CEP uses Service Discovery to access IoT resources and their observation data characteristics, and also to subscribe to and unsubscribe from observation streams of ICOs. To deliver these features, CEP requires the Discoverer services and functionality described in Tables 16-19.

Table 16. CEP - Get IoT systems

	Get IoTSystems
Description	CEP queries the discoverer for registered underlying IoT systems. This is used by the CEP for acquiring the active IoT systems and populating lists for data source selection.
URL	BASE_URL/iotsystemSearch
Method	GET
Input	Filter options on location covered, or type of system.
Output	`List of IoT systems with any metadata information maintained by VITAL in JSON-LD format`

Table 17. CEP - Get IoT system

	Get IoTSystem
Description	CEP queries the discoverer for a specific underlying IoT platform. This is used by the CEP for acquiring the observation data and its characteristics to be used in CEP stream emitters.
URL	BASE_URL/getIoT
Method	GET

Input	Example format:
	`{`
	`  "IoT_system": { "systemId": "sid", "componentId":"cid" } // cid is optional`
	`}`

Output	`Metadata of IoT System in JSON-LD format`

Table 18. CEP - Subscribe observation stream

	Subscribe to Observation Stream
Description	CEP subscribes to the observation stream of a specific IoT System or ICO.
URL	BASE_URL/subscribeObserver
Method	POST

Input	Example format:
	`{`
	`  "IoT_system": { "systemId": "sid", "componentId":"cid" }`
	`OR`
	`  "Ico" : { "componentId":"cid"  }`
	`}`

Output	CEP subscribed to Observation stream of the Iot System / ICO
	
	`{`
	`  "subscription_id": { "9236438ac38045188e63f47af9ced8dd" }`
	`}`

Table 19. CEP - Unsubscribe observation stream

	Unsubscribe from Observation Stream
Description	CEP unsubscribes from the observation stream
URL	BASE_URL/unsubscribeObserver
Method	POST

Input	Example format:
	{
	"subscription_id": { "9236438ac38045188e63f47af9ced8dd" }
	}

Output	CEP unsubscribed from Observation stream of the Iot System / ICO

Additional interfaces requests by the CEP module are `Get ICOs` (as in Table 10) and `Get ICO` (as in

Table 11).

5.4 INTERFACES TO ORCHESTRATION (SiLo)

The goal of VITAL's Orchestration/BPM module (VOB) is to achieve cross-platform and cross-business-context process integration, supporting VITAL's overall goal of providing an abstract digital layer over the application silos of the modern smart city. To this end, VOB allows for definition of new VITAL eServices that combine multiple underlying services and processes (as homogenized by the VITAL platform) and these can be called by the applications. VOB requires the following Discoverer services/functionality:

Table 20. VOB - Search service endpoints

	Search service endpoints
Description	VOB queries the Discoverer for service endpoints and the supported services available in the specific VITAL platform deployment. These are services offered by the sub-components (e.g. CEP, data services), e.g. data streams offered by an IoT system for specific ICOs in an area, data streams related to energy consumption in a specific geographic area, a specific CEP service. This is used by the VOB designer application.
URL	BASE_URL/serviceSearch
Method	GET

Input	Filtering parameters in JSON for searching based on location, IoT system, service type, or a specific ontology.
	
	Example format:
	{
	"location": { "long": "x", "lat": "y", "range": "r" } // or polygon definition
	"IoT_system": { "systemId": "sid", "componentId":"cid" } // cid is optional
	"ServiceType": "type" // type taxonomy that groups services
	"Ontology": "o" // specific ontology from the VITAL unified data model
	}

Output	List of services in JSON-LD format

Table 21. VOB - Service registration management

	Service registration management
Description	VOB registers or de-registers the orchestrated services that it manages / offers via the VUAIs to applications or via the internal REST API to other VITAL modules.
URL	BASE_URL/serviceRegistration
Method	POST

Input	Service metadata, action {register|de-register}
	
	Example format:
	{
	"serviceDescription": { "name": "…", "URI": "…", "type":"…", … }
	"accessPolicy": { "role1": "RW", "role2":"R", … }
	"action": "register | de-register"
	}

Output	Status code {success with serviceID & metadata | failure} in JSON-LD format

The VOB will work also using supplementary interfaces above defined, such as Get ICOs (see Table 10) and Get IoTSystems (see

Table 16).

6 CONCLUSION AND NEXT STEPS

In this first release of this document, we have introduced the Service Discovery (SD) module in the VITAL architecture. We have positioned it within the global architecture, presented the requirements and expectations of the module and explained our reasoning for the technical implementation choices we have made.

This document also describes the main interfaces and the current status of the service, setting out the next steps and the directions we will follow to deliver the full service.

We have chosen an iterative implementation, progressively adding more services and allowing the discovery of more item types based on classification of ICOs, flows and services. In order to enhance the service that can be provided by the VITAL framework, the Service Discovery will embed different predictive model mechanisms, miming approaches such as the ones in [BJR+08], [MTBSS+12], and [MSTS+13].

All these mechanisms are being investigated, designed and evaluated by VITAL Consortium partners; the next release of this document will be enriched by descriptions of these mechanisms in the context of the overall project development.

7 REFERENCES

[BJR+08] G. Box, G.M. Jenkins, and G.C. Reinsel, “Time Series Analysis: Forecasting and Control”, fourth edition, Wiley, 2008.

[MTBSS+12] M. Marchini, M. Tortonesi, G. Benincasa, N. Suri, C. Stefanelli, “Prediction Peer Interactions for Opportunistic Information Dissemination Protocols”, in IEEE Symposium on Computers and Communications (ISCC), July 2012.

[MSTS+13] A. Morelli, C. Stefanelli, M. Tortonesi, N. Suri, “Mobility Pattern prediction to Support Opportunistic Networking in Smart Cities”, in International Conference on Mobile Wireless Middleware, Operating Systems and Applications (MOBILWARE), November 2013.

[PLM+14] R. Petrolo, V. Loscrí, N. Mitton, “Towards a Smart City based on Cloud of Things,” in ACM International MobiHoc Workshop on Wireless and Mobile Technologies for Smart Cities (WiMobCity), August 2014.

[PLM2+14] R. Petrolo, V. Loscrí, N. Mitton, “Towards a Smart City based on Cloud of Things,” in IEEE Multimedia Communications Technical Committee (MMTC), vol 9., Number 5, Special Issue on: Technologies, Services and Applications for Smart Cities, September 2014.

[RFielding00] R. Thomas Fielding, “Architectural Styles and the Design of Network-Based Software Architectures”, Ph.D. Dissertation, 2000, http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[Vital-2.2-2014] J. Soldatos, J. Kaldis et al. Vital Project, Deliverable D2.2, “Reference and Validation Scenarios for IoT virtualization”, April 2014.

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

[Vital-D3.1.1-2014] G. Schiele, T. Geraghty et al. Vital Project, Deliverable D3.1.1, “Virtual Models, Data and Metadata for ICOs V1”, September 2014.

