HORIZONS 2020 PROGRAMME

Research and Innovation Action – FIRE Initiative

Call Identifier:	H2020–ICT–2014–1
Project Number:	643943
Project Acronym:	FIESTA-IoT
Project Title:	Federated Interoperable Semantic IoT/cloud Testbeds and Applications

Specification and Implementation of Common Testbed Interfaces V1

Document Id:	FIESTAIoT-D33-160704-Draft
File Name:	FIESTAIoT-D33-160704-Draft.pdf
Document reference:	Deliverable 3.3
Version:	Draft
Editor:	Nikos Kefalakis
Organisation:	Athens Information Technology
Date:	04 / 07 / 2016
Document type:	Report, Other
Dissemination level:	PU

DOCUMENT HISTORY

Rev.	Author(s)	Organisation(s)	Date	Comments
V01	Nikos Kefalakis	AIT	2015/11/03	Initial ToC Proposal
V02	David Gomez	UNICAN	2015/11/10	Provided UNICAN testbed description
V03	Tarek Elsaleh	UNIS	2015/11/25	Provided UNIS testbed description and NGSI
V04	Nikos Kefalakis	AIT	2016/03/29	Initial TPI API description and architecture.
V05	Amelie Gyrard	NUIG	2016/04/08	Section OGC SWE, Section MQTT, Section CoAP
V06	Katerina Pechlivanidou	AIT	2016/04/15	Section 6 Background Implementation Technologies
V07	Nikos Kefalakis	AIT	2016/04/20	Updated TPI TPS, DMS, SRR APIs
V08	David Gomez	UNICAN	2016/04/26	Added info for Class I Query Services API
V09	Tarek Elsaleh	UNIS	2016/05/04	Added Semantic Registry API definition
V10	Nikos Kefalakis	AIT	2016/05/20	Added OpenIoT and VITAL platform descriptions
V11	David Gomez	UNICAN	2016/05/25	Updated Class I Testbed services
V12	Nikos Kefalakis	AIT	2016/06/10	Added TPI TPS and DMS installation instructions
V13	Minwoo Ryu	KETI	2016/06/16	Added KETI testbed information and oneM2M
V14	Konstantinos Bountouris	Com4Innov	2016/06/17	Testbed interface and model description of Com4Innov
V15	Katerina Pechlivanidou, Nikos Kefalakis	AIT	2016/06/14	Added TPI configurator descreption
V16	David Gomez	UNICAN	2016/06/19	Updated Semantic Registry and Class I Query Services APIs
V17	Katerina Pechlivanidou	AIT	2016/06/20	Updated TPI TPS and DMS APIs
V18	Nikos Kefalakis, Katerina Pechlivanidou	AIT	2016/06/21	Merged and updated various sections. Prepare the document for internal review

V19	John Soldatos	AIT	2016/06/22	Conclusions, Executive Summary
V20	Paul Grace	ITINNOV	2016/06/23	Internal Quality review
V21	Amelie Gyrard	NUIG	2016/06/30	Internal Technical review
V22	Alexander Willner	FHG	2016/07/02	Internal Technical review
V23	Nikos Kefalakis	AIT	2016/07/04	Updates base on Technical & Quality Review
V24	Nikos Kefalakis	AIT	2016/07/04	Final Version for Submission
V25	Nikos Kefalakis	AIT	2016/07/04	Circulated for Approval
Draft	Martin Serrano	NUIG	2016/07/10	EC Submitted

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TERMS AND ACRONYMS

3G	Third Generation of mobile telecommunication technology
API	Application Program Interface
dB	Decibel
DSL	Domain Specific Language
EaaS	Experiment as a Service
GPS	Global Positioning System
GUI	Graphical User Interface
ICO	Internet Connected Object
IoT	Internet of Things
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
LOD	Linked Open Data
LOV	Linked Open Vocabularies
LOV4IoT	Linked Open Vocabularies for Internet of Things (LOV4IoT)
OTAP	Over The Air Programing
SEL	Service Experimentation Layer
SSC	Super Stream Collider
SSN	Semantic Sensor Networks
urn	Uniform Resource Name
VE	Virtual Entity
CoAP	Constrained Application Protocol
MQTT	MQ Telemetry Transport
SWE	Sensor Web Enablement
JMS	Java Message Server
XA	eXtended Architecture
MQTT	Message Queuing Telemetry Transport
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
NIO	Non-blocking I/O
3G	Third Generation of mobile telecommunication technology
AMQP	Advanced Message Queuing Protocol

API	Application Program Interface
API	Application Program Interface
CoAP	Constrained Application Protocol
dB	Decibel
DSL	Domain Specific Language
EaaS	Experiment as a Service
ESB	Enterprise Service Bus
FTP	File Transfer Protocol
GPS	Global Positioning System
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICO	Internet Connected Object
IoT	Internet of Things
JBI	Java Business Integration
JDBC	Java Database Connectivity
JMS	Java Message Server
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
LOD	Linked Open Data
LOV	Linked Open Vocabularies
LOV4IoT	Linked Open Vocabularies for Internet of Things (LOV4IoT)
LPWA	Low-Power Wide-Area Network
LTE	Long Term Evolution
M2M	Machine to Machine
MQTT	Message Queuing Telemetry Transport
NGSI	Next Generation Service Interface
NIO	Non-blocking I/O
OSGi	Open Services Gateway initiative
OTAP	Over The Air Programing
PPIs	Platform Provider Interfaces
SEL	Service Experimentation Layer
SOA	Service-Oriented Architecture
SSC	Super Stream Collider

SSL	Secure Sockets Layer
SSN	Semantic Sensor Networks
SWE	Sensor Web Enablement
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
urn	Uniform Resource Name
VE	Virtual Entity
WSDL	Web Services Description Language
XA	eXtended Architecture
XML	Extensible Markup Language
XSD	XML Schema Definition

1 POSITIONING

Note that sections 1.2 to 1.5 below are repeated in all WP3 deliverables in order to keep each deliverable self-contained.

1.1 Executive Summary

Among the main goals of the FIESTA-IoT is to enable IoT researchers and experimenters to specify and execute data-intensive IoT experiments over multiple IoT testbeds i.e. testbed agnostic experiments. To this end, FIESTA-IoT is building an experimental infrastructure, which provides the means for flexibly integrating data and services from multiple IoT testbed, while at the same time facilitating the configuration and management of these testbed. This deliverable is dedicated to the specification and implementation of interfaces to IoT testbeds, as required in order to facilitate testbed owners to integrate their testbeds in FIESTA-IoT, but also in order to enable researchers to execute IoT testbed agnostic experiments. The specification process considers the main functionalities and properties, which should be exposed by IoT testbeds in order to facilitate their integration within FIESTA-IoT for the purposes of testbed agnostic experimentation. Likewise, the specified interfaces have been implemented for the testbeds of the FIESTA-IoT partners in Spain, France and Korea, thus ensuring an initial prototype implementation of the specification along with its validation in real-life conditions.

As its ultimate target, the deliverable ends-up presenting a Testbed Provider Interface (TPI) specification, along with its prototype implementation in FIESTA-IoT partners' testbeds. The TPI specification considers and comprises two different layers:

- A Configuration & Management layer that runs at the FIESTA-IoT platform side and controls the functionality of the TPI by utilizing the offered user interface for the User (Testbed provider).
- A Testbed Provider Interface (TPI) API, as part of which the Testbed provider has to implement a list of predefined services that enables the management and manipulation of the offered data.

The TPI specification has been driven by various requirements, including flexibility and ease in the integration of testbeds, support of maintstream IoT standards for data and services representation, compatibility with the FIESTA-IoT testbeds and more. To address these requirements, the deliverable has considered the properties and standards that are supported by the FIESTA-IoT testbeds, including for example the oneM2M standards supported by KETI's testbed and NGSI standards supported by Comm4Innov testbed. Furthermore, platform agnostic interfaces specified and implemented as part of other projects (e.g., FP7 OpenIoT, FP7 VITAL) have been also considered since they provide valuable insights on the nature and properties of a platform/testbed agnostic TPI. Thus, the deliverable includes also a review of existing testbeds and platform-agnostic interfaces, as a key element of the methodology that led to the TPI specification.

In addition to the TPI specification, the deliverable presents also the technology selected for its implementation, including the ActiveMQ messaging middleware framework and the Quartz job scheduling framework. The selection of these

technologies has been performed following a thorough review of other candidate technologies for services integration and scheduling.

As part of the deliverable we also illustrate the prototype implementation of the TPI. The TPI will be a key element of the FIESTA-IoT platform as it will drive both the integration of IoT testbeds in the experimental infrastructure and the execution of data-driven platform agnostic IoT experiments. As such it will be also a key element of the infrastructure that will support the open call processes of the project.

1.2 FIESTA-IoT Scope

Recent advances in the Internet of Things (IoT) area have progressively moved in different directions i.e. designing technology, deploying the systems into the cloud, increasing the number of inter-connected entities, improving the collection of information in real-time, and no less important—the security aspects in IoT. IoT advances have drawn a common grand challenge that focuses on the integration of the heterogeneous IoT generated data. This key challenge is to provide a common sharing model or a set of models organizing the information coming from the connected IoT services, IoT technology and systems, and more important, to be able to offer them as experimental services in order to optimize the design of new IoT systems and facilitate the generation of solutions more rapidly.

In FIESTA-IoT we focus on the problem of formulating and managing IoT data from heterogeneous systems and environments and their entity resources (such as smart devices, sensors, actuators, etc.), this vision of integrating IoT platforms, Testbeds and their associated silo applications within cloud infrastructures is related to several scientific challenges, such as the need to aggregate and ensure the interoperability of data streams stemming from different IoT platforms or Testbeds, as well as the need to provide tools and techniques for building applications that horizontally integrate diverse IoT Solutions. The convergence of IoT with cloud computing is a key enabler for this integration and interoperability, since it allows the aggregation of multiple IoT data streams towards the development and deployment of scalable, elastic and reliable applications that are delivered on-demand according to a pay-as-you-go model.

The activity in FIESTA-IoT is distributed in 7 Work Packages (WP). WP1 is dedicated to the project activities coordination, considering consortium administration, financial management, activity co-ordination, reporting and quality control. In FIESTA-IoT one of the main objectives is to include experimenters and new Testbeds to test and provide feedback about the platform and tools, thus open calls for those tenders will be issued (these are also part of the WP1 activity and it is called selection of third-parties).

WP2 focuses on stakeholder's requirements and the analysis of IoT platforms and Testbeds in order to define strategies for the definition and inclusion of experiments, tools and Key Performance Indicators (KPIs). The activities in WP2 are focused on studying the IoT platforms and Testbeds and the specification of the experiments, the detail of the needed tools for experimentation, and the KPIs for validating the proposed solutions. This WP will conduct the design and development of the Meta-Cloud Architecture (including the relevant directory of IoT resources) and will define the technical specification of the project. WP2 also focuses on analysing the Global Market Confidence program and establishes the Certification Program Specifications

that will drive the global market confidence and certification actions around the IoT experimentation model.

WP3 focuses on providing technologies, interfaces, methods and solutions to represent the device and network nodes of the Testbeds as virtualized resources. The virtualized resources will be represented as services and will be accessible via common service interfaces and Application Program Interfaces - APIs (i.e. the FIESTA-IoT Testbed interfaces/APIs). The virtualized resources and their capabilities and interfaces will be also described using semantic metadata to enable (semi-) automated discovery, selection and access to the Testbed devices and resources.

WP4 will implement an infrastructure for accessing data and services from multiple distributed diverse Testbeds in a secure and Testbed agnostic way. To this end, it will rely on the semantic interoperability of the various Testbeds (realized in WP3) and implement a single entry point for accessing the FIESTA-IoT data and resources in a seamless way and according to an on-demand Experimentation-as-a-Service (EaaS) model. The infrastructure to be implemented will be deployed in a cloud environment and will be accessible through a unified portal infrastructure.

WP5 focuses on designing, deploying and delivering a set of experiments, so as to assess the feasibility and applicability of the integration and federation techniques, procedures and functions developed during the project lifetime. It will define a complete set of experiments to test the developments coming from other WPs (mainly WP3 and WP4), covering all of the specifications and requirements of WP2. Developments will be tested over available IoT environments and/or smart cities platforms. WP5 will also provide evaluation of the KPIs defined for every experiment/pilot. The final deployed experiments will include a subset of those coming from WP2, WP3 and WP4, as well as those provided by FIESTA-IoT Open Calls.

WP6 focuses on the establishment and validation of the project's global market confidence on IoT interoperability, which will provide a vehicle for the sustainability and wider use of the project's results. The main activity in this WP focuses on specifying and designing an IoT interoperability program, including a set of well-defined processes that will facilitate the participation of researchers and enterprises. WP6 works on providing a range of certification and compliance tools, aimed at auditing and ensuring the openness and interoperability of IoT platforms and technologies. WP6 also focuses on interoperability testing and validation and to provide training, consulting and support services to the FIESTA-IoT participants in order to facilitate platforms and tool usability, but also to maximize the value offered to them by using FIESTA-IoT suite and tools.

WP7 focuses on ensuring that the FIESTA-IoT suite, models and tools engage well with the community outside of the project; from promotion and engagement of new customers, to the front line support of current users, and the long-term exploitation of results and sustainability of the facility itself. This will be carried out in a coordinated manner such that a consistent message and professional service is maintained. Dissemination activities and the KPI to measure the impacts will be studied and used in this WP. An ecosystem plan including the specification of processes, responsibilities and targets will be generated and the evaluation and effectiveness of the operating model will be evaluated within this WP. In this WP the successes of stakeholder engagement and reports on their satisfaction with the services offered in FIESTA-IoT will be put in place at the end of the project.

1.3 WP3 overview

This work package focuses on providing technologies, interfaces, methods and solutions to represent the device and network nodes of the test-beds as virtualized resources. The virtualized resources will be represented as services and will be accessible via common service interfaces and APIs (i.e. the FIESTA-IoT Testbed interfaces/APIs). The virtualized resources and their capabilities and interfaces will be also described using semantic metadata to enable (semi-) automated discovery, selection and access to the Testbed devices and resources. The virtualized resources will enable access to the devices for management and configuration proposes, will allow sending actuation commands and feedback to the test-bed devices and will also provide interfaces to the streaming data emerging from the resources. The data streams will represent the observation and measurement data that is collected by the Testbed resources. With respect to FIESTA-IoT objectives, WP3 main objectives are to:

- provide an higher-level service abstraction and virtualized representations of the Testbed resources;
- enable access and configuration of the Testbed resources using common interfaces;
- develop (semantic) annotation models for interoperable data/service exchange between the Testbeds and also between Testbeds and higher-level services/applications;
- enable (semantic) stream reasoning, provide re-usable components and develop common methods for data analytics for data streams;
- provide common interfaces, and command and control mechanisms for real-time management and configuration of the streams.

The activities in this work package first start with developing common models and (semantic) annotation frameworks to describe the Testbed resources. The activities will also focus on providing common interfaces to access the Testbed resources, represent their streaming data and exchange configuration, control and actuation commands with the Testbed resources. Developing and/or extending common description models for the services and the IoT data provide semantic interoperability for the services and the data streams. The work package activities also provide re-usable software components, tools and mechanisms to process the streaming IoT data and to manage the stream using real-time or near real-time control and monitoring mechanisms.

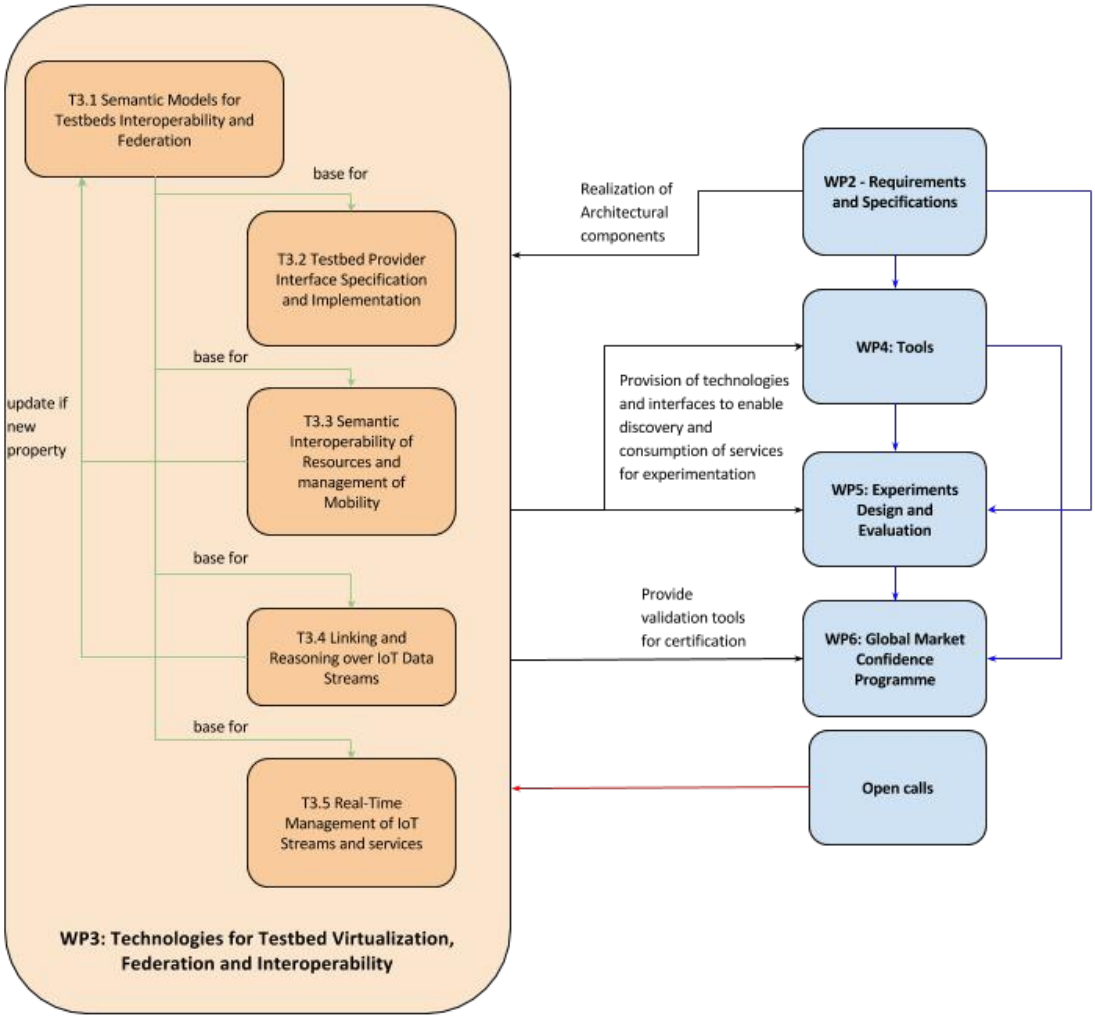

Figure 1 WP3 relation with different work packages

The WP3 Tasks cross all aspects of the FIESTA-IoT Infrastructure and links with different work packages (See Figure 1). The set of Tasks (T) within WP3 are:

- **T3.1 Semantic Models for Testbeds Interoperability and Federation**: This task aims to develop common semantic description models to describe the Testbed resources, services and the IoT data. There are several existing models for describing the IoT data, services and network resources. Some of the existing Testbeds have also adapted various description models and interfaces for their Testbeds. The activities in this context mainly focus on re-using the existing common models, extending and/or adapting the common description frameworks and providing optimum, customizable, extensible and modular semantic description framework to enable interoperability among different Testbeds and data/service providers. The semantic models are provided for Testbed devices, IoT data and data streams. A set of tools and (semi-automated) mechanisms will be developed to enable federation of resource and creation of composite data/services using multiple resources. The semantic description model developed in this activity is used in T3.3 for interoperability test and evaluation.

- **T3.2 FIESTA-IoT Testbed Provider Interfaces Specification and Implementation**: This task is devoted to developing common interfaces in order to access and exchange data and commands between the Testbed resources and higher-level services and applications. For the interface development, existing common models and enablers such as Next Generation Service Interfaces (NGSI) and the services models that are developed in the FP7 IoT-A project as well as the common interfaces that are currently been used in the existing Testbeds will be investigated. This activity will then provide a set of common interface specification with a focus on compatibility with the most common technologies (e.g. RESTful, supporting technologies such as Constrained Application Protocol (CoAP)) and existing interfaces (e.g., NGSI, Internet of Things-Architecture (IoT-A)) and will also provide a reference implementation of those interfaces on the Testbeds. Furthermore, the adaptation of standards-based cloud interfaces (such as Open Cloud Computing Interface (OCCI), SNIA's Cloud Data Management Interface (CDMI), DeltaCloud APIs) will be enhanced, adapted and used for accessing IoT clouds. Using common interfaces in collaboration with the common semantic description models in T3.1 will enable interoperability and flexible integration of the resources among heterogeneous Testbeds.

- **T3.3 Semantic Interoperability of Resources and Management of Mobility**: This task will provide tools, interfaces and will describe best practices to publish data and services according to the semantic models that will be developed in this activity. A set of tools and interfaces will be also provided to enable a third-party developer to test and evaluate their semantic description against the provided models and to verify their semantic interoperability and/or receive feedback to enhance and adapt their models. We will also provide mechanisms for different serialization formats and we will also evaluate the efficiency, size and complexity of various representations according to different use-case scenarios. The mobility of the resources, updating the description of the pervasive resources and supporting access and subscription to the mobile resources via the Testbeds will be also investigated in this activity. Mechanisms to enable management of the mobile resources will be provided to enable access and hand-over between different gateway/Testbed resources in mobile scenarios.
- **T3.4 Linking and Reasoning over IoT Data Streams**: This task will focus on data analytics and developing mechanisms to process the real-time dynamic IoT data streams. The data analytics method will initially focus on stream processing and creating patterns, data abstractions from real-time data that emerges from the test-bed as well as processing the static and historical data. We will investigate the association and reasoning methods and will develop solutions to detect different correlations between the patterns in the streams and also multiple and multi-modal data streams. The multi-modality of data and linking between different data streams will be a key challenge to solve and for this purpose we will use knowledge-based solutions (that will also utilize the semantic descriptions provided in T3.1) and reasoning mechanisms; as well as machine learning techniques to process and interpret the data. The

information extraction techniques will provide actionable knowledge that can be used by higher-level services and applications. The main focus will be on providing the developed methods and techniques as re-usable and common components with a set of common interfaces that can be used over various Testbed and virtualized data streams and can be also called by different higher-level services and applications.

- **T3.5 Real-Time Management of IoT Streams and Services**: This task will develop a set of mechanisms and interfaces to manage the IoT streams and to submit/receive the control and feedback commands to/from the lower-level resources that provide the IoT data. The control and feedback command and management mechanisms will enable adapting and changing the data access, publication and in-network processing parameters in (near) real-time by considering the device level and Testbed level resources such as changing the data collection and communication frequency when a device has low battery, providing in-network abstraction, aggregation to decease the emerging data traffic, changing the sampling frequency or setting neighbouring devices to on/off based on the resolution that is required. These activities will require a set of management components and common command and feedback mechanisms and also intelligent learning and adaptation methods to enable automated and real-time control of the IoT devices and services.

The deliverables of this WP are depicted in

Table **1** below:

Table 1 WP3 Deliverables

No.	Deliverable	Related Task	Responsible Partner	Contributors
D3.1	Semantic models for Testbeds, interoperability and mobility support, and best practices	T3.1 T3.3	Inria, UNIS	NUIG-DERI, AIT, KETI, UNICAN, FOCUS
D3.2	Specification and implementation of common Testbed interfaces	T3.2	AIT	UNICAN, UNIS, NUIG-DERI, Com4Innov
D3.3	Concept and Development for IoT data analytics and IoT stream and service management	T3.4 T3.5	NUIG-DERI, UNIS	UNICAN, KETI, Inria

1.4 Audience

This deliverable addresses following audiences:

- **Researchers and engineers within the FIESTA-IoT consortium** will take into account various requirements in order to research, design and implement the APIs needed to support Testbeds associated to FIESTA-IoT Platform.

- **Testbed owners who wish to join FIESTA-IoT** will be able to use the tools to annotate the data their Testbed is producing. These annotation tools should comply with the semantic model proposed within FIESTA-IoT. By doing so, the Testbed can either become Class I, Class II, or Class III Testbed (see [1] for the definitions of various classes of Testbeds).
- **Experiment owners who wish to join FIESTA-IoT** will be able to understand how and what IoT data is stored within the FIESTA-IoT Meta Cloud and thus would be able to align their experiments that could utilize such data.
- **Researchers on Future Internet Research and Experimentation (FIRE) focusing on semantically storing data produced by their experiments** will find guidelines to store data produced by their experiments in a semantic manner either in their own repository or utilizing the FIESTA-IoT platform. The researchers will be able use the ontology and the tools as the reference. Further, if they wish to extend/modify the ontology and tools for their own research, they will be able to do so.
- **Members of other Internet of Things (IoT) communities and projects (such as projects of the IERC cluster)** can take this document as an initial reference or inspiration to design and implement their own Testbed that also stores data that is semantically annotated.
- **Open call** participants will be able to understand better the technical details needed for them to join the FIESTA-IoT consortium.
- **Standardization bodies** will have access to this deliverable as it will be a public document and therefore the ontology developed can be standardized following the involvement and reach a wider adoption.

1.5 Terminology and Definition

This sub-section intends to clarify the terminology used during this project. This initial step intends to clarify all of the important terms used, in order to minimize misunderstandings when referring to specific parts involved in the generation of data and the FIESTA-IoT concepts. The following definitions (listed in the Table below) were set regarding the domain area of FIESTA-IoT, and so are aligned with terminologies used in the Future Internet Research and Experimentation (FIRE) community and in reference to IoT-related projects (such as IoT-A).

Table 2 Terminology and Definitions table

Term	Definition
Device	Technical physical component (hardware) with communication capabilities to other Information Technology (IT) systems. A device can be attached to, or embedded inside a physical entity, or monitor a physical entity in its vicinity [5]. The device could be: • **Sensor**: A sensor is a special device that perceives certain characteristics of the real world and transfers them into a digital representation [6]. • **Actuator**: An actuator is a mechanical device for moving or controlling a mechanism or system. It takes energy, usually transported by air, electric current, or liquid, and converts that into some kind of motion [6].

Discovery	Discovery is a service to find unknown resources/entities/services based on a rough specification of the desired result. It may be utilized by a human or another service. Credentials for authorization are considered when executing the discovery [5].
Domain	Refers to an application area where the meaning of data corresponds to the same semantic context. For instance, pressure in Water Management Domain may refer to water pressure on pipes while in Air Quality Domain it refers to atmospheric pressure.
Information	Content of communication; data and metadata describing data. The material basis is raw data, which is processed into relevant information, including source information (e.g., analogue and state information) and derived information (e.g., statistical and historical information) [7].
Measurement	The important data for the experimenter. It represents the minimum piece of information sent by a specific resource, which the experimenter needs in order to fulfil the objective of the experiment.
Metadata	The metadata is the additional information associated with the measurement, facilitating its understanding.
Physical Entity	Any physical object that is relevant from a user or application perspective [6]. Physical Entities are the objects from the real world that can be sensed and measured and they are virtualized in cyber-space using Virtual Entities.
Requirement	A quantitative statement of business-need that must be met by a particular architecture or work package [8].
Resource	Computational element that gives access to information about or actuation capabilities on a Physical Entity [6].
Testbed	A Testbed is an environment that allows experimentation and testing for research and development products. A Testbed provides a rigorous, transparent and replicable environment for experimentation and testing [9].
Federated Testbeds	A Testbed federation or federated Testbeds is the interconnection of two or more independent Testbeds for the creation of a richer environment for experimentation and testing, and for the increased multilateral benefit of the users of the individual independent Testbeds [9].
Interoperability	The ability of two or more systems or components to exchange information and use the information that has been exchanged [10].
Experiment	Experiment is a test under controlled conditions that is made to demonstrate a known truth, examine the validity of a hypothesis, or determine the efficacy of something previously untried [11].
Semantic Interoperability	Semantic interoperability is the ability of computer systems to exchange data with unambiguous, shared meaning. Semantic interoperability is a requirement to enable machine computable logic, inference, knowledge discovery, and data federation between information systems.
Service	Services (Technology) are services designed to facilitate the use of technology by end users. These services provide specialized technology-oriented solutions by combining the processes/functions of software, hardware, networks, telecommunications and electronics.
Virtual Entity	Computational or data element representing a Physical Entity. Virtual Entities can be either Active or Passive Digital Entities [5].

2 BACKGROUND TECHNOLOGIES / AVAILABLE INTERFACE ANALYSIS

In the following sections, we identify the background technologies, interfaces and platforms that have been studied in order to design the required interfaces and tools for plugging-in testbeds to the FIESTA-IoT platform.

2.1 Testbeds

2.1.1 SmartSantander

As was described in Deliverable 2.2 [3], the SmartSantander platform[1] relies on a proprietary format. Based on a couple of descriptive JSON schemas, both the resource descriptions and the observations/measurements stick to a particular format that is to be validated before being stored into the local repositories (based on MongoDB non-relational databases). It is worth highlighting that, as of today, the current version of all SmartSantander's data does not support any kind of semantics. Hence, one thing that will have to be carried out before injecting any data stream from SmartSantander towards the FIESTA-IoT platform: the translation or mapping between the aforementioned SmartSantander's format and that of those being defined in the FIESTA-IoT core. Whereas the readers might refer to Deliverable 2.2 [3] should they want to get more information about the former case, the answer for all the FIESTA-IoT annotation questions can be found in Deliverable 3.1 [4].

Regarding the access and interaction point between the SmartSantander and the outside world, they cater for all services through a well-known and widespread REST interface. Namely, it provides access to the different resources/devices, including characteristics and available data (e.g. latest observations, historical data, etc.).

The access to the SmartSantander Service Experimentation Layer (SEL) platform functionalities is enabled through a REST interface that gives access to the different resources/devices, including characteristics and available data (i.e. historical and last values).

Each of the REST resources mapped in the tree representation shown in Figure 2 enables access to the different SmartSantander testbed functionalities.

The organization of the API allows multiple ways of accessing the platform functionalities depending on the necessities and focus from the experimenters. In terms of IoT data access, the /subscriptions and /queries resources enable access to the above described subscriptions and complex queries respectively. As it has also been introduced when describing the Service Proxy module [3], the same API is used for inserting data coming from the IoT infrastructure through the /observations resource. Other API resources deal with other platform functionalities out of the scope of this paper.

[1] http://smartsantander.eu/

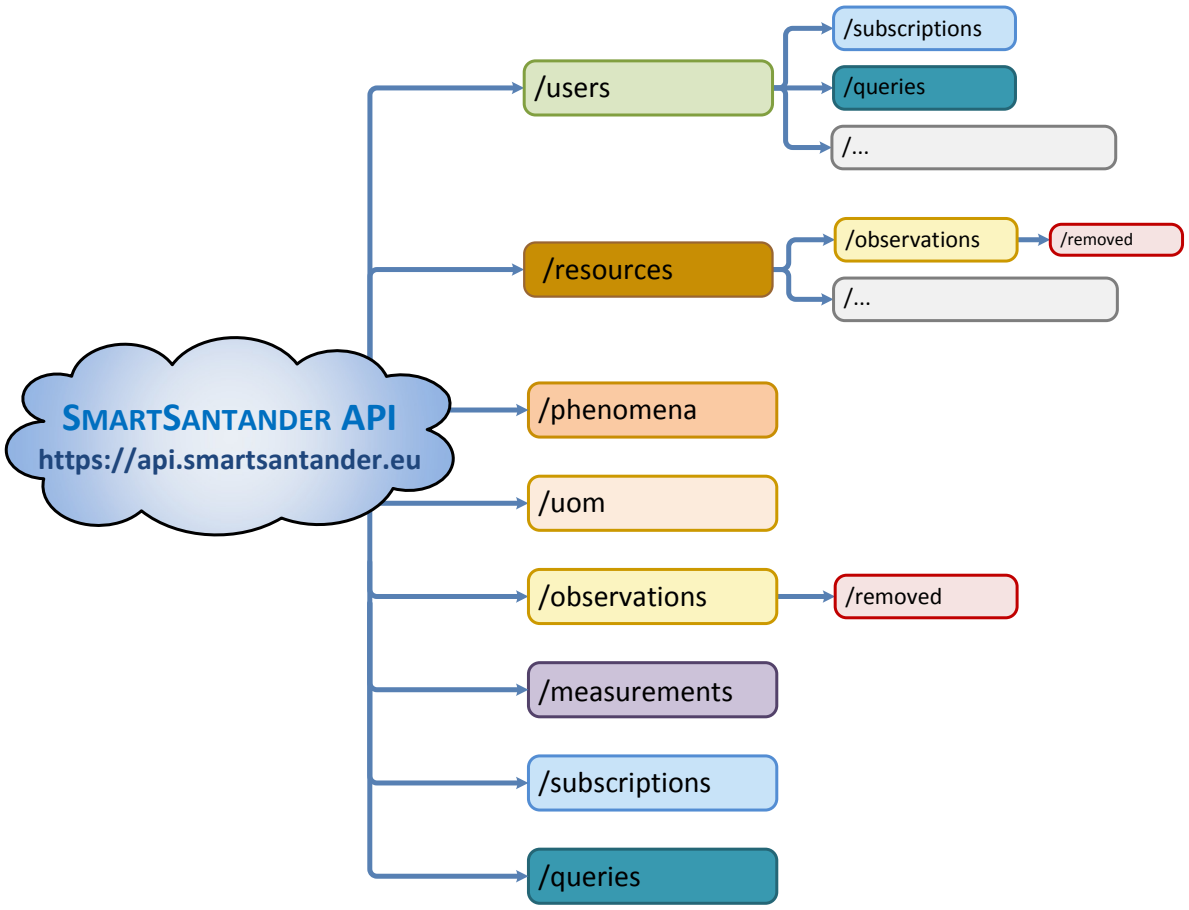

Figure 2 SmartSantander IoT API resources tree

In addition to this, we list below some of the main features brought about by the SmartSantander IoT-API:

- The organization of the API allows multiple ways of accessing the information depending on the necessities and focus from the experimenters.
- The API supports both synchronous (i.e. “queries” following request-response model) as well as asynchronous (i.e. “subscriptions” following publish-subscribe model) access. As will be seen below, the implementation of both types of access will allow to easily adopt the two main gathering methods from the FIESTA-IoT platform: GetObservation and PushObservations (see Section 3.3 for a deeper description of them).
- The same API is used for both inserting and retrieving information (i.e. bidirectional API).
- Synchronous queries do not only provide access to last values but also allow accessing historic information as well.
- Complete API documentation is available at (https://api.smartsantander.eu/docs).

2.1.2 UNIS (SmartCCSR)

A set of interfaces are used in SmartCCSR, one for exposing available resources/devices and another for the retrieval of captured measurements.

2.1.2.1 Data Format

The REDUCE REST interface exposes the data in JSON or XML representations. The information is given at the node level. Each node contains a reading for ID, light, noise, proximity infrared, temperature, power consumption level (watts), and a timestamp.

The communication protocol is HTTP, and employs a REST-Easy framework for implementation of REST APIs.

- **Monitoring info data format for experiments/services**

Description:	APIs to retrieve data about energy consumption at each desk. Data can be access per node_id, floor, room.
Standard:	Non-standard – JSON or XML format are available but data are not based on existing ontology. It provides: temperature, light, presence, noise level, watts
Example:	http://131.227.23.2:8080/SmartCCSR-testbed/restful-services/REDUCE/sensors/{node_id} http://131.227.23.2:8080/SmartCCSR-testbed/restful-services/REDUCE/json/sensors/{node_id}

- **Resources naming**

Description:	APIs to retrieve a map of deployed resources
Standard:	Non-standard – JSON or XML format are available but data are not based on existing ontology. It provides nodes id and room, floor id.
Example:	http://131.227.23.2:8080/SmartCCSR-testbed/restful-services/REDUCE/map (for XML data format) http://131.227.23.2:8080/SmartCCSR-testbed/restful-services/REDUCE/json/map

2.1.2.2 Data Model/ Ontology

The Surrey Test bed uses the two following ontologies:

- **IoT-lite[2]:** The IoT-lite (see Figure 3 below) is intended to be a "lite" ontology for describing devices, sensors, services and object (Virtual Entities). It is simpler than the IoT-A ontology but keeps its "spirit". It also references a fragment of SSN (dealing with Sensor description). The following figure shows the main concepts of the ontologies and how they relate to each other.

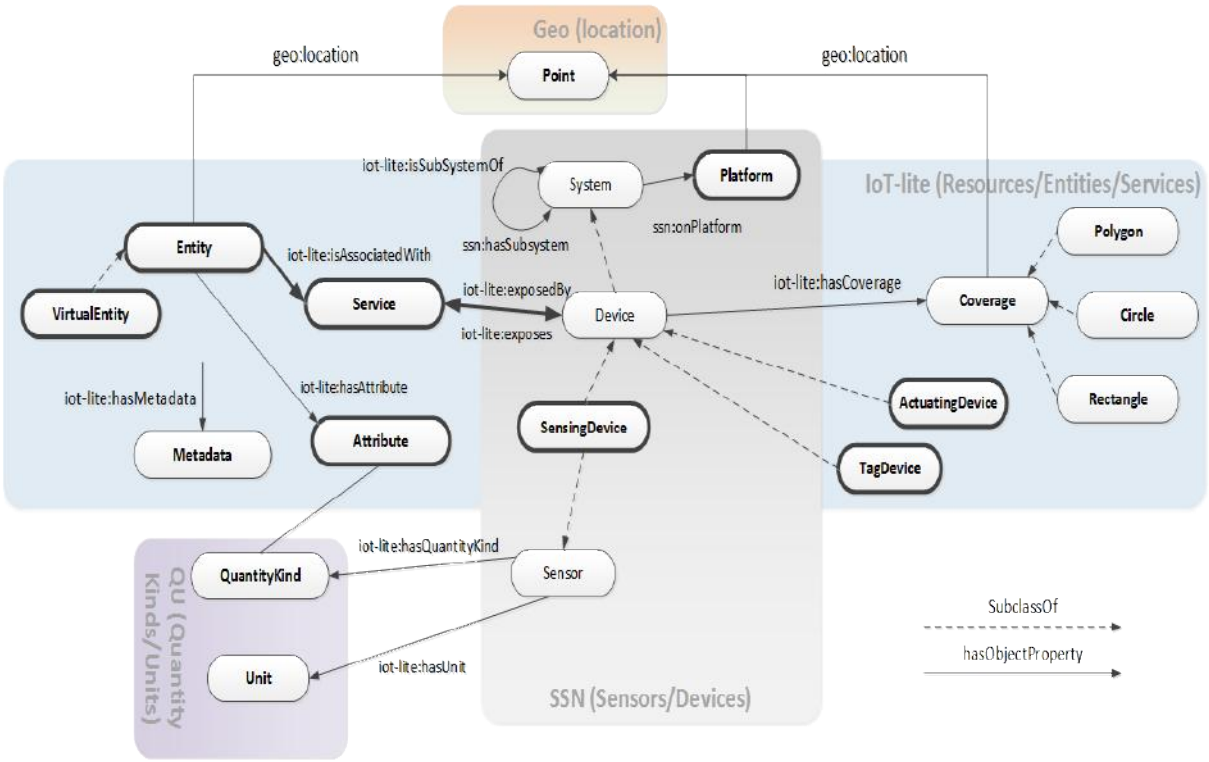

Figure 3 IoT-lite ontology

- SAO[3]: The Stream Annotation Ontology (see Figure 4) allows the representation of real-time data streams and time series. It links to additional ontologies like OWL-time and SSN:observation.

[2] http://iot.ee.surrey.ac.uk/fiware/ontologies/iot-lite

[3] http://iot.ee.surrey.ac.uk/citypulse/ontologies/sao/sao

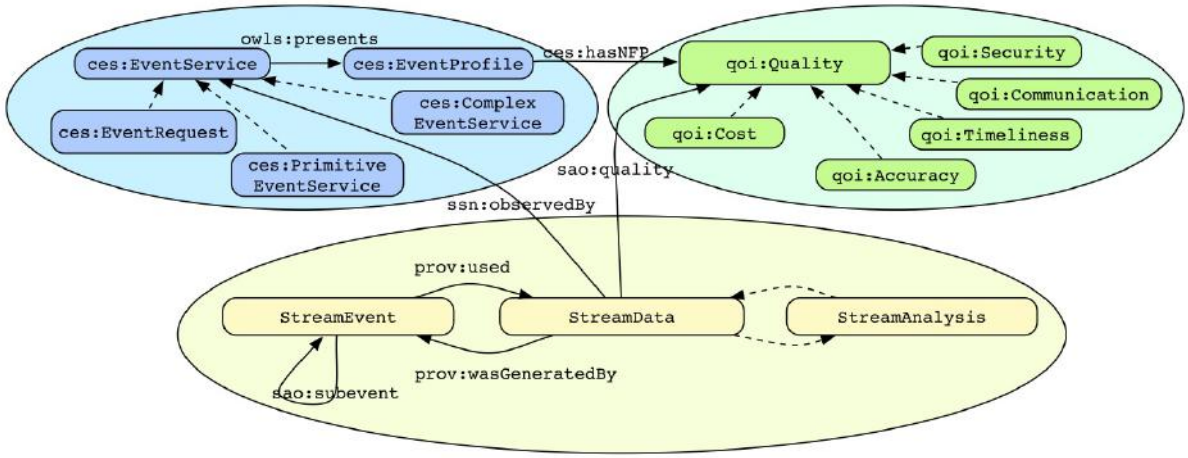

Figure 4 SAO ontology

2.1.2.3 Data Access API

Access to the SmartCCSR Testbed Experimentation Data is enabled through a REST interface (see Figure 5 below).

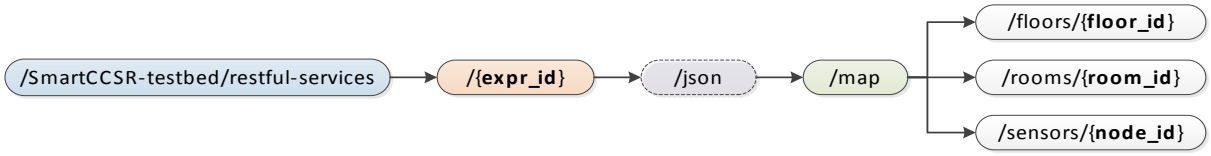

Figure 5 SmartCCSR Data API

expr_id should be replaced with the ID of the experiment required to access; currently only “REDUCE” is available. If JSON format is required then it must be stated in the URL after the **expr_id**. The “map” will present all the registered IoT nodes with the server, in the form of “nodeId” and “roomId”. Data can be presented in the context of IoT nodes on a floor, or room, or the ID of a particular IoT node.

2.1.2.4 Actuation Support

Despite the fact that the platform allows experimenters to take reserve resources and change state, e.g. measurement intervals, remotely reprogram the nodes based on the WISEBED API, the current version of the IoT-API does not contemplate this type of operations yet.

2.1.3 KETI

The KETI testbed has been implemented on the 5th floor of KETI’s building for various experimental studies. It aims to monitor and collect sensing data from a set of areas of offices (e.g., meeting area, relaxing area, work area), and office utilities in which relevant sensors are deployed.

2.1.3.1 Data Format

The KETI testbed works on Mobius[4]. The Mobius platform is implemented conforming to the oneM2M[5] standard, which applies to RESTful architectures so that Mobius can exchange data with other oneM2M platforms in a standardized format.

- **Monitoring info data format for experiments/services**

Description:	Simple Data types incorporated from XML Schema are used to describe data resources collected from IoT devices registered with Mobius platform. Resource common attributes specified in oneM2M standard are also described by W3C XML Schema Definition Language (XSD).
Standard	Standard- W3C XML Schema Definition Language (XSD) is a W3C Recommendation.

- **Resources naming**

Description:	Three resource type<AE> (application entity), <container>, and <contentInstance> are defined in oneM2M standard to describe three kind of resources.
Standard:	Standard-Each type of resource specified in oneM2M standard are also described by W3C XML Schema Definition Language (XSD), which is a W3C Recommendation.

- **Resources description format**

Description:	Three resource type<AE> (application entity), <container>, and <contentInstance> are defined in oneM2M standard to describe three kind of resources. For each resource type, resource common attributes (such as *parentID, resourceType, resourceID, creationTime,* and *typeOfContent etc.),* resource specific attributes are defined for data resource descriptions.
Standard:	Standard-Resource attributes specified in oneM2M standard are also described by W3C XML Schema Definition Language (XSD), which is a W3C Recommendation.

[4] http://iotmobius.com/

[5] http://www.onem2m.org/

2.1.3.2 Data Model/ Ontology

The KETI testbed used the oneM2M standard ontology, named Base Ontology. The Base Ontology[6] (Figure 6) has been designed with the intent to provide a minimal number of concepts, relation and restrictions that are necessary for semantic discovery of entities in the oneM2M System. To make such entities discoverable in the oneM2M System they need to be semantically described as classes in a – technology/vendor/other-standard specific – ontology and these classes need to be related to some classes of the Base Ontology as subclasses.

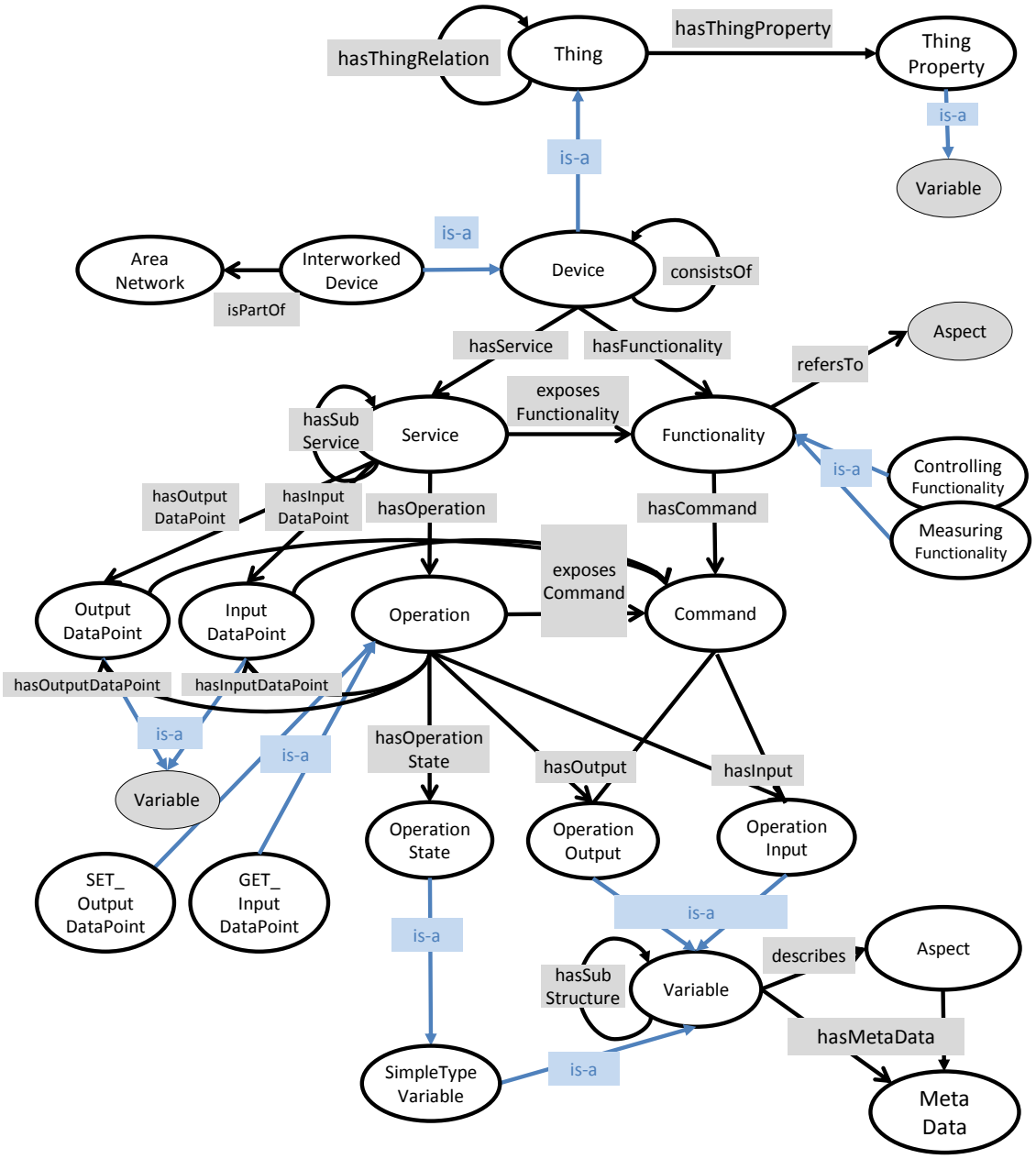

Figure 6 oneM2M BaseOntology

[6] http://www.onem2m.org/technical/onem2m-ontologies

2.1.3.3 Data Access API

Access to the KETI testbed experimentation data is enabled through an oneM2M standard interface as below:

API	oneM2M-defined APIs (using Mca. Mcc').
Description:	oneM2M provides a subscription/notification interface to access the historical data resources asynchronously. In addition, oneM2M supports data retrieval including application entity (AE), AE container, and AE content instance retrieving, which are terms defined in the oneM2M standard.

For more details on oneM2M, please visit the oneM2M official website (http://www.onem2m.org/).

2.1.3.4 Actuation Support

The KETI testbeds allows experimenters to take reserve resources using oneM2M functionalities (i.e., subscription/notification). In addition, it enables to control power on/off regarding 10 smart socket deployed in the KETI's testbed.

2.1.3.4.1 Supported Commands

In order to control power on/off regarding the smart sockets, the KETI testbed supports own commands for on and off as bellow:

- Power on: 1
- Power on: 0

These commands should be used with API defined by oneM2M as clause 2.1.3.4.3.

2.1.4 Com4Innov

Com4Innov is a unique shared environment providing the market with the means to develop and validate their technologies, applications or services in the domains of networks and mobile services of the future (4G/LTE/IMS), M2M (Machine to Machine) and communicating objects / smart devices for the future IoT. Com4Innov can produce data from various sources like 4G/5G cellular data as well as from the innovative technology long-range networks LPWA with low power consumption, i.e. LoRa and SigFox.

2.1.4.1 Data Format

Com4Innov infrastructure is free of data format. It rather transports data and information between the core network and the devices and between the devices themselves. This means that it can host multiple and different types of data.
As Com4Innov is a FIWARE collaborator, it makes use of the NGSi API (described in section 2.2.1 below), which among others has the following data structures (in the Context Management part – show in Figure 7):

- ContextElement structure: Context in FIWARE is in turn represented through context elements. A context element extends the concept of data element by associating an EntityId and EntityType to it, uniquely identifying the entity (which in turn may map to a group of entities) in the FIWARE system to which the context element information refers. In addition, there may be some attributes as well as meta-data associated to attributes that we may define as mandatory for context elements as compared to data elements.

Element name	**Element type**	**Optional**	**Description**
EntityId	EntityId	No	Identifies the Context Entity for which the Context Information is provided.
AttributeDomainName	xsd:string	Yes	Name of the attribute domain that logically groups together a set of Context Information attributes. Examples of attribute domain are: device info (battery level, screen size, ...), location info (position, civil address, ...).
ContextAttribute	ContextAttribute [0...unbounded]	Yes	List of Context Information attributes. Note: In case of the attributeDomainName is specified all contextAttribute have to belong to the same attributeDomainName.
DomainMetadata	ContextMetadata [0..unbounded]	Yes	Metadata common to all attributes of the logical domain (related to the AttributeDomain)

- ContextAttribute structure

Element name	**Element type**	**Optional**	**Description**
Name	xsd:string	No	Name of the Context Information attribute
Type	xsd:anyURI	Yes	Indicates the type of the value field
ContextValue	xsd:any	No	The actual value of the Context Information attribute

Metadata	ContextMetadata [0..unbounded]	Yes	Metadata about the Context Information attribute

- ContextMetadata structure

Element name	**Element type**	**Optional**	**Description**
Name (Timestamp, Expires, Source, ID)	xsd:string	No	Name of the metadata
Type	xsd:anyURI	Yes	Indicates the type of the value field
Value	xsd:any	No	The actual value of the metadata

- ContextRegistration structure: This structure is used in the ContextRegistrationList parameter of registerContext operation and the ContextRegistration field of the ContextRegistrationResponse structure. This structure can be used either to register/update the information about ProvidingApplication or to register/update the availability of ContextEntities and their related attributes. If the ContextRegistration structure is used to update the registered information about the ContextEntities, EntityIdList SHALL be present together with the ContextRegistrationAttribute parameter. If the ContextRegistration structure is used to register the metadata about ProvidingApplication, RegistrationMetadata SHALL be present. The registration/update of ContextEntities and ProvidingApplication can be done at the same time.

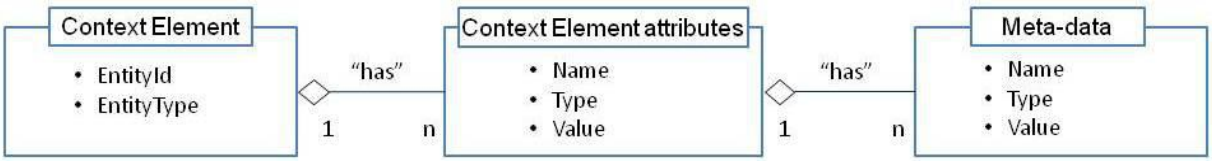

Figure 7 Context Element Structure Model

2.1.4.2Data Access API

As Com4Innov is one of the 17 FIWARE nodes, it makes use of the FIWARE NGSI Specification. The FIWARE specification is a modification of the RESTful OMA NGSI specification which aims to maximize the role of NGSI for massive data collection from the IoT world, where a myriad of IoT resources are providing context elements occurrence/updates involving low level protocols. In other words, FIWARE NGSI is mainly the binding over OMA NGSI; however, some small differences have been implemented in FIWARE NGSI with respect to OMA specification.

RESTful OMA NGSI (Next Generation Service Interface) Operations are grouped into two major interfaces:

- NGSI-10
  - updateContext
  - queryContext
  - subscribeContext / unsubscribeContext / updateContextSubscription
  - notifyContext

- NGSI-9
  - registerContext
  - discoverContextAvailability
  - subscribeContextAvailability / unsubscribeContextAvailability / updateContextAvailabilitySubscription
  - notifyContextAvailability

Com4Innov has several means offered to manage the environment:
- Capabilities to register new object
- Capabilities to trace and monitor traffic messages at different level of the network
- Capabilities to simulate traffic at different level (gateway level and sensor level)

2.2 Other Standard RESTful Interfaces

In this section, we mention other restful interfaces that where considered in the Testbed interface specification.

2.2.1 NGSI

Next Generation Service Interface (NGSI) is a suite of interfaces and components defined by the Open Mobile Alliance which enables network providers to become application platform providers whereby web application developers can exploit to create enhanced web services and applications. The interfaces that are of interest and relevance to FIESTA-IoT for the exchange of contextual information are NGSI 9 and 10.

NGSI-9 focuses on information about the availability of context information, i.e. information about actual data values from the context source are not exchanged, e.g. the reading from a temperature sensor. It focuses more on information about the context source; what value can be retrieved, specific information relating to the value e.g. resolution, accuracy, freshness etc., and how to reach it. NGSI-10 focuses on the actual contextual information.

There are two roles involved, the NGSI-9/10 server and client. A client can be a context provider or a context consumer.
For NGSI-9, a server can be a backend component which acts as a registry that allows the registration and discovery of context entities. For NGSI-10, a server can take the role of a broker of data that has been published by context producers, and consumed by users or an application querying about the data.

Each interface has three main interaction types:

- NGSI-9:
  - **registration** of context information, i.e. announcements that certain context information is available (invoked by context providers)
  - one-time queries for **discovering** hosts (also called 'agents' here) where certain context information is available
  - **subscriptions** for context availability information updates (and the corresponding **notifications**)

- NGSI-10:
  - one-time **queries** for context information
  - **subscriptions** for context information updates (and the corresponding **notifications**)
  - unsolicited **updates** (invoked by context providers)

NGSI also defines a simple information model which describes "entities" (Figure 8). An entity is a virtual representation of all kinds of physical objects in the real world. An entity would have an ID, a type, and a set of attributes. Each attribute has a name, type, value, and optional metadata. Each metadata would also have a name, type, and a value.

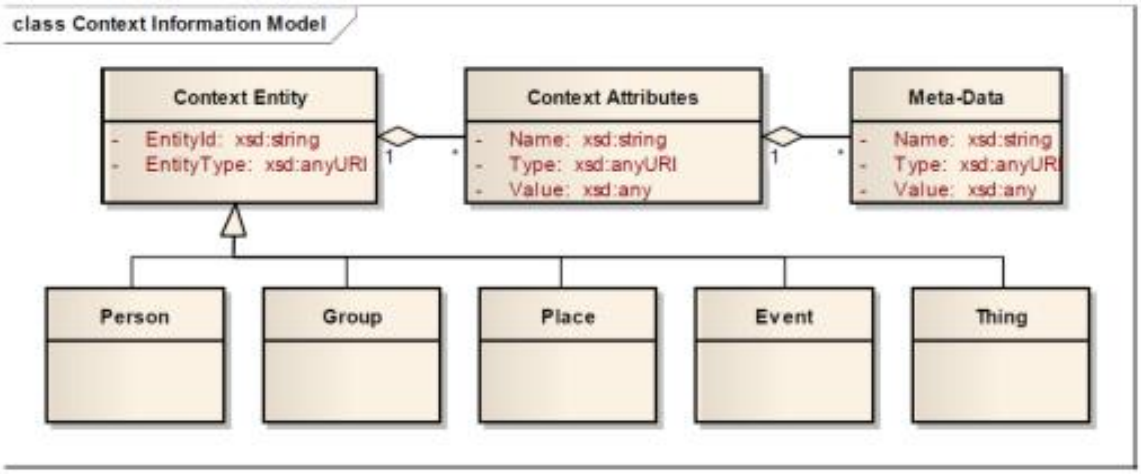

Figure 8 NGSI Context Entity Information Model

The FIWARE project has provided an implementation of the NGSI-9/10 specification which is essentially a RESTful API via HTTP. The data representation for the requests and responses is either XML or JSON. For each interface, FIWARE has provided two sets of operations (Figure 9); standard and convenience. All standard operations require requests to be defined in the body of the HTTP message. Each operation has its own structure. Convenience allows a more RESTful approach, whereby queries are appended to the URL of the request.

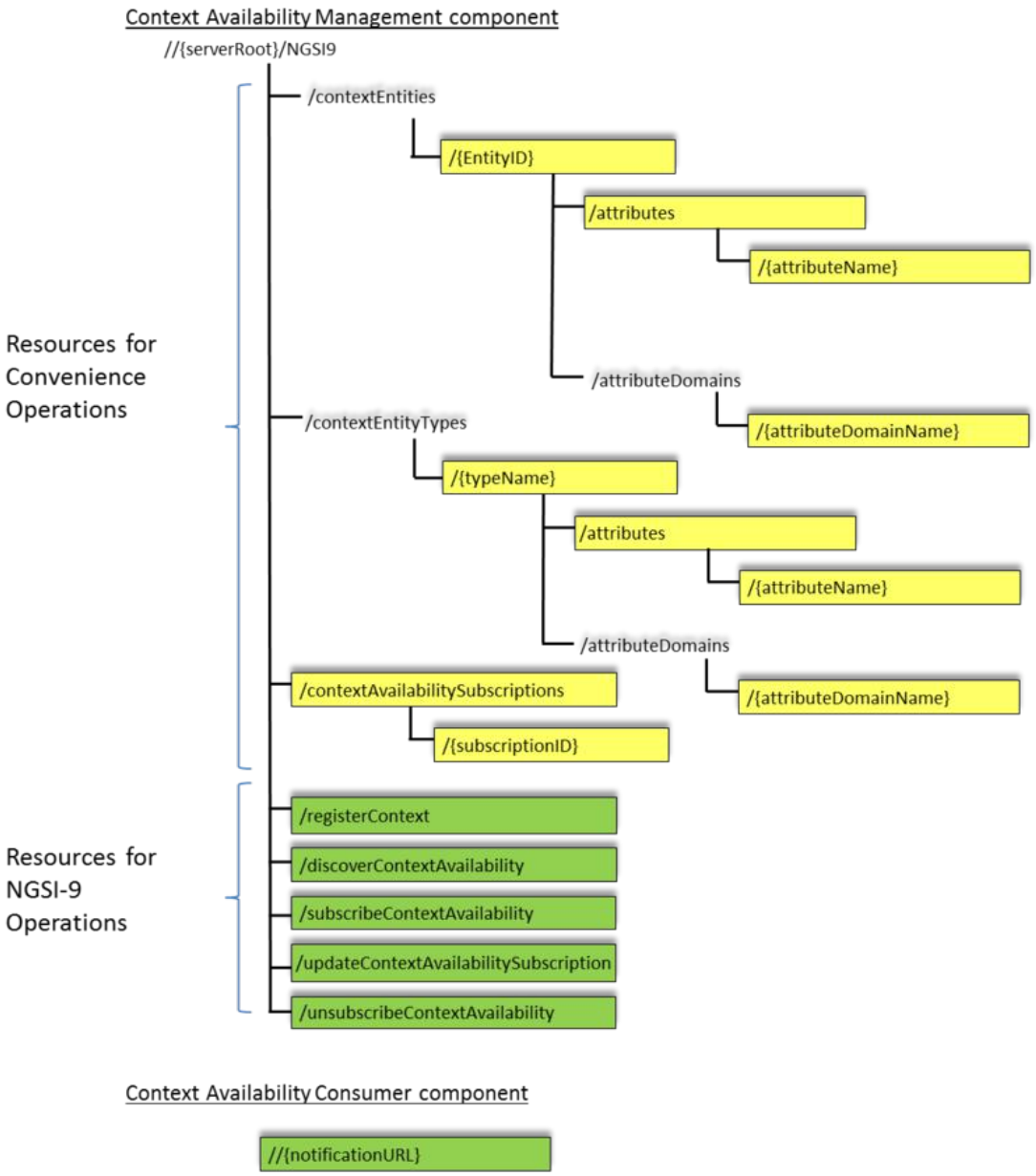

Figure 9 NGSI-9 Operations

The NGSI interface for discovering IoT entities is effective due to its simplicity and generality. However, it is also regarded as a loose binding in the sense that it does not provide a global naming convention for entities and their attributes, which will essentially affect interoperability between descriptions that might match in terms of syntax, and not in semantics. Therefore, it is planned to expose descriptions annotated in NGSI as linked open data, so that its attributes can be compared with other entities on the web, whether it is spatial, temporal or thematic.

2.2.2 OneM2M

oneM2M is a standard to develop technical specifications which address the need for a common M2M Service Layer that can be readily embedded within various hardware and software, and relied upon to connect the myriad of devices in the field with M2M application server worldwide.

The oneM2M functional architecture comprises the following functions:

- Application Entity (AE): Application Entity is an entity in the application layer that implements an M2M application service logic. Each application service logic can be resident in a number of M2M nodes and/or more than once on a single M2M node. Each execution instance of an application service logic is termed an "Application Entity" (AE) and is identified with a unique AE-ID (see clause 7.1.2). Examples of the AEs include an instance of a fleet tracking application, a remote blood sugar monitoring application, a power metering application, or a controlling application.
- 2) Common Services Entity (CSE): A Common Services Entity represents an instantiation of a set of "common service functions" of the M2M environments. Such service functions are exposed to other entities through the Mca and Mcc reference points. Reference point Mcn is used for accessing underlying Network Service Entities. Each Common Service Entity is identified with a unique CSE-ID. Examples of service functions offered by CSE include: Data Management, Device Management, M2M Service Subscription Management, and Location Services. Such "sub-functions" offered by a CSE may be logically and informatively conceptualized as Common Services Functions (CSFs). The normative Resources which implement the service functions in a CSE can be mandatory or optional.
- 3) Underlying Network Services Entity (NSE): A Network Services Entity provides services from the underlying network to the CSEs. Examples of such services include device management, location services and device triggering. No particular organization of the NSEs is assumed.

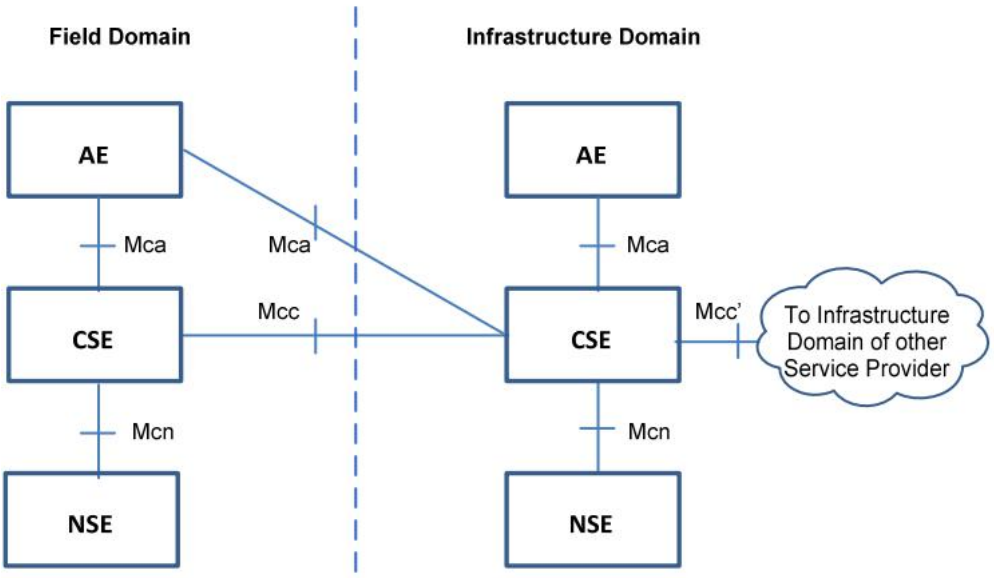

Figure 10 oneM2M Functional Architecture

In the above figure, oneM2M provides a reference which consists of one or more interfaces of any kind. The following reference points are supported by the Common Services Entity (CSE). The “Mc(-) nomenclature is based on the mnemonic “M2M communications”.

- Mca Reference Point: Communication flows between an Application Entity (AE) and a Common Services Entity (CSE) cross the Mca reference point. These flows enable the AE to use the services supported by the CSE, and the CSE to communicate with the AE.
- Mcc Reference Point: Communication flows between two Common Services Entities (CSEs) cross the Mcc reference point. These flows enable a CSE to use the services supported by another CSE.
- Mcn Reference Point: Communication flows between a Common Services Entity (CSE) and the Network Services Entity (NSE) cross the Mcn reference point. These flows enable a CSE to use the supported services (other than transport and connectivity services) provided by the NSE.
- Mcc' Reference Point: Communication flows between two Common Services Entities (CSEs) in Infrastructure Nodes (IN) that are oneM2M compliant and that resides in different M2M SP domains cross the Mcc' reference point. These flows enable a CSE of an IN residing in the Infrastructure Domain of an M2M Service Provider to communicate with a CSE of another IN residing in the Infrastructure Domain of another M2M Service Provider to use its supported services and vice versa.

2.2.3 OGC Sensor Web Enablement (SWE)

Sensor Web Enablement (SWE) is a standard providing high-energy consuming services such as being alerted when a specific event occurs or asking for more detailed measurements [Jirka et al. 2009]. The goal of SWE is to seamlessly integrate heterogeneous hardware devices, and define structures for describing measured data from devices to a suite of web services available on the Web.
The main drawback of SWE is the learning curve to set up, configure and deploy this technology.

The information layer defined by the SWE comprises:

- SensorML for describing sensors (e.g., a sensor on a bottle of milk)
- O&M is an encoding for data observed or measured by sensors (e.g., the bottle of milk has an expiration date).
- SOS (Sensor Observation Service) which provides access for discovering and retrieving sensor data (e.g., the bottle of milk).
- SAS (Sensor Alert Service) which is used for alerting clients once an event they are interested in occurs (e.g., the expiry date of the bottle of milk is tomorrow).
- SPS (Sensor Planning Service) is for being informed that an event occurred via SMS (Short Message Service), email or phone.
- WNS (Web Notification Service) is for managing sensors remotely (e.g., switch off the fridge).

2.2.4 CoAP

CoAP (Constrained Application Protocol) is a request/response communication model to retrieve data from constrained devices and send it to the web, thus enabling easy integration with the standard web services [16], [20]. CoAP provides standardized and easy-to-use methods to make sensor data available. CoAP belongs to the CORE group which aims to reuse the RESTful architecture to prove web services adapted to constrained environments. CoAP supports discovery of services or resources as well as multicast and asynchronous messaging.

2.3 Platforms

2.3.1 OpenIoT

Open IoT Architecture uses, as depicted in Figure 11, two main components to introduce data to the system [17]. The core one is the Sensor Middleware and the other one is the Pub-Sub server.

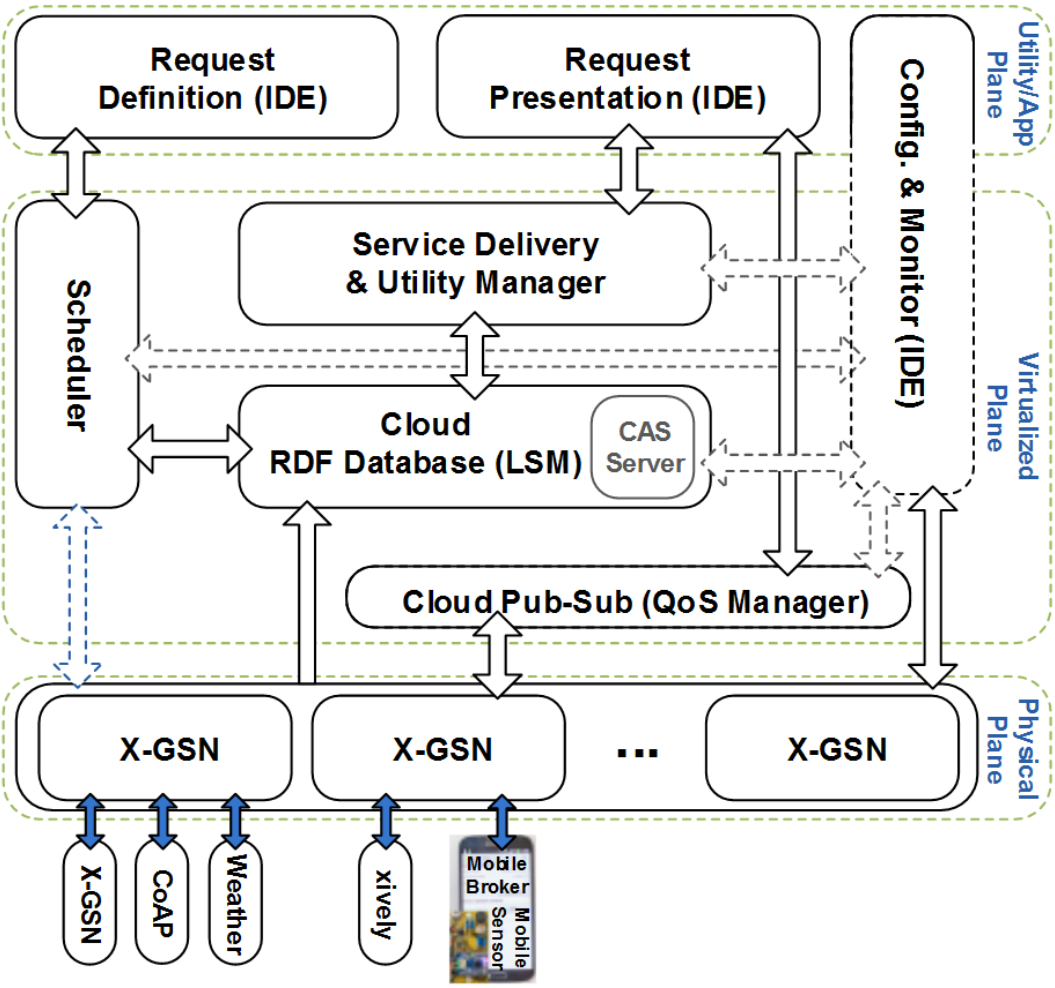

Figure 11 OpenIoT Main Core Components Functional Blocks

The Sensor Middleware (Extended Global Sensor Network, X-GSN) collects, filters and combines data streams from virtual sensors or physical devices. It acts as a hub between the OpenIoT platform and the physical world. The Sensor Middleware is deployed on the basis of one or more distributed instances (nodes), which may belong to different administrative entities. The implementation of the OpenIoT platform uses the GSN sensor middleware that has been extended and called X-GSN (Extended GSN) to which it adds semantic capabilities. X-GSN is designed to facilitate the deployment and programming of sensor networks. It runs on one or more computers composing the backbone of the acquisition network.

The Pub-Sub server (QoS Manager) is the component which monitors over time the global demand for sensor data generated by MIOs and manages the data acquisition process from MIOs to achieve a desired sensing coverage while optimising parameters such as energy and bandwidth consumption, sensor trustworthiness and/or data propagation latency.

2.3.2 VITAL

VITAL project provides the concept of Platform Provider Interfaces (PPIs) [18] that provide a low-latency interface for accessing data streams and capabilities of the underlying IoT systems.

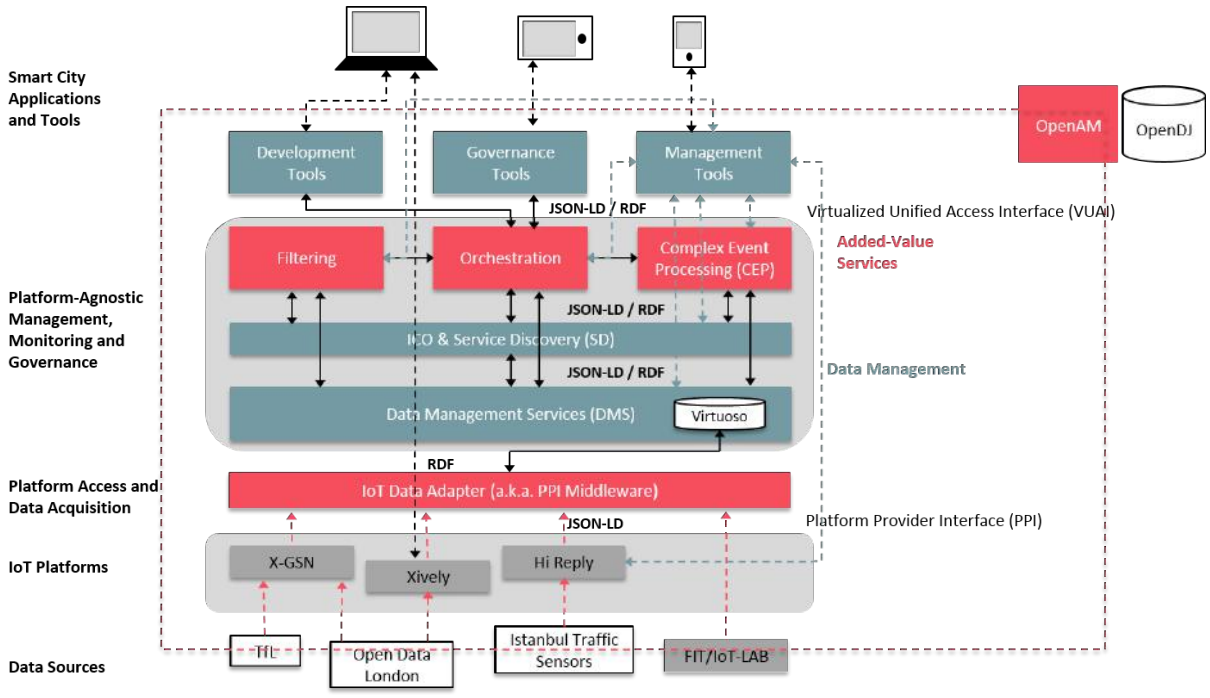

Figure 12 Overview of VITAL architecture

Figure 12 illustrates the (high-level) VITAL architecture. It illustrates concepts regarding high-performance access to platforms and data sources using PPIs. VITAL enables the federation of IoT platforms (such as FIT/IoT-LAB, X-GSN, Xively and Hi Reply). Access to individual data sources can be carried out through a PPI implementation, and more specifically based on a light implementation that does not implement optional functionalities. This concept is illustrated in Figure 13.

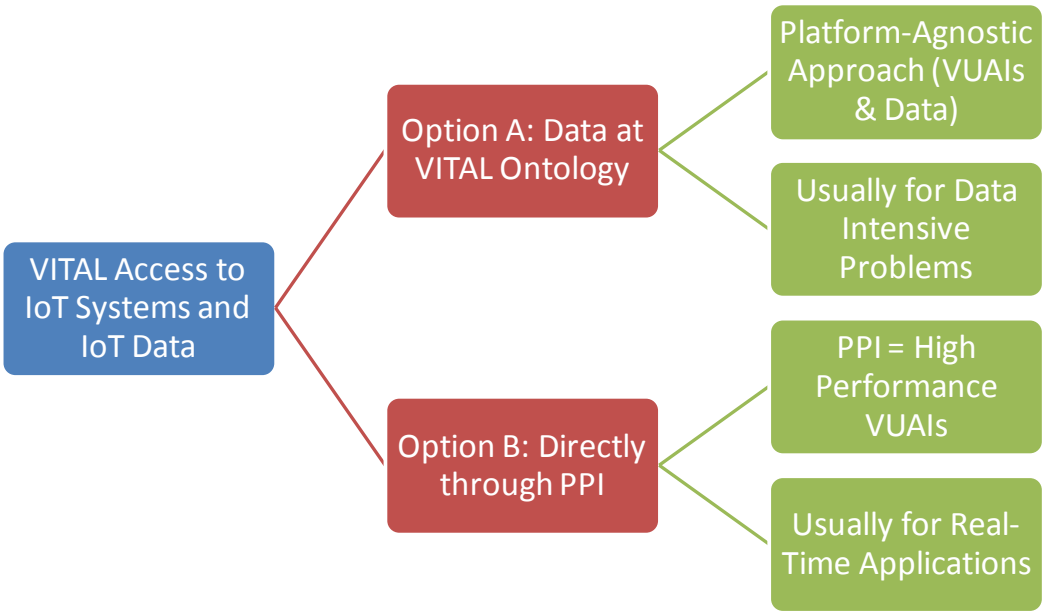

Figure 13 Two main options from VITAL Virtualization Layer (VUAIs) in IoT data access.

PPI is defined as a set of RESTful web services that IoT systems to be integrated into VITAL should expose, and that the VITAL platform 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 sensors that an IoT system manages (e.g. what they observe).
- Observations made by the sensors that an IoT system manages.

3 TESTBED PROVIDER INTERFACE SPECIFICATIONS.

3.1 Overview

The TPI (Testbed Provider Interface) is a set of RESTful web services that enables the integration of the testbeds to the FIESTA-IoT platform. The TPI is spanning across two different layers (Figure 14). The first is the TPI Configuration & Management layer that runs at the FIESTA-IoT platform side and controls the functionality of the TPI by utilizing the offered user interface for the User (Testbed provider). The second is the Testbed Provider Services (TPS) API where the Testbed provider has to implement a list of predefined services that enables the management and manipulation of the offered data.

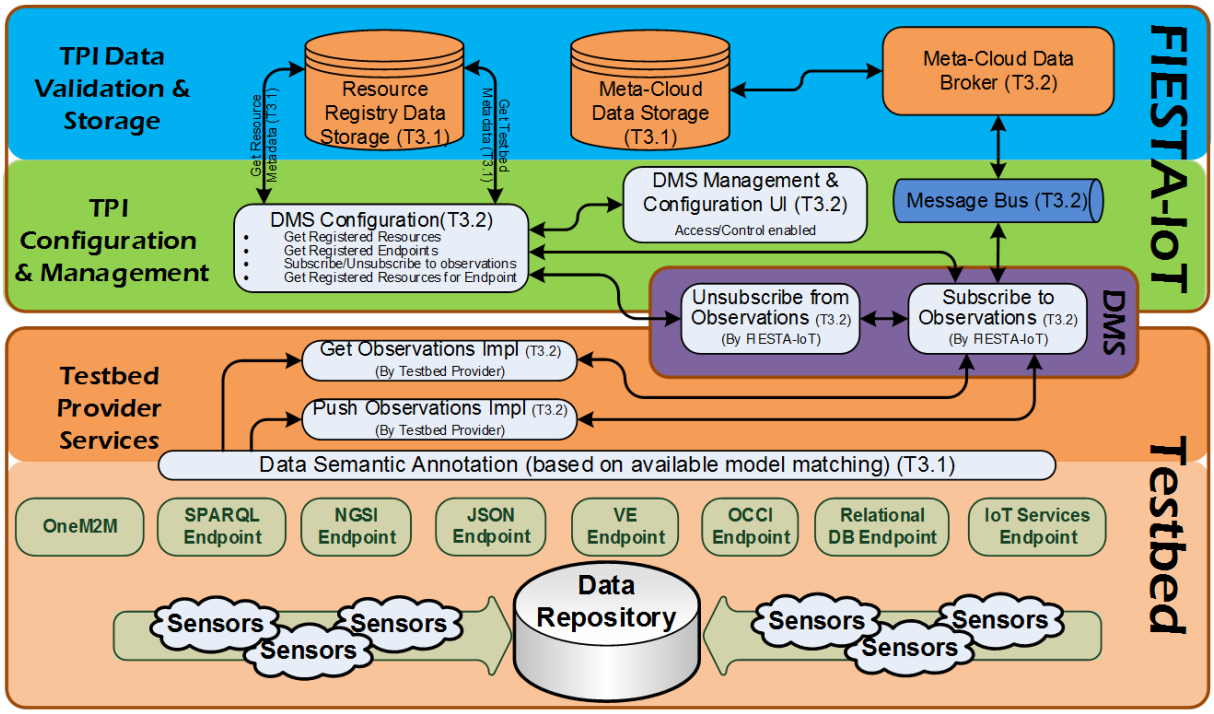

Figure 14 TPI Architecture

As we can see in Figure 14 above a testbed may expose internally various standard or proprietary interfaces in order to interact with the sensor data. FIESTA-IoT has specified a list of core services that should be exposed from a testbed in order to enable different connection methods to the platform. These services (i.e. getObservations) should be exposed from the testbed and will comprise the Testbed Provider Interface services. The behaviour of these methods will be controlled from a list of services that are provided from FIESTA-IoT and will enable the testbed provider to consume and control the TPS services that his/her Testbed exposes. The group of these services are called TPI Data Management Services (DMS) and will provide the ability to consume the services either by identifying a specific schedule or by enabling a data stream connection (i.e. Subscribe to Observations from Figure 14). In order to be able to initiate this configuration and set up process the Testbed provider needs first to register the metadata of his/her testbed and resources. This is done by utilizing the services that are exposed from the Semantic Registry Data management services (SRD) which stores the information in the Resource Registry

Data storage (see Figure 14 above). To retrieve the information stored in this data storage we are using the services that are identified by the Semantic Registry data Retrieval services (SRR). The SRR services along with the TPI DMS are used by the TPI configurator (Figure 14 above) which is a User Interface (UI) component. This UI enables the testbed provider to discover the semantically registered resourced by utilizing the SRR, and manage the data retrieval process by utilizing the TPI DMS.

In the sections below we provide a list of the different services exposed by the components that where introduced here.

3.2 TPI Configuration and Function Sequence example

An end-to-end example for the sequence of the different interactions of a testbed with the FIESTA-IoT platform is depicted in Figure 15 below. In this example, we presume that the testbed has implemented and exposes the “getObservations” services, which provides the FIESTA-IoT annotated measurements of a given list of resources for a specific time period.

- The first step for the Testbed provider is to utilize the **SRD component** and store (register) the description of his/her testbed/resources, based on the FIESTA-IoT ontology, to the FIESTA-IoT semantic registry repository.
- After successfully registering all the Testbed resources the Testbed provider can use the **TPI configurator UI** component in order to define how his annotated data are going to be introduces to the FIESTA-IoT platform. The **TPI configurator UI** provides the graphical tools in order to discover the already registered resources of his/her testbed by utilizing the **SRR** discovery interfaces.
- The service utilised at the **SRR** component is to provide all the registered resources for a specific Testbed ID. The **SRR** builds a SPARQL query with this request and executes it against the **Resource Registry** SPARQL interface. The **SRR** receives the list of registered resources that forwards it to the **TPI Configurator UI**.
- The User is capable now to choose the list of resources he/she is interested to interact with and by identifying the periodicity of the execution schedule and the Subscription URI (which in our case is the URI of the Message Bus). With the help of **TPI Configurator UI** the User can instruct the **Data Management Services** component to initiate the process of retrieving data from the Testbed and send them to the FIESTA-IoT platform.
- By receiving the command from the **TPI Configurator UI** with the list of involved resources along with the periodicity and the subscription URI thee **Data Management Services** component enters a continues loop, based on the execution schedule defined by the Testbed Provider. In every execution cycle it requests from the **Testbed Provider Services** (getObservations) all the measurements for the resource list for the duration of the last execution cycle.
- The annotated results received are then pushed to the **TPI Message Bus**.
- At this point, it is worthy to mention that upon deployment the **TPI MCDB** (Meta Cloud Data Broker) subscribes to the message bus for the messages produced by the **DMS** component.
- Therefore, as soon as a message is pushed to the **Message Bus** the TPI **MCDB** receives it and stores it to the **MCDS** (Meta Cloud Data Storage).

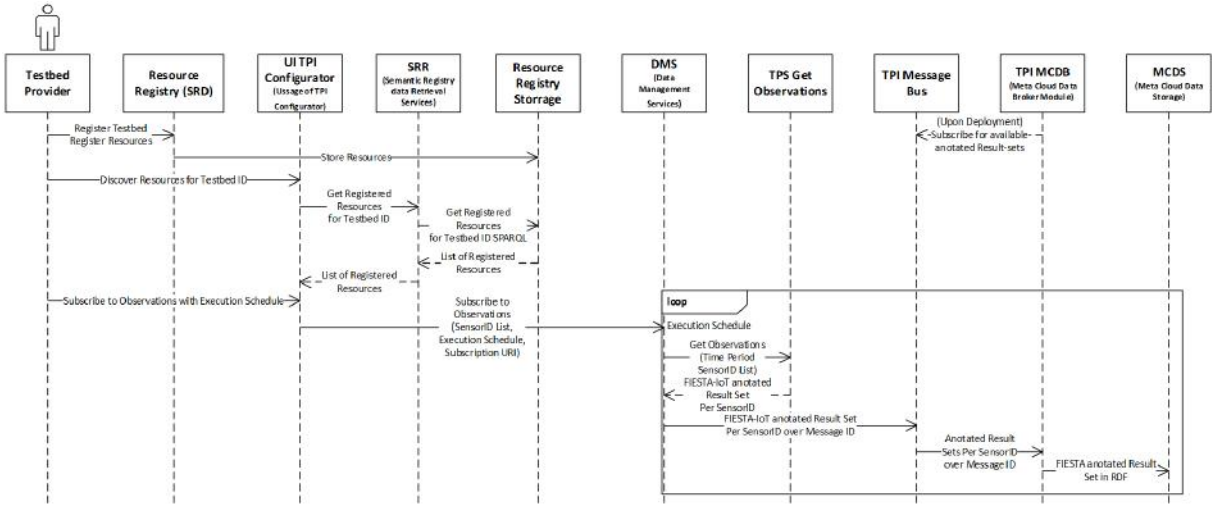

Figure 15 Configuration and Function Sequence example (getObservations)

3.3 TPI offered Services

As mentioned above, the TPI is a set of RESTful web services that enables the integration of the testbeds to the FIESTA-IoT platform. These services are going to be offered to the Testbed Provider in order to enable the "plugability" and management of their testbeds in relation with the FIESTA-IoT platform. These services have been categorized in three different groups depending on their functionality. These groups are:

- Semantic Registry Data services (SRD)
- Semantic Registry data Retrieval services (SRR)
- Testbed Provider services (TPS)
- TPI Data Management services (TPI DMS)
- Class I query services

3.3.1 Semantic Registry (SRD)

FIESTA-IoT has specified a list of services, which allows owners of IoT data sources to register semantic descriptions about their resources/devices, entities/objects on a linked-data platform. It also supports other CRUD methods for the management of the descriptions such as lookup, update and removal. In turn, the API allows consumers of IoT data (applications, experimenters, etc.) to discover their sources through semantic querying using a SPARQL endpoint.

This API offers two main services:

- The Registry which supports the management of Resource descriptions using the following operations for: Registration, Retrieval, Update and Removal of descriptions, and also supports multiple RDF serialization formats.
- The Query Service which supports the querying of Resource descriptions using SPARQL, embedded in the URL or in the body of the request.

List of Services:

- **Register Testbed**
  - When any testbed wants to become part of the FIESTA-IoT federation, it *MUST* be first registered into the platform. This latter point leads us to bring up the necessity of defining new elements in the FIESTA-IoT ontology in order to cover all the testbeds' key elements (e.g. ID, description, class, etc.).
- **Update Testbed Description**
  - A testbed may change during its lifetime (for instance, evolve from Class II to Class I). For that reason, a method that allows a dynamic modification of the underlying testbed's properties is deemed necessary.
- **Delete Testbed**
  - Testbed providers might decide that the testbed should not be part of the FIESTA-IoT federation any more. For those potential situations, we implement a service that allows them (i.e. testbed providers) to remove their platform from the system.
- **Register Resource**
  - As long as a testbed wants to inject information (i.e. observations) coming from its different underlying resources, it must register them beforehand. To do so, an annotated description is being sent to the FIESTA-IoT platform, compliant with the FIESTA-IoT ontology [4].
- **Update Resource**
  - The initial description of a resource is prone to vary throughout its lifetime. For example, its legacy testbed provider may add/remove sensors at any time. Thus, we provide this web service so that testbed owners can warn the FIESTA-IoT platform about these additions/deletions and keep the platform up-to-date.
- **Lookup Resource**
  - This web service will return a resource's description from its ID. This annotated description can be serialized to a number of compatible RDF formats.
- **Remove Resource**
  - As can be easily inferred, this service removes a resource from the Resource Registry Triple Store.
- **Resource Discovery**
  - This service allows a testbed provider to create a look-up to discover his/her resources, based on e.g. the location or phenomena.

3.3.2 Semantic Registry data Retrieval services (SRR)

FIESTA-IoT has specified a set of registry retrieval services, which will enable the testbed provider to review and manage the registered testbeds. These services are:

- **Get Registered Testbeds**
  - This service will provide a list of registered testbeds within a specific location so as to be used in the next step which is the resource discovery.

- **Get Registered Resources**
  - This service will provide a list of registered resources of a specific testbed so as to be used from the data acquisition services.
- **Get Registered Endpoints**
  - This service provides a list of Endpoints where a Testbed is sending data to.
- **Get resources registered to an Endpoint for a specific testbed**
  - This service provides a list of resources that have been registered to a specific URI from a specific Testbed to produce data.
- **Get Observation Streams**
  - This service provides a list of sensor streams of a testbed.
- **Get subscribers to Observation Streams**
  - This service provides a list of subscribers to Sensor streams.

3.3.3 Testbed Provider Services (TPS)

FIESTA-IoT has specified a list of services of which at least one should be implemented and exposed from the Testbed in order to enable the "plugability" of it to the FIESTA-IoT platform. These services are:

- **getLastObservations**
  - This service provides the latest values of a specific Sensor list.
- **getObservations**
  - This service provides the values of a specific Sensor list for a specific time-period once.
- **pushLastObservations**
  - This service pushes continuously the latest values of a specific Sensor list to a specific endpoint.
- **pushSingleObservation**
  - This service pushes continuously the latest value of a specific Sensor to a message bus with the Sensor ID as queue topic.

3.3.4 TPI Data Management Services (TPI DMS)

FIESTA-IoT has specified a list of services that will enable the testbed provider to manage the services exposed by the testbed in order to retrieve the data of the registered resources.

These services are:

- **pushObservations**
  - This service pushes all the values of a specific Sensor list with a specific time interval to an endpoint.
- **stopPushOfObservations**
  - This service stops the Observations pushing.
- **subscribeToObservations**
  - This service pushes the values of a specific Sensor list to a specific endpoint based on a specific execution schedule. In every execution

when the Sensors have to be reported the time period starts from the last execution.

- **subscribeToObservationStream**
  - This service pushes the values of a specific Sensor list to a specific endpoint as soon as they have new values.
- **subscribeToObservationStreamWithTopic**
  - This service pushes the values of a specific Sensor list as soon as they have new values to a message bus individually by using the Sensor ID as the queue topic.
- **unsubscribeFromObservation**
  - This service unregisters a list of Sensors from an endpoint.

3.3.5 Class I Query Services

According to the taxonomy defined in the Deliverable 2.4 [1], the FIESTA-IoT platform will have to deal with up to three different types (or classes) of testbeds. For a more thorough explanation of each of them, the reader should refer to the above deliverable. Whilst the most common case will be Class II-III testbeds, where they maintain their legacy formats to describe their resources, measurements and even Virtual Entities[7]. To become part of the FIESTA-IoT federation, they must adapt their likely proprietary formats to those proposed in [4]. In order to achieve that, these testbeds have to implement and execute, at testbed side, an annotator whose inputs are the different non-FIESTA-IoT datasets, producing as output information compliant to the one specified under the umbrella of the FIESTA-IoT ontology. Anyway, this part is clearly out of the scope of this deliverable and, as mentioned above, it is handled in [4].

On the other hand, Class I testbeds (regardless of their underlying role, either as Raw Data Producer or Virtualizer) follow a completely different approach. They can be seen as a natural extension of the FIESTA-IoT platform, since they internally "speak" the FIESTA-IoT ontology "language" to describe the different datasets (i.e. IoT services/resources, observations, VEs, etc.), hence they do not have to "translate" from their legacy formats to FIESTA-IoT's. In addition, they also store all this already annotated information into their own local repositories (or triple stores in case of semantic data), so neither the Meta-Cloud Data Endpoint, the VE registry nor the IoT Service/Resource Registry within the FIESTA-IoT core do need to replicate anything into the FIESTA-IoT core. Technically speaking, Class I testbeds are a distributed version (or, said in other words, a logical extension) of the FIESTA-IoT platform. Therefore, in what respects to the TPI specification, this type of testbed will belong to an exceptional situation, in which, from a physical point of view, the connection between FIESTA-IoT and these platforms would go across another TPI instance. Nonetheless, reality tells us that everything is inside the very same meta-cloud, which is embraced by the FIESTA-IoT federation, so every operation/method will be executed within the own platform (internally).

[7] Class II testbed are foreseen to implement a Virtual Entity (VE) Wrapper module through which Virtualizer might create new VEs and store their associations into the FIESTA-IoT VE registry.

Once we have seen the rationale for having a different API definition for this category of testbeds, below we describe the specific services that provide an interaction between Class I testbeds and the FIESTA-IoT platform.

- **registerTestbed**
  - As was commented above, the first step a testbed must carry out in order to become a part of the FIESTA-IoT federation will be its very own registration. Of course, Class I testbeds have to follow the same approach.

- **resourceDiscovery**
  - Before delving into the details of this concrete method, it is worth highlighting that the equivalent functionality will not be used for Class II-III testbeds, since it will be solved at FIESTA-IoT level and will not need an explicit access to the underlying testbeds. Then, focusing on Class I, an experimenter generates a SPARQL query and delivers it to the FIESTA-IoT Service & Resource Registry (i.e. Resource Broker), who is in charge of 1- parsing and 2- forwarding the query to the underlying testbeds.

- **getObservations**
  - Experimenters can directly access the data repositories – triple stores - through generating their own SPARQL requests (e.g. for retrieving historical values). In the same way as the “resourceDiscovery” method, these queries only reach Class I testbeds (in the other cases, they are solved at FIESTA-IoT level), forwarded by the FIESTA-IoT platform across the Meta-Cloud Data Endpoint’s Data Broker.

- **subscribeToObservationStream**
  - With this method, we introduce the possibility of using an asynchronous service (publish/subscription) to get access to data instead of manual polls. Experimenters have to define the sources of the data that they want to receive data from, together with some subscription-based info (e.g. period of subscription, etc.).

- **notifyObservation**
  - Class I testbeds must be aware of the experimenters/platforms that are subscribed to their resources/VEs. Otherwise, they would never send data (observations) to the FIESTA-IoT platform. Therefore, whenever they generate a new observation, they have to check (e.g. through their own Subscription Manager) whether the data must be sent or not. In case they must, they will deliver the message to wherever corresponds.

- **unsubscribeToObservationStream**
  - Brief description: experimenters might need to manually unsubscribe from a stream, apart from the expiration of the subscription time. Thanks to this operation, FIESTA-IoT offers the possibility of breaking down a stream.

4 CONFIGURATION AND MANAGEMENT USER INTERFACES

In this section, we are going to introduce the User Interfaces that are provided by FIESTA-IoT in order to facilitate the testbed provider to register and manage the testbed behaviour with the FIESTA-IoT platform.

4.1 TPI Configuration and management

As mentioned in section 3, FIESTA-IoT will offer a TPI configuration UI that will enable the testbed provider to discover the semantically registered resources, by utilizing the SRR services and manage the data retrieval process, by utilizing the TPI DMS services.

For the implementation of such user interface, we have decided to use a popular open source “flow wiring” environment called Node-Red[8]. In Figure 16 we can see the main layout of the user interface along with a simple service set up which is consuming the TPI DMS and SRR services.

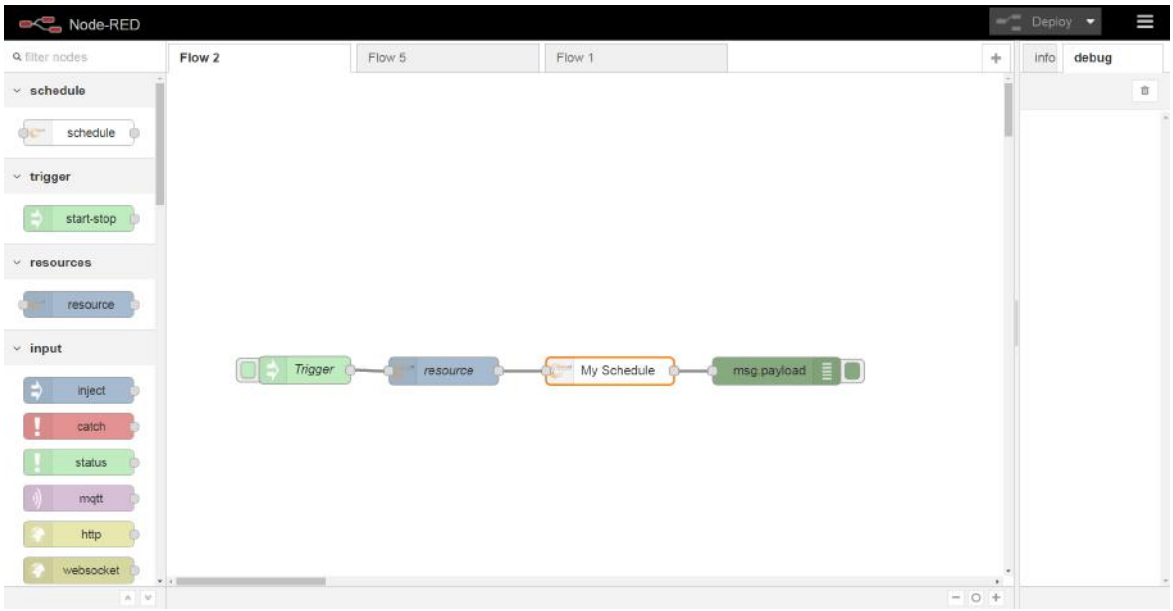

Figure 16 TPI configurator layout

As already mentioned the TPI configurator is utilizing the SRR services in order to discover the semantically registered resources of a Testbed. The discovery is achieved by using the properties of the resource node, which is offered from FIESTA-IoT at the Node-Red environment toolbox. The resource properties node, as shown in Figure 17, lists the registered resources and enables the testbed provider to choose which resources he/she would like to configure.

[8] http://nodered.org/

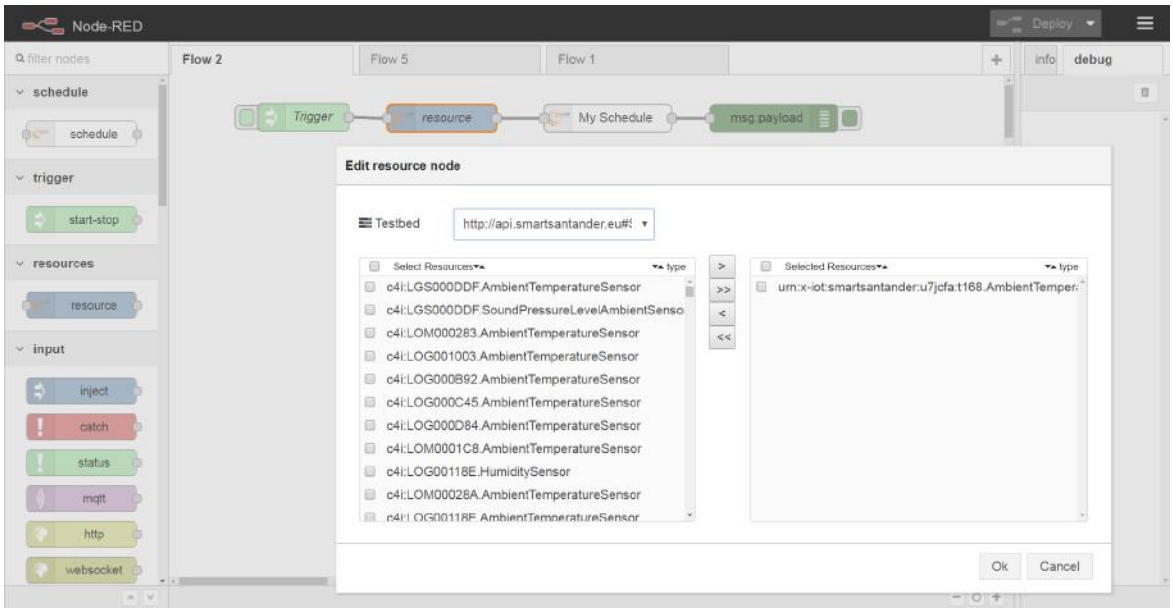

Figure 17 TPI configurator resource discovery

After identifying the resources, the Testbed provider could define a specific schedule to retrieve the measurements of the chosen resources. This is achieved by defining it in the “My Schedule” node properties window as shown in Figure 18. This node allows the user to define an endpoint where the actual measurements should be forwarded and stored. Moreover, the “schedule” node stores information about the start time of the schedule and the time intervals between each repeat (i.e. the frequency and time unit field shown in Figure 18). Note that in the time unit selection the available fields are seconds, minutes, hours, day and month). Finally, the “Default Endpoint” is the message bus we use for the measurement gathering.

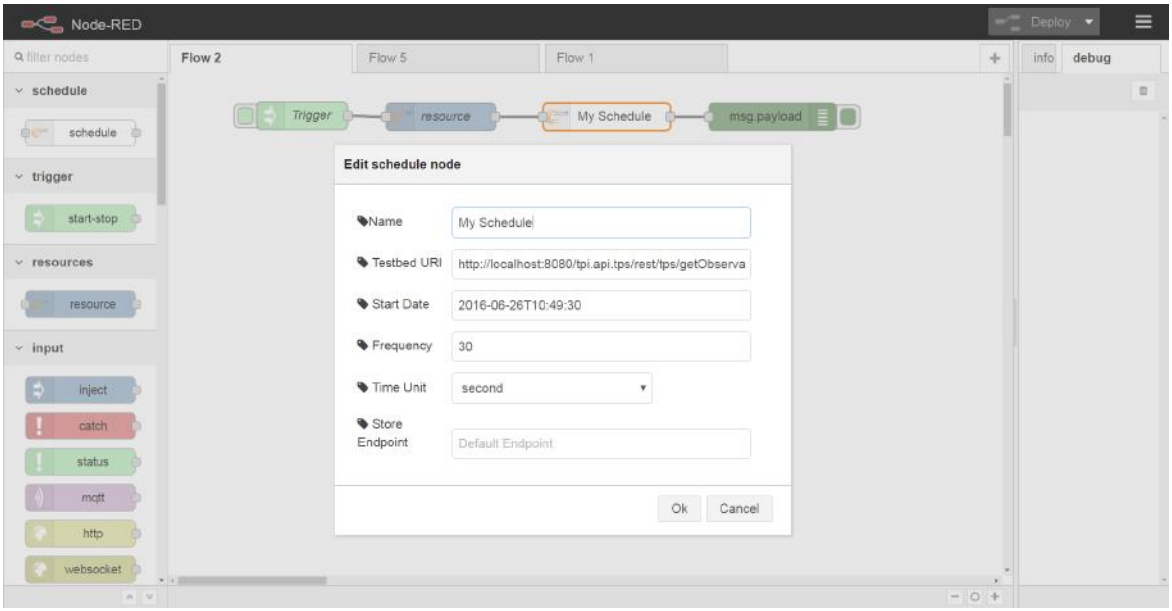

Figure 18 TPI configurator execution schedule definition

After completing the required flow and the definition of the requested properties, the testbed provider needs to hit the start node to initiate both the equivalent service of the TPI DMS and the process of pushing data to FIESTA-IoT platform or retrieving data from the Testbed.

5 TPI API SPECIFICATION

5.1 Semantic Registry (SRD)

5.1.1 API Definition

Table 3 below illustrates the main API primitives that supports the TPI Management functionalities, while Table 4 provides more details about each one of the functions that comprise the API.

Table 3 List of primitives comprising the implemented Semantic Registry API

<<interface>> SemanticRegistryInterface --- POST: registerTestbed (testbedDescription: JsonObject): String POST: updateTestbed (testbedDescription: JsonObject): String DELETE: removeTestbed (testbedDescription: JsonObject): String POST: registerResource (IoT_Resource_Description:String): IoT_Registration_Ack_Status:String GET/POST: lookupResource (IoT_Resource_ID:String): IoT_Resource_Description:String PUT: updateResource (IoT_Resource_ID:String ,Description:String): IoT_Update_Ack_Status:String DELETE: removeResource (IoT_Resource_ID:String): IoT_Remove_Ack_Status:String POST: resourceDiscovery (SparqlQuery:String): SPARQLQueryResult:String

A FIESTA-IoT implementation SHALL implement methods of the TPI Semantic Registry data Retrieval API as specified in Table 4 below:

Table 4 Implemented Semantic Registry API definition

Service Name	Input	Output	Info
registerTestbed	Testbed description	String successMessage	Register a Testbed
updateTestbed	Testbed description	String successMessage	Update a testbed description
removeTestbed	Testbed ID	String successMessage	Remove a testbed from the FIESTA-IoT federation
registerResource	Semantic Description for an IoT Resource	String successMessage	Register IoT Resource Description(s)
lookupResource	ID for IoT resource	Semantic Description for an IoT Resource	Lookup IoT Resource Description(s)
updateResource	ID for IoT Resource, Semantic Description	Acknowledgment of receipt and update status	Update IoT Resource Description(s)
removeResource	ID for IoT Resource	Acknowledgment of receipt and remove status	Remove IoT Resource Description(s)
resourceDiscovery	SPARQL query	SPARQL query result	Query for IoT Resources

5.1.2 Object Definition

For the registration and the updating part, the message sent to the SRD can be a description of a Resource (in a JSON-LD serialization) that is compliant with the FIESTA-IoT ontology:

```
{
  "@context": {
    "geo": "http://www.w3.org/2003/01/geo/wgs84_pos#",
    "iot-lite": "http://purl.oclc.org/NET/UNIS/fiware/iot-lite#",
    "j.0": "http://purl.oclc.org/NET/UNIS/iot-lite#",
    "j.1": "http://www.w3.org/2001/",
    "owl": "http://www.w3.org/2002/07/owl#",
    "qu": "http://purl.org/NET/ssnx/qu/qu#",
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
    "ssn": "http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#",
    "time": "http://www.w3.org/2006/time#",
    "xsd": "http://www.w3.org/2001/XMLSchema#"
  },
  "@graph": [
    {
      "@id": "j.0:LocationDesk1",
      "@type": "geo:Point",
      "j.0:RelativeLocation": "Desk1"
    },
    {
      "@id": "j.0:SensingDevice1Service",
      "j.0:endpoint": "http://iot.ee.surrey.ac.uk/testbed/SensingDevice1Service"
    },
    {
      "@id": "j.0:CoverageDesk1",
      "j.0:hasPoint": {
        "@id": "j.0:LocationDesk1"
      }
    },
    {
      "@id": "j.0:Desk1SensingDevice1",
      "@type": "ssn:SensingDevice",
      "j.0:hasLocation": {
        "@id": "j.0:LocationDesk1"
      },
      "j.0:hasQuantityKind": {
        "@id": "qu:temperature"
      },
      "j.0:isExposedBy": {
        "@id": "j.0:SensingDevice1Service"
      }
    }
  ]
}
```


For querying, the message body sent to the SRD must be like the following:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    PREFIX owl: <http://www.w3.org/2002/07/owl#>
    PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    PREFIX ssn: <http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#>
    prefix qu-rec20: <http://purl.org/NET/ssnx/qu/qu-rec20#>
    prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
    prefix iot-lite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#>
    prefix iot-l-Ins: <http://purl.oclc.org/NET/UNIS/fiware/iot-liteInstance#>

    SELECT ?sensDev ?endp
    WHERE {
        ?sensDev iot-lite:hasQuantityKind qu-rec20:temperature;
        iot-lite:isExposedBy ?serv;
        iot-lite:hasCoverage ?cover.
        ?cover iot-lite:hasPoint ?point.
        ?point iot-lite:RelativeLocation ""Desk2"".
        ?serv iot-lite:endpoint ?endp.
    }
```

Information to be exchanged between a testbed and FIESTA during the testbed registration phase:

Table 5 Testbed description data format

```
{
      "id": string (e.g. SmartSantander, SmartICS),
      "position": geojson (e.g. point, polygon/multipolygon, etc.),
      "endpoints":  [
            {
                  "iot-service": url,
                  "etc": ...
            },
            {
                  "data": url,
                  "etc": ...
            },
            {
                  "ve-service": url,
                  "etc": ...
            },
            {
                  "ve-data": url,
                  "etc": ...
            }
      ],
      "credentials": e.g. API key, X.509 certificate, Open AM-based solution,
etc. ,
      "capabilities": [
            {
                  "push-observations": boolean,
                  "etc": ...
            },
```



```
            {
                  "get-observations": boolean,
                  "etc": ...
            },
            {
                  "virtualizer": boolean,
                  "etc": ...
            },
            {
                  “resource storage”: {“proprietary”, “fiesta”, “none”},
                  “data storage”: : {“proprietary”, “fiesta”, “none”},
                  “etc”:
            }
      ]
      “metadata”: [
            “website”: url,
            “etc”: ...
      ]
}
```

5.2 Semantic Registry Data Retrieval Services (SRR)

5.2.1 API Definition

Table 7 below illustrates the main API primitives that support the TPI Management functionalities, while Table 7 provides more details about each one of the functions that comprise the API.

Table 6 List of primitives comprising the TPI Semantic Registry data Retrieval API

<<interface>>
SemanticRegistryDataRetrievalServicesInterface

GET:**getRegisteredTestbeds** (longitude:double, latitude:double, radius:float):List <String>
GET:**getRegisteredResources** (testbedID:String):List <String>
GET:**getRegisteredEndpoints** (testbedID:String,):List <String>
GET:**getRegResourcesToEndpointForTestbed** (testbedID:String, endpointURL String):List <String>
GET:**getObservationStreams** (testbedID:String):List <String>
GET:**getSubscribersToObservationStreams** (observationStreamURI:String):List <String>

A FIESTA-IoT implementation SHALL implement methods of the TPI Semantic Registry data Retrieval API as specified in Table 7 below:

Table 7 TPI Semantic Registry data Retrieval API definition

Service Name	Input	Output	Info
getRegisteredTestbeds	double longitude, double latitude, float radius	List <String>	This service provides a list of registered testbeds within a specific location.
getRegisteredResources	String testbedID	List <String>	This service provides a list of registered resources of a

			specific testbed so as to be used from the data acquisition services.
getRegisteredEndpoints	String testbedID	List <String>	This service provides a list of Endpoints that a Testbed is sending data to.
getRegResourcesToEndpoint ForTestbed	String testbedID , String endpointURL	List <String>	This service provides a list of resources that have registered to a specific URI from a specific Testbed for producing data.
getObservationStreams	String testbedID	List <String>	This service provides a list of sensor streams of a testbed.
getSubscribersToObservation Streams	String observationStreamU RI	List <String>	This service provides a list of subscribers to Sensor streams.

5.2.2 Requesting Large Amounts of Data

The service report allows a client to make a request for a potentially unlimited amount of data. For example, the response to a query that asks for all available sensor information within a given interval of time could conceivably return one, a thousand, a million, or a billion records depending on the time interval and how many events had been captured. This may present performance problems for the service implementations.

To mitigate this problem, a Scheduler Register service MAY reject any request by raising a QueryResultTooLarge exception. This exception indicates that the amount of data being requested is larger than the service is willing to provide to the client. The QueryResultTooLarge exception is a hint to the client that the client might succeed by narrowing the scope of the original query or by presenting the query at a different time (e.g., if the service accepts or rejects queries based on the current computational load on the service).

5.2.3 Exceptions

Methods of the TPI Management API signal error conditions to the client by means of exceptions. The following exceptions are defined. All the exception types in the following table are extensions of a common TpiManagementException base type, which contains one string element giving the reason for the exception.

Table 8 Exceptions associated with the TPI Management API

Exception Name	Meaning
SecurityException	The operation was not permitted due to an access control violation or other security concern. This includes the case where the service wishes to deny authorization to execute a particular operation based on the authenticated client identity. The

	specific circumstances that may cause this exception are implementation-specific, and outside the scope of this specification.
NoSuchTestbedIdException	The TPI Management failed to identify the Testbed ID i.e. the Testbed ID is not available within the resources repository.
ImplementationException	A generic exception raised by the implementation for reasons that are implementation-specific. This exception contains one additional element: a severity member whose values are either ERROR or SEVERE. ERROR indicates that the TPI Management implementation is left in the same state it had before the operation was attempted. SEVERE indicates that the TPI Management implementation is left in an indeterminate state.
QueryResultTooLargeException	An attempt to execute a query resulted in more data than the service was willing to provide.
QueryParameterException	One or more query parameters are invalid, resulting from any of the following situations: • the parameter name is not a recognized parameter for the specified query. • the value of a parameter is of the wrong type or out of range. • two or more query parameters have the same parameter name.

The exceptions that may be raised by each TPI Management method are indicated in the table below. A TPI Management implementation SHALL raise the appropriate exception listed below when the corresponding condition described above occurs. If more than one exception condition applies to a given method call, the TPI Management implementation may raise any of the exceptions that applies.

Table 9 Exceptions thrown by the different TPI Management services

Service Name	Throws
getRegisteredTestbeds	SecurityException NoSuchTestbedIdException ImplementationException QueryResultTooLargeException QueryParameterException
getRegisteredResources	SecurityException NoSuchTestbedIdException ImplementationException QueryResultTooLargeException QueryParameterException
getRegisteredEndpoints	SecurityException NoSuchTestbedIdException ImplementationException QueryResultTooLargeException

	QueryParameterException
getRegResourcesToEndpointForTestbed	SecurityException NoSuchTestbedIdException ImplementationException QueryResultTooLargeException QueryParameterException
getObservationStreams	SecurityException NoSuchTestbedIdException ImplementationException QueryResultTooLargeException QueryParameterException
getSubscribersToObservationStreams	SecurityException NoSuchTestbedIdException ImplementationException QueryResultTooLargeException QueryParameterException

5.3 TPI Services (TPS)

5.3.1 API Definition

Table 10 below illustrates the main API primitives that support the Testbed Provider Services functionalities, while Table 11 provides more details about each one of the functions that comprise the API.

Table 10 List of primitives comprising the Testbed Provider Services API

<<interface>> TpiServicesInterface --- POST:**getLastObservations** (getLastObservationsPayload:JsonObject):List <SensorValue> POST:**getObservations** (getObservationsPayload:JsonObject):List <SensorValue> POST:**pushLastObservations** (pushLastObservationsPayload:JsonObject):String GET:**pushSingleObservation** (sensorID:String, endPointURI:String):String

A FIESTA-IoT implementation SHALL implement methods of the Testbed Provider Services API as specified in Table 11 below:

Table 11 Testbed Provider Services API definition

Service Name	Input	Output	Info
getLastObservations	JsonObject getLastObservationsPayload	List <SensorValue>	This service provides the latest values of a specific Sensor list.
getObservations	JsonObject getObservationsPayload	List <SensorValue>	This service provides the values of a specific Sensor list for a specific time period once.

pushLastObservations	JsonObject pushLastObservations Payload	String successMessage	This service pushes continuously the last values of a specific Sensor list to a specific endpoint.
pushSingleObservation	String endPointURI, String sensorID	String successMessage	This service pushes continuously the last value of a specific Sensor to a message bus with the Sensor ID as queue topic.

5.3.2 Object Definition

The analysis of the JSON objects that are introduced in the previous section is provided below:

Definition of getLastObservationsPayload object:

```
{
    "sensorIDs": [<String> sensorIDs]
}
```

Definition of getObservationsPayload object:

```
{
     "sensorIDs": [<String> sensorIDs],
     "timePeriod": [<Date> startDate, <Date> stopDate]
}
```

Note: the Date has the following format "yyyy-MM-ddThh:mm:ss" and the start date precedes the stop date in time.

Definition of pushLastObservationsPayload object:

```
{
    "sensorIDs": [<String> sensorIDs],
    "endpointURI": <String> theEndpointURI,
}
```

5.4 TPI Data Management Services (TPI DMS)

5.4.1 API Definition

Table 12 below illustrates the main API primitives that support the TPI Management functionalities, while Table 13 provides more details about each one of the functions that comprise the API.

Table 12 List of primitives comprising the TPI Data Management Services API

<<interface>>
TpiDataManagementServicesInterface

POST:**pushObservations** (pushObservationsPayload:JsonObject):String
POST:**stopPushOfObservations** (stopPushOfObservationsPayload:JsonObject):String
POST:**subscribeToObservations** (stopPushOfObservationsPayload:JsonObject):String
POST:**subscribeToObservationStream** (subscribeToObservationsPayload:JsonObject):String
POST:**subscribeToObservationStreamWithTopic**
(subscribeToObservationStreamWithTopicPayload:JsonObject):String
POST:**unsubscribeFromObservation** (unsubscribeFromObservationPayload:JsonObject):String

A FIESTA-IoT implementation SHALL implement methods of the TPI Data Management Services API as specified in Table 13 below:

Table 13 TPI Data Management Services API definition

Service Name	Input	Output	Info
pushObservations	JsonObject pushObservationsPayload	String successMessage	This service pushes the values of a specific Sensor list with a specific time interval to an endpoint. (it utilizes either the getLastObservations or the getObservations from TPS)
stopPushOfObservations	JsonObject stopPushOfObservationsPayload	String successMessage	This service stops the Observations pushing.
subscribeToObservations	JsonObject subscribeToObservationsPayload	String successMessage	This service pushes the values of a specific Sensor list to a specific endpoint based on a specific execution schedule. The time period of the Sensors to be reported in every execution starts from the last execution.
subscribeToObservationStream	JsonObject subscribeToObservationStreamPayload	String successMessage	This service pushes the values of a specific Sensor list as soon as they have new values to a specific endpoint.
subscribeToObservationStreamWithTopic	JsonObject subscribeToObservationStreamWithTopicPayload	String successMessage	This service pushes the values of a specific Sensor list as soon as they have new values to a message bus individually by using the Sensor ID as the queue topic.
unsubscribeFromObservation	JsonObject unsubscribeFromObservationPayload	String successMessage	This service unregisters a list of Sensors from an endpoint.

5.4.2 Object Definition

The analysis of the JSON objects that are introduced in the previous section is provided below:

Definition of subscribeToObservationsPayload object:

```
{
    "sensorIDs": [<String> sensorIDs],
    "endpointURI": <String> theEndpointURI,
    "testbedURI": <String> theEndpointURI
    "timeSchedule": {
         "startTime" : <Date> startDate,
         "frequency" : <int> theFrequency,
         "timeUnit" : <String> thetimeUnit
     }
}
```

Note:

1. the Date has the following format " yyyy-MM-ddThh:mm:ss"
2. the startDate has to be a future Date
3. the frequency is an integer that defines the time interval, e.g. 5 seconds
4. the timeUnit is one of the following Strings: second, minute, hour, day month. It represents the time step, e.g. e.g. 5 seconds

Definition of pushLastObservationsPayload, stopPushOfObservationsPayload, subscribeToObservationsPayload, unsubscribeFromObservationPayload and subscribeToObservationStreamWithTopicPayload objects:

```
{
    "sensorIDs": [<String> sensorIDs],
    "testbedURI": <String> theEndpointURI
}
```

Definition of pushObservationsPayload object:

```
{
    "sensorIDs": [<String> sensorIDs],
    "endpointURI": <String> theEndpointURI,
    "testbedURI": <String> theEndpointURI,
    "periodicity": <long> seconds
}
```


5.5 Class I Query Services

5.5.1 API Definition

Table 14 below illustrates the main API primitives that support the TPI Management functionalities, while **Error! Reference source not found.** provides more details about each one of the functions that comprise the API.

Table 14 List of primitives comprising the implemented Class I Query Services API

<<interface>>
RegisterTestbed (Description: RDF/JSON, Endpoints: Array of URLs): String
---
POST: registerTestbed (testbedDescription: JsonObject)
POST: resourceDiscovery (SparqlQuery:String): SPARQLQueryResult:String
POST: getObservations (getObservationsPayload: JsonObject): List <SensorValue>
POST: subscribeToObservation (JsonObject subscribeToObservationsPayload): String
POST: notifyObservation (JsonObject pushObservationsPayload) String
POST: unsubscribeFromObservations (unsubscribeFromObservationPayload:JsonObject):String

A FIESTA-IoT implementation SHALL implement methods of the Class I Query Services API as specified in **Error! Reference source not found.** below:

Table 15 Implemented Class I Query Services API definition

Service Name	Input	Output	Info
registerTestbed	JsonObject Testbed description	Acknowledgment and completed testbed description	This service sends a request for registering a testbed into the FIESTA-IoT federation.
resourceDiscovery	SPARQL query (What/Where), Credentials (e.g. API key), serialization format (e.g. JSON-LD, RDF/XML, etc.)	List of resource descriptions (default: complete graph; otherwise, tailored response to the SPARQL)	This service provides a list of resource descriptions filtered from a (SPARQL) query part of the input parameters. NOTE: a testbed will only retrieve information of its own resources, not others'.
getObservations	getObservationsPayload	List <SensorValues>	This service provides the values of a specific Sensor list for a specific time period once. If the time period is missing the latest value of the resources will be provided.
subscribeToObservation	JsonObject subscribeToObservationsPayload	String successMessage	This service pushes the values of a specific Sensor list to a specific endpoint based on a specific execution schedule. The time period of the Sensors to be reported in every execution starts from the last execution. If the execution schedule is missing this service will push the values of

			the specific Sensor list as soon as they have new values to a specific endpoint.
notifyObservations	JsonObject pushObservationsPayload	String successMessage	This service pushes all the values of a specific Sensor list with a specific time interval to an endpoint. However, Class I testbeds have to process and deliver only those observations which somebody has already subscribed to.
unsubscribeFromObservationStream	JsonObject unsubscribeFromObservationPayload	String successMessage	This service unregisters a list of Sensors from an endpoint.

6 BACKGROUND IMPLEMENTATION TECHNOLOGIES

In this section we list various technologies that where considered for the different components implementation. We mainly list the candidates for processing engines and message bus usage. The list is not exhaustive but we choose to mention the most important of them.

6.1 Message Bus

The **Apache ActiveMQ**[9] is an open-source[10] message broker and integration patterns server written in Java. Instead of letting multiple applications communicate with each other in several diverse formats, ActiveMQ allows them to contact an ESB that takes over the manipulation and routing of messages. To act as a transit system, it supports a variety of cross language clients and protocols, and comes coupled with a Java Message Server (JMS) client to transform messages across a variety of systems for interoperable reliable messaging. Full support of JMS and J2EE assist to maintain the persistent, transactional and transient XA (eXtended Architecture) messaging of the ActiveMQ message broker. In order to exchange information in a language-neutral way RESTful services are also provided, while meeting fast persistence and high performance standards.

ActiveMQ is designed to be fast and adopts easy to use Enterprise Integration Patterns. One special Enterprise feature of ActiveMQ is that it can handle communication from more than one clients or servers. In IoT environments, lightweight messaging protocols such as the MQTT (Message Queuing Telemetry Transport) protocol are essential to foster communication from multiple sources. ActiveMQ is therefore ideal for high performance clustering, machine-to-machine (M2M) and peer-based communication. Moreover, it also supports other pluggable transport protocols to achieve the communication such as TCP, UDP, SSL, NIO and multicast.

[9] http://activemq.apache.org/

[10] Released under the Apache 2.0 license

RabbitMQ[11] is an open-source[12] message broker. It acts as an intermediate that provides a common platform for applications to accept, forward, store and trace messages. This allows the software applications to connect to each other as part of larger applications or to other devices and scale. Multiple RabbitMQ servers can also be clustered together to act as a single broker or can connect to form a federation of servers that support a more loosely connection. RabbitMQ supports asynchronous messaging, non-blocking operations push notifications and publish-subscribe mechanisms.

One of the principle features that RabbitMQ offers is reliability. Reliability is achieved in many ways such as delivery acknowledgements, high availability through mirrored queues, publisher confirmations and persistence. The messages are routed via several embedded exchange types and over a variety of messaging protocols.

WSO2 Message Broker[13] is an open source[10], easy-to-use distributed message brokering server. It is lightweight and it can scale up to several severs in a cluster, while offering high availability. WSO2 is designed to handle a large number of queues, subscribers and messages persistently. In case the number of concurrent events is high, MQTT protocol allows a machine-to-machine (M2M) communication to achieve enterprise messaging to IoT.

One important advantage of the WSO2 Message Broker is its interoperability with many languages and its in-order reliable messaging. The latter is achieved both via the JMS and the AMQP (Advanced Message Queuing Protocol) support and offers additionally distributed queuing and a publish-subscribe model. Moreover, the broker architecture is dynamically scalable at run-time and its activity and state can be monitored through a management console.

The **WSO2 Enterprise Service Bus**[14] is an open source[10], lightweight and versatile ESB. It supports integration standards and the entire set of enterprise integration patterns. Both the application and the system interoperability allow acting on messages across systems and on external services. To achieve the mediation flows, WSO2 comes coupled with a group of building blocks, called mediators that are responsible for any data manipulation and for operating with external applications and services.

In order to be highly agile and flexible, it offers a large set of connectors and a cloud-enabled version of the WSO2 ESB. It also supports SaaS applications as well as RESTful services and APIs. MQTT and Apache Kafka make it possible for WSO2 to be part of IoT scenarios.

WSO2's Message Broker guaranties the asynchronous delivery of messages and the WSO2 ESB is designed for high throughput managing thousands of transactions per second. It also ensures load balancing and high available deployment through failover techniques. (WSO2)

[11] https://www.rabbitmq.com/

[12] Licenced under the Mozilla Public Licence Version 1.1

[13] http://wso2.com/products/message-broker/

[14] http://wso2.com/products/enterprise-service-bus/

Oracle Service Bus[15] is an ESB acquired by Oracle. Its fundamental goals are mainly to transform architectures in order to connect both services and applications and also to manage their mediations and interactions. It is designed to deliver high performance for critical SOA environments by supporting integration standards. Moreover, the Oracle Service Bus offers scalability and reliability event for critical service implementations. (OSB)

The **Mule ESB**[16] is a service bus and integration platform that has been released[17] to connect applications together superficially and allow them to communicate with messages. Mule ESB is capable of acting efficiently as a mediator for interactions between components and applications whether they are placed on the same machine or are distributed over a cluster of machines, regardless of their actual communication protocols. JMS, Web Services, JDBC and HTTP are only some of many integration patterns that are supported. Generally, Mule promises to handle components of any type, as it transforms it to another desired category.

If content and rule-based interactions are required, Mule ESB exposes its capability to filter, route and aggregate the messages that are forwarded. To achieve the messaging with the Mule ESB, one can also include already existing components without needing to make any change to run it, since the business logic and the messaging logic are independently working.

OpenESB[18] is an open source[19] Java-based ESB tool for designing, building and testing SOA applications. Relying on JBI (Java Business Integration), its unique key capability is its development process that is suggested to be migrated to real service oriented development and organization.

Another important advantage of OpenESB is that it is simple and easy to use and at the same time also scalable. OpenESB has found the best balance between scalability and power. While promising high-availability for tens of ESB instances distributed over the same cluster, the optimized bus is able to support tens of millions of messages per day efficiently and safely.

Users of OpenESB also benefit from the standard integration patterns it adopts, e.g XML, WSDL, XSD and others. This makes any deployment of OpenESB projects compliant to those standards, compatible with any other SOA project and consistent. (OpenESB)

MQTT[20] is a publish/subscribe, simple and lightweight messaging protocol designed for constrained devices and low-bandwidth, high-latency and unreliable networks. The protocol minimises network bandwidth and device resource consumption while keeping the reliability and assurance of delivery.

[15] http://www.oracle.com/technetwork/middleware/service-bus/overview/index.html

[16] https://www.mulesoft.com/resources/esb/what-mule-esb

[17] Licensed under the Common Public Attribution License (CPAL)

[18] http://www.open-esb.net/

[19] Released under Common Development and Distribution License (CDDL)

[20] http://mqtt.org/

6.2 Process engines - Schedulers

Apache ServiceMix[21] is an open-source[10] integration container that combines powerful features and functionalities of Apache ActiveMQ, Apache CXF, Apache Camel and Apache Karaf, making it a lightweight and easily used tool to build tailored integration solutions. Remoting, distributed failover and clustering are only some of the main advantages ServiceMix promises. More specifically, ActiveMQ assures the reliable messaging and Apache Camel handles routing and Enterprise Integration Patterns. It is based on the SOA model and is built on the application interfaces of JBI (Java Business Integration). Its RESTful web services are supported by Apache CXF while the OSGi-based ESB server runtime is powered by Apache Karaf.

The Apache ServiceMix toolkit is easily embeddable and can run both inside a client and a server. It can be used as a standalone ESB provider or as a service within another ESB component. ServiceMix is available in Java's Enterprise and Standard Edition and includes a complete JBI container, that receives messages that are written in files or polling directories. The messaging is JMS supported, it is rule-based routed and the actual transactions are achieved by the XA standard. The toolkit uses the Quartz[22] library in order to integrate the timer and enables Java APIs for XML-based Web Services to make remote service calls or host Web services over a set of protocols. (ServiceMix Wiki) (SevriceMix Javaworld)

JBOSS FUSE[23] is an open-source[10], service oriented (SOA) integration platform including a modern ESB that supports methodologies, servers and tools for distributed application integration. Actually, it can be integrated both on premise and in the cloud efficiently and it can connect all enterprise components. Enterprise solutions rely on the JBI and OSGi components of the Fuse system. JBoss Fuse consists of a set of industry-standards that allow on-the-fly configurations and dynamic management of integration brokers. At the same time the controlled access to the services and the platform, eliminate possible security gaps.

JBoss Fuse would be a great solution for meeting service-oriented architecture requirements, as external applications are able to connect through a variety of connectors. JDBS, HTTP/HTTPS, FTP/SFTP and file are only some of the protocols used for binding the components. Moreover, JBoss Fuse offers a productive development environment that is ideally for creating seamlessly APIs and for visual data transformation. (JbossFuse RedHat) (JbossFuse Forrester)

The **Quartz Job Scheduler**[24] is an open source[10] library designed for job scheduling. It is a suitable solution for any Java application that is a standalone project or even commercial products. Schedules can include simple scenarios comprising of tens or hundreds of jobs that are waiting to be executed, or more complex schedules of thousands of jobs.

With Quartz, one can schedule jobs to be carried out in a specific timeslot in future, to log data into files from databases or fire alarms in particular scenarios. Jobs are

[21] https://servicemix.apache.org/

[22] https://www.quartz-scheduler.org/

[23] http://www.jboss.org/products/fuse/overview/

[24] https://www.quartz-scheduler.org/

considered in Quartz as standard Java classes that implement the Job interface and can execute virtually any task. In order to perform a task, the scheduler notifies a Listener each time a trigger occurs, a job has been completed or needs to be resubmitted; a listener is for example a JMD publisher. Various solutions to store the Jobs, in a database on in RAM, are also provided. The Job execution happens within a JTA transaction.

One of the main advantages of Quartz is that it can work equally well as a standalone instance within its own JVM and as part of another application, or even within an application server to participate in XA transactions. Finally, Quartz guaranties load balancing, failover and clustering. (QuartzScheduler)

As its name states, the **JobScheduler**[25] is an open source[26] scheduler for workload automation. It is designed to launch shell scripts and executable files. All database procedures run automatically and any information is stored in a database management system.

The JobScheduler can prepare jobs to be executed, independently, in chains or in parallel. One of its great features is that it allows jobs to run in particular time slots and can be triggered by multiple events. Such trigger events can be calendar schedules, API events or signals from incoming files from external applications. On top of that, JobScheduler supports batch scheduling and accepts priority settings for the jobs.

In order to provide high availability, there is always at least one backup JobScheduler that is running on another computer. This ensures that the system is automatically failover in case of any fail. However, high availability is also supported when a large volume of time consuming processes appears. In this case, multiple JobSchedulers are used to spread the workload over the hosts. Any file transfer of the jobs is based on FTP and SFTP protocols and only one of the jobs is allowed to access any shared resource at the time.

All the above, make the JobScheduler a very flexible solution even for enterprise level projects as it can be customized according to the project's demands.

6.3 The selected tools

Having in mind the goals of Fiesta-IoT and the advantages of the above tools we adopted the following technologies: We chose ActiveMQ as our service bus, since it is ideally for handling communication from multiple sources, in a high performance and lightweight way. On top of this, and maybe the key factor of choosing ActiveMQ over the other options, is its ability of interoperable reliable messaging. Finally, since Fiesta-IoT will offer scheduled tasks, we picked the Quartz scheduler for the job scheduling, as it is suitable for Java standalone projects that can handle tens or hundreds of simple or more complex jobs.

[25] http://www.sos-berlin.com/jobscheduler

[26] Licensed under GNU GPL (General Public License)

7 TPI PROTOTYPE IMPLEMENTATION

7.1 Source code Availability and Structure

The Testbed Provider Interface components are offered at FIESTA-IoT GitLab repository which is named "core" and is available at: https://gitlab.fiesta-iot.eu/platform/core/. The latest version of the components are under the "develop branch"[27] .

The repository is organized in 4 categories/folders and the TPI components are placed as follows:

- doc : provides all the related documents with the platform.
- module: provides the core modules of the platform
  - tpi
    - tpi.api.tps : Testbed provider services API
    - tpi.api.dms: Testbed provider interface data management services
  - srd: Semantic Registry Data management services
  - srr: Semantic Registry Data Retrieval Services
- ui: provides all the modules related to the User Interface
  - ui.tpi.configurator: Configuration UI for TPI by using the TPI Configurator, TPI Data Management, SRD and SRR
- utils: provides utilities related with the platform
  - utils.commons: include the common utils/objects used in more than 2 projects/components

7.2 Common Information

All of the described components are deployed within WildFly container and are using Maven as project management. Below you can find some common information that applies to all of the components.

7.2.1 System Requirements

The prototypes runs on Windows, Linux, Mac OS X, and Solaris. In order to run the prototypes, you need to ensure that Java 8 and WildFly are installed and/or available on your system. In order to build the prototypes you will also need Maven.

Before attempting to deploy and run the prototype applications, make sure that you have started WildFly.

More details about the specific versions of the tools and libraries that have been used for the development or that are required for the deployment and execution of the prototypes are given in the section below.

[27] https://gitlab.fiesta-iot.eu/platform/core/tree/develop

7.2.2 Install & Run

The prototypes has been implemented as a Maven-based web application. Below **WILDFLY_HOME** indicates the root directory of the WildFly distribution, and **PROJECT_HOME** indicates the root directory of the project.

In order to **configure** the prototype,

1. make sure that all properties listed in **$PROJECT_HOME/src/main/resources/fiesta-iot.properties** have the appropriate values,
2. copy that file into **$WILDFLY_HOME/standalone/configuration**, and
3. issue the following commands:

In order to **build** the prototype, run the following command in **PROJECT_HOME**:

mvn clean package

Finally, in order to **deploy** the prototype, run the following command in **PROJECT_HOME**:

mvn wildfly:deploy

The last step assumes that WildFly is already running on the machine where you run the command.

Alternatively copy the produced (from the build process above) **ProjectName.war** file from the **target** directory (**$PROJECT_HOME/target/**), into the **standalone/deployments** directory of the WildFly[28] distribution, in order to be automatically deployed.

If the deployment has been successfully completed, you will be able to access all web services described in the section above using the following URL:

http://[HOST]:[PORT]/ProjectName

where [HOST] is the host and [PORT] the port that WildFly uses.

7.3 Components

7.3.1 Semantic Registry (SRD)

The produced component (from the build process above) is the **srd-X.war** file.

If the deployment has been successfully completed, you will be able to access all web services described in the section above using the following URL:

http://[HOST]:[PORT]/srd

where [HOST] is the host and [PORT] the port that WildFly uses.

[28] http://wildfly.org/

7.3.1.1Containers and Libraries

The following table lists the containers and libraries that have been used for the implementation of the prototype. The versions specified in the table are the ones that have been used during the development.

Container / Library / Framework	**Version**
WildFly	9.0.1.Final
Java Platform, Standard Edition	1.8.0_25
Maven	3.1.1
Jena	3.0.1
restlet	2.3.2
californium-core	1.0.0-M3
commons-io	2.4

Table 16 Containers and libraries used for the prototype implementation

7.3.1.2Usage

The operations below will partly or wholly have the URL structure below:

```
http://{server_host}/srd/{endpoint_name}/{repository_id}/{resource_id}
```

Where:

- **server_host**: is the IP address or hostname of the server. This will also include the port number.
- **endpoint_name**: name of the endpoint that is done. This can be either registry or sparql.
- **repository_id**: is the ID of the target repository on the server. The server might have one or more repositories.
- **resource_id**: is the id of the IoT resource or entity.

7.3.2 SRR

7.3.2.1System Requirements

The produced (from the build process above) project name is the **srr-X.war** file.

If the deployment has been successfully completed, you will be able to access all web services described in the section above using the following URL:

http://[HOST]:[PORT]/srr

where [HOST] is the host and [PORT] the port that WildFly uses.

7.3.2.2Containers and Libraries

The following table lists the containers and libraries that have been used for the implementation of the prototype. The versions specified in the table are the ones that have been used during the development.

Container / Library / Framework	Version
WildFly	9.0.1.Final
Java Platform, Standard Edition	1.8.0_25
Maven	3.1.1

Table 17 Containers and libraries used for the prototype implementation

7.3.3 TPI DMS

7.3.3.1System Requirements

The produced (from the build process above) project name is the **tpi.api.dms-X.war** file.

If the deployment has been successfully completed, you will be able to access all web services described in the section above using the following URL:

http://[HOST]:[PORT]/ tpi.api.dms

where [HOST] is the host and [PORT] the port that WildFly uses.

7.3.3.2Containers and Libraries

The following table lists the containers and libraries that have been used for the implementation of the prototype. The versions specified in the table are the ones that have been used during the development.

Container / Library / Framework	Version
WildFly	9.0.1.Final
Java Platform, Standard Edition	1.8.0_25
Maven	3.1.1

Table 18 Containers and libraries used for the prototype implementation

7.3.4 TPI TPS

The produced (from the build process above) project is the **tpi.api.tps-X.war** file.

If the deployment has been successfully completed, you will be able to access all web services described in the section above using the following URL:

http://[HOST]:[PORT]/ tpi.api.tps

where [HOST] is the host and [PORT] the port that WildFly uses.

7.3.4.1Containers and Libraries

The following table lists the containers and libraries that have been used for the implementation of the prototype. The versions specified in the table are the ones that have been used during the development.

Container / Library / Framework	Version
WildFly	9.0.1.Final
Java Platform, Standard Edition	1.8.0_25
Maven	3.1.1

Table 19 Containers and libraries used for the prototype implementation

8 CONCLUSIONS

This deliverable has illustrated the specification and implementation of the FIESTA-IoT TPI, which is the interface enabling interaction and information exchange between the FIESTA-IoT experimental infrastructures and IoT testbeds. The TPI specification has been produced taking into account the envisaged functionalities of the interface, the nature and properties of the various testbeds, as well as a set of IoT standards that should be supported regarding the representation of data and services. To this end, the functionalities and properties of the FIESTA-IoT testbeds have been thoroughly reviewed and briefly presented in the deliverable. The consideration of these testbeds and their properties provided a sound basis for specifying a fairly general TPI, which can cover the needs of multiple IoT testbeds, but also the needs of the IoT experimenters.

A main conclusion from the TPI specification process is that such interfaces specify two sets of functionalities: Functionalities targeting the configuration and management of the testbeds and functionalities aimed at acquiring data and/or invoking services from the testbed. This dual nature of functionalities is also evident in the review of similar platform agnostic interfaces from other projects. A second important conclusion relates to the importance of supporting standards-based interfaces to testbeds. In addition to technological longevity, support for standards provides a sound basis for easy integration and wider use of the TPI by testbed owners. For example, NGSI support enables the integration of a large number of testbeds and IoT platforms that support this interface (including for example FIWARE based IoT platforms).

In terms of the TPI implementation, we can also conclude that an integrating middleware infrastructure (such as message oriented middleware or even a service bus) is essential, along with a job scheduling infrastructure. Nowadays there is a variety of options of such infrastructures to select from. In addition to criteria such as efficiency, performance and maturity, openness should be also added in the selection criteria, in order to enable future community extensions to the experimental infrastructure.

The TPI is an essential element of the FIESTA-IoT's experimental infrastructure. It will be well documented and offered to the participants of the open call processes, given that these participants will have to implement the TPI in order to integrate their testbed or IoT infrastructure with FIESTA-IoT. Open calls will enable a larger scale validation of the FIESTA-IoT TPI prototype implementation, while also providing feedback for fine tuning the specification and implementation of the interface.

9 REFERENCES

[1] FIESTA-IoT, “Deliverable 2.4: FIESTA-IoT Meta Cloud Architecture”, 2015.

[2] FIESTA-IoT, “Deliverable 2.1: Stakeholders Requirements”, 2015.

[3] FIESTA-IoT, “Deliverable 2.2: Analysis of IoT Platforms and Testbeds,” 2015.

[4] FIESTA-IoT, “Deliverable 3.1.1: Semantic models for testbeds, interoperability and mobility support and best practices,” 2016

[5] IoT-A, “Deliverable D1.5: Final Architectural Reference Model for the IoT,” 2013.

[6] IoT-A, “Deliverable D1.2: Initial Architectural Reference Model for IoT,” 2011.

[7] IEEE, “Guide for Monitoring, Information Exchange, and Control of Distributed Resources Interconnected with Electric Power Systems,” 2007. [Online]. Available: http://goo.gl/gEQlqd. [Accessed: 31-Mar-2016].

[8] V. Haren, “TOGAF Verson 9.0,” 2009.

[9] A. Gavras, “Experimentally Driven Research White Paper,” 2010.

[10] IEEE, “IEEE Standard Glossary of Software Engineering Terminology,” 1990. [Online]. Available: http://goo.gl/svqXqB. [Accessed: 31-Mar-2016].

[11] A. H. Soukhanov, K. Ellis, and M. Severynse, *The American Heritage Dictionary of the English Language*. Boston: Houghton Mifflin, 1992.

[12] Hoornweg, D., & Blaha, D. (2006). The Current Status of City Indicators. The World Bank.

[13] MyFIRE. (2011, May). Retrieved Jully 6, 2015, from D1.2 Taxonomy on common interpretation of testing, testing approaches and test beds models: http://www.my-fire.eu/documents/11433/38630/D1.2+-+taxonomy+on+common+interpretation+of+testing%2c%20testing+approaches+and+test+bed+models?version=1.0

[14] Project Management Institute. (2013). A Guide to the Project Management Body of Knowledge (5th Edition ed.). USA.

[15] Soukhanov, A. H., Ellis, K., & Severynse, M. (1992). The american heritage dictionary of the english language. Boston, MA: Houghton Mifflin.

[16] Carsten Bormann, Angelo P Castellani, and Zach Shelby. Coap: An application protocol for billions of tiny internet nodes. IEEE Internet Computing, 16(2):62–67, 2012.

[17] S. Jirka, A. Bröring, and C. Stasch. Applying ogc sensor web enablement to risk monitoring and disaster management. In GSDI 11 World Conference, Rotterdam, Netherlands, 2009.

[18] OpenIoT consortium, “Deliverable D4.3.2: Core OpenIoT Middleware Platform b”, 07 2014

[19] VITAL consortium, “Deliverable D3.2.2: Specification and Implementation of Virtualized Unified Access Interfaces V2”, 06 2015

[20] Z Shelby, K Hartke, C Bormann, and B Frank. Constrained application protocol (coap), draft-ietf-core-coap-13. Orlando: The Internet Engineering Task Force–IETF, Dec, 2012. Open

