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 V2

Document Id:	FIESTA-IoT-D34-170323-Draft
File Name:	FIESTA-IoT-D34-170323-Draft.pdf
Document reference:	Deliverable 3.4
Version:	Draft
Editor:	Nikos Kefalakis
Organisation:	Athens Information Technology
Date:	23 / 03 / 2017
Document type:	Report, Other
Dissemination level:	PU

DOCUMENT HISTORY

Rev.	Author(s)	Organisation(s)	Date	Comments
V01	Nikos Kefalakis	AIT	2016/12/02	Initial updated content
V02	Katerina Pechlivanidou	AIT	2016/12/07	Updated TPD/DMS interfaces. Added Message Buss installation Guide
V03	Nikos Kefalakis	AIT	2017/01/19	Updated TPS architecture and various sections
V04	Nikos Kefalakis	AIT	2017/02/06	Updated and added components sequence diagrams
V05	David Gomez	UNICAN	2017/02/21	Contributions and updates on the IoT-Registry components
V06	Hung Nguyen	NUIG	2017/02/23	Initial contributions for testbed and resource registration component
V07	Nikos Kefalakis	AIT	2017/02/24	Integration of contributions
V08	Hung Nguyen, Aqeel Kazmi	NUIG	2017/02/28	Updates on the testbed and resource registration component
V09	Nikos Kefalakis	AIT	2017/03/08	Integration and correction of contributions. Final updates to the APIs.
V10	Nikos Kefalakis	AIT	2017/03/19	Introduction, conclusion. Prepared and released document for internal review
V12	Aqeel H. Kazmi	NUIG	2017/03/21	Internal Technical review
V11	Paul Grace	ITINNOV	2017/03/22	Internal Quality review
V13	Ronald Steinke	FHG	2017/03/22	Internal Technical review
V14	Nikos Kefalakis	AIT	2017/03/22	Updates base on Technical & Quality Review
V15	Nikos Kefalakis	AIT	2017/03/23	Final Version for Submission
Draft	Martin Serrano	NUIG	2017/03/23	EC Portal Submitted

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TERMS AND ACRONYMS

3G	Third Generation of mobile telecommunication technology
AMQP	Advanced Message Queuing Protocol
API	Application Program Interface
CoAP	Constrained Application Protocol
dB	Decibel
DMS	Data Management Service
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
IR	IoT-Registry
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
TPI	Testbed Provider Interface
TPS	Testbed Provider Services
TRR	Testbed & Resource Registration
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 INTRODUCTION

1.1 Executive Summary

Among the main goals of 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 a 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. The TPI specification considers and comprises two different layers:

- A Configuration & Management layer that runs at the FIESTA-IoT platform side that is called Data Management Service (DMS) and controls the functionality of the Testbed Provider Services (TPS) by utilizing the offered user interface for the User (Testbed provider).
- A Testbed Provider Service (TPS) 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 mainstream 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) has been also considered since they provide valuable insights on the nature and properties of a platform/testbed agnostic TPI. The review of existing testbeds and platform-agnostic interfaces, as a key element of the methodology that led to the TPI specification have been included in the first version of the deliverable [2] and not in this document in order to avoid repetition.

Additionally this deliverable presents the components and APIs that enables the data storage and retrieval within the FIESTA-IoT platform. These components are utilised from the TPI services in order to discover resources and successfully manipulate the retrieved data. Finally the required tools for registering testbeds and resources are presented which provide a user-friendly manner for the testbed providers to register and manage their resources.

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 which also has been included in the first version of this deliverable [2].

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 Audience

This deliverable addresses the 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.

1.3 Structure

In addition to this introductory section, the deliverable comprises the following sections:

- Section 2 provides an overview of the FIESTA-IoT Testbed Provider Interfaces and the relative components. The section provides the placement of the components within the FIESTA-IoT architecture along with examples in the form of sequence diagrams describing the interactions between them.
- Section 3 focuses on the User Interfaces that facilitate the resource & testbed registration and configuration of the testbed provider interface. It provides a description of them along with a user manual.
- Section 4 specifies the different components, described in the previous sections, APIs.
- Section 5 identifies and describes the core technologies used for the materialization of the main components of the testbed provider interface family.
- Section 6 provides the prototype implementation source code availability along with requirements and installation manual.
- Section 7 finally provides the deliverable's conclusions.

1.4 Updates from the previous version

This deliverable constitutes a second and final iteration from Deliverable D3.3 [2]. The main changes with respect to the previous iteration are summarised below:

- Introductory sections regarding FIESTA-IoT scope, WP3 overview and terminology & definition have been removed in order not to repeat material from other deliverables.
- Section regarding background technologies and available interface analysis has been removed since there were no updates.
- The architecture has been updated and additional sequence diagrams explaining the interactions have been added.
- New components have been added to the deliverable described in sections 2 and 4.
- Existing components have been updated as far as their description and API is concerned.
- Existing User interfaces (TPI Configurator) have been updated and have also been enhanced with additional ones (Testbed and Resource Registration).
- Finally prototype implementation source code availability and installation manuals have also been updated.

2 TESTBED PROVIDER INTERFACE AND PREREQUISITE COMPONENTS PLACEMENT & DESCRIPTION.

2.1 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 realms (Figure 1). The first is the FIESTA-IoT platform side with the TPI Configuration & Management layer that controls the functionality of the TPI by utilizing the offered user interface for the User (Testbed provider). The second is the Testbed Side with 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. Along with the two aforementioned layers we identify additional layers and tools which are utilized from the TPI components at the configuration and runtime. These layers are the Data Validation & Storage that resides at the core of the FIESTA-IoT platform and hosts the repositories along with tools and interfaces that enables the storage and retrieval of the annotated data. And the Testbed & Resource registration layer which enables the testbed provider and other components within the platform to register and manage testbeds and resources.

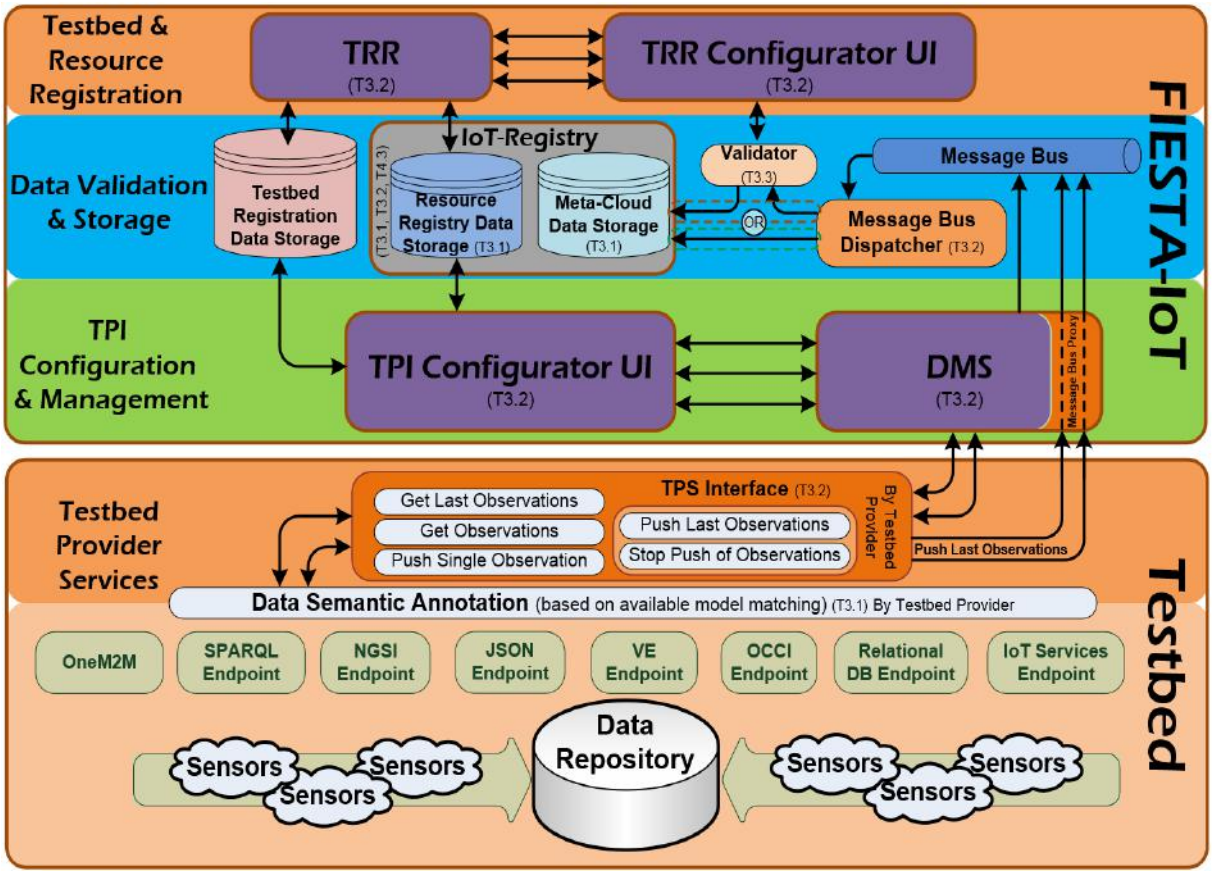

Figure 1 TPI Architecture

As we can see in Figure 1 above a testbed may expose internally various standard and/or proprietary interfaces in order to interact with the sensor data. FIESTA-IoT has specified a list of core services (TPS) that should be exposed by a testbed in order to enable different connection methods to the platform. These services (i.e. getObservations) should be exposed by 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 by 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. Push Last Observations from Figure 1). 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 by the Testbed & Resource Registration (TRR) component which stores the information in the Testbed Registration Data Storage (non-semantic data) and Resource Registry Data (semantic data) Storage (see Figure 1 above). To retrieve the information stored in this data storage we are using the services that are identified by the IoT-Registry and the Testbed & Resource Registration (TRR) components. The IoT-Registry, TRR and TPI DMS services are utilized by the TPI Configurator (Figure 1 above) which is a User Interface (UI) component. The TPI Configurator enables the testbed provider to discover the semantically registered resources, and manage the data retrieval process by utilizing the TPI DMS.

In the sections below we provide a short introduction and a list of the different services exposed by the components that where mentioned here. Moreover we provide a more detailed description and a sequence diagram of the process and dataflow described above.

2.2 Configuration and Runtime Sequence example

The required steps and interactions between the Testbed provider and the different FIESTA-IoT components in order to set up and integrate a Testbed to the system is depicted in Figure 2 to Figure 5 below. Where we can see the Testbed & Resource registration and the TPI configuration and runtime. For the Testbed setup we presume that the Testbed Provider has already generated the Data Semantic Annotator and initiated an instance of the Testbed Provider Interface for his/her Testbed.

For the Testbed Registration as depicted in Figure 2 below:

1. The first step is to open the Testbed Registration UI (preferably thru the FIESTA-IoT portal). If the Testbed Provider is not logged in yet the User credential will be requested to be entered.
2. The credentials are forwarded to the Security component where a SSO token id is generated and sent back to the container (browser) that the Testbed Registration UI is opened from.
3. This SSO token id is then used from the Testbed Registration UI in order to retrieve from the security module information like the UserID, User role etc.

4. With the user ID, the Testbed Registration UI now contacts Testbed Registration Data Storage (TRR) in order to retrieve potentially other Testbeds info that has already been registered from the specific Testbed Provider (by UserID).
5. The Testbed Provider now initiates the process of registering a new Testbed.
6. An already annotated resource script (representation) has to be entered to the form in order to be validated.
7. The resource script is send from the Testbed Registration UI to the Validator component and a report is generated with a success or no success message from it.
8. The rest of the Testbed information are entered to the Testbed Registration UI form like the Testbed' s TPS endpoints along with the API key (if it exists).
9. By hitting the register button the information are stored both on the Testbed Registration data storage (TRR) and the Resource Registry data storage (IoT Registry) depending on the non-semantic (TRR) or semantic (IoT-Registry) nature of it.

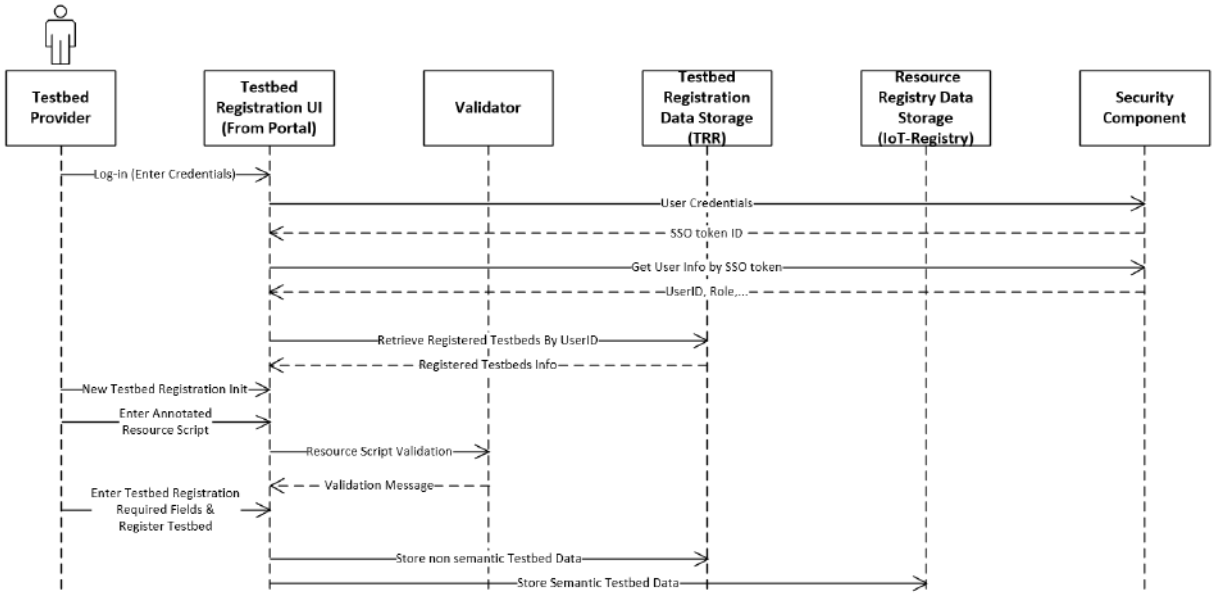

Figure 2 Testbed registration Sequence Diagram

An equivalent process for the Resource registration shall be followed and is depicted in Figure 3 below :

1. The first step is to open the Resource Registration UI (preferably thru the FIESTA-IoT portal). If the Testbed Provider is not logged in yet the User credential will be requested to be entered and the next two steps are the same as step 2 and 3 of the Testbed Registration Sequence above.
2. With the user ID the Resource Registration UI now contacts Testbed Registration Data Storage (TRR) in order to retrieve all the testbeds registered from the specific Testbed Provider (by UserID).
3. The Testbed Provider can now choose the Testbed of interest from the presented list. If no testbed is registered yet then the Testbed registration process should be followed to register one.

4. With the Testbed ID the Resource Registration UI is now contacting Testbed Registration Data Storage (TRR) in order to retrieve potentially other resource info that have already been registered for the specific Testbed.
5. The Testbed Provider can now initiate the process of registering a new Resource.
6. An already annotated resource script has to be entered to the form in order to be validated.
7. The Resource Registration UI to the Validator component sends the resource script and a report is generated with a success or no success message from it.
8. The rest of the Resource information is entered to the Resource Registration UI form.
9. By hitting the register button the information are stored both on the Testbed Registration data storage (TRR) and the Resource Registry data storage (IoT Registry) depending on the non-semantic or semantic nature of it.

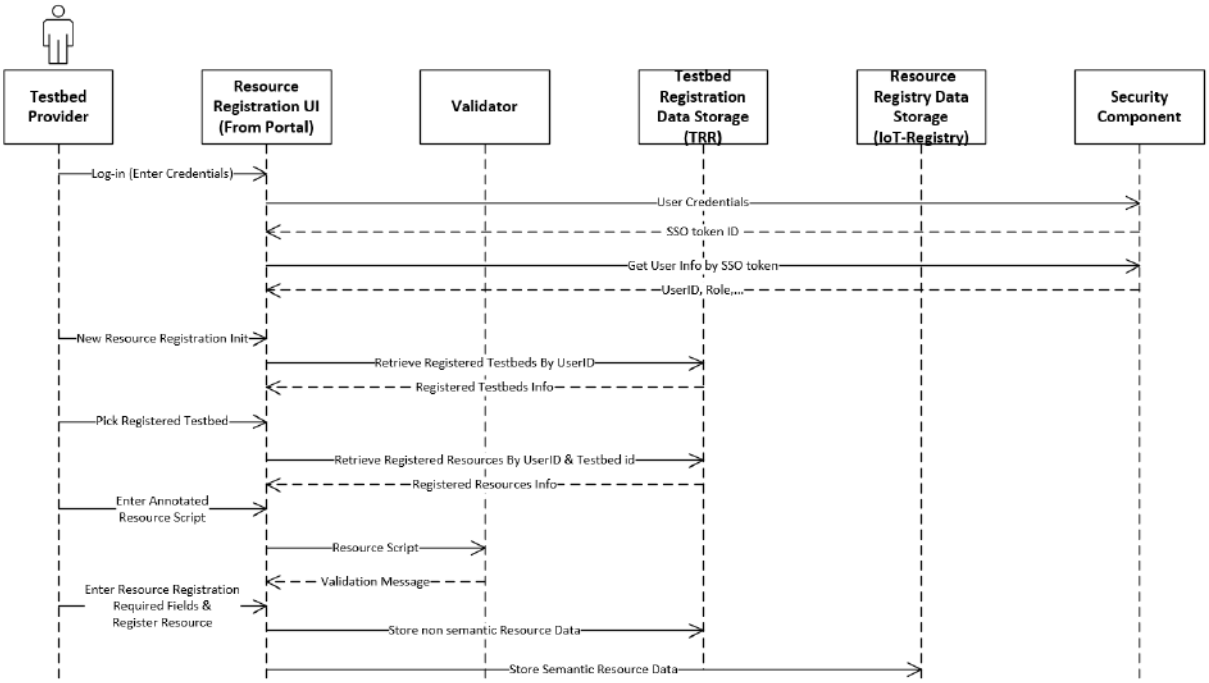

Figure 3 Resource registration Sequence Diagram

Now we can move to the TPI Configuration sequence shown in Figure 4 below:

1. The first step is to open the TPI Configuration UI (preferably thru the FIESTA-IoT portal). If the Testbed Provider is not logged in yet the User credential will be requested to be entered and the next two steps are the same as step 2 and 3 of the Testbed Registration Sequence above.
2. With the user ID the Resource Registration UI now contacts Testbed Registration Data Storage (TRR) in order to retrieve all the testbeds registered from the specific Testbed Provider (by UserID).
3. The Testbed Provider can now choose the Testbed of interest from the presented list. If no testbed is registered yet then the Testbed registration process should be followed to register one.
4. When the Testbed Provider picks a Testbed ID the TPI Configuration UI contacts the Resource Registry Data storage (IoT-Registry) in order to retrieve

the list of registered resources for the specific testbed. If there are no registered resources the user should follow the Resource registration process to register some.

5. The User can choose a list of resources to set up, identify an execution schedule and choose the TPS retrieval method (i.e. getObservation). Finally all these info can be sent to the DMS in order to start a scheduled job which is going to retrieve the annotated measurements from the TPS and forward them to the FIESTA-IoT platform (which is described below).

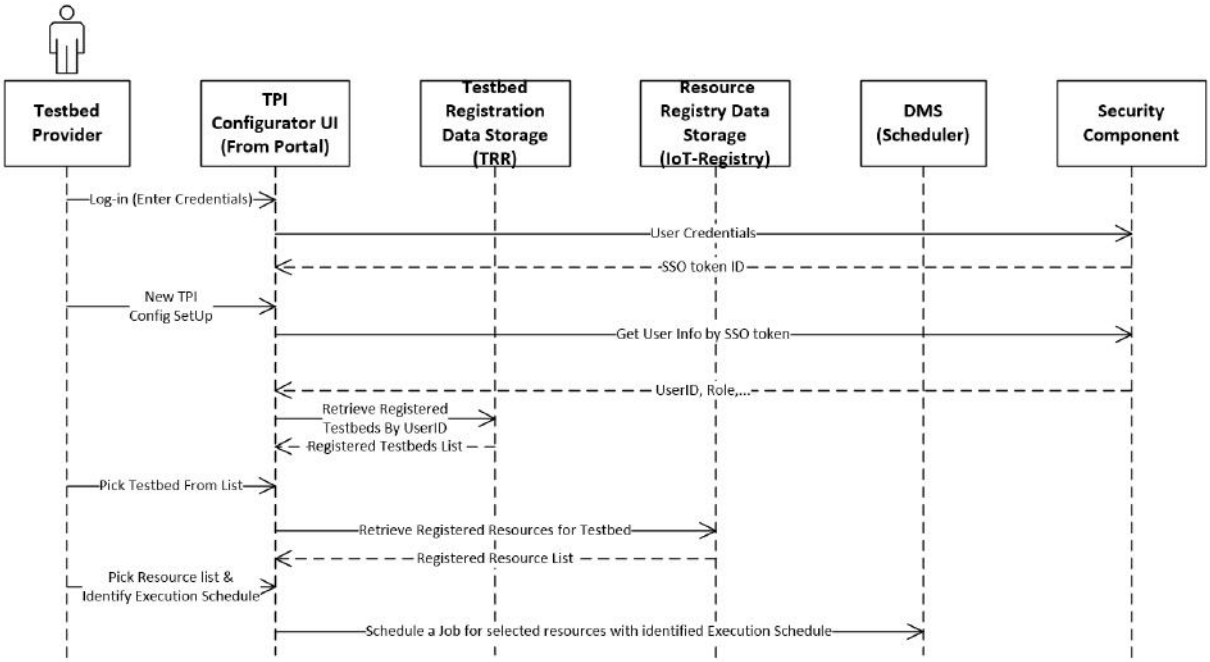

Figure 4 TPI Configuration Sequence Diagram

Finally a runtime sequence of acquiring the annotated measurements and forwarding them to the FIESTA-IoT platform is depicted in Figure 5 below. In this example, we presume that the testbed TPS instance has been implemented and exposes the "getObservations" services, which provides the FIESTA-IoT annotated measurements of a given list of resources for a specific time period.

- After successfully initiating a DMS scheduled job as described in the sequence diagram above the DMS scheduler enters a loop with the identified time period from the Testbed provider. In every loop the DMS sends a message to the identified Testbed TPS with a list of resource IDs and a timeperiod (from->to time) that the TPS needs to produce the annotated measurements.
- TPS replies back to DMS with an FIESTA-IoT annotated document with all the requested measurements.
- DMS collects this document and forwards it along with the annotation type to the Message Bus under a specific topic.
- In the meantime the Message Bus Dispatcher (MBD) has subscribed to the above topic so every time something is published to it the MBD gets informed and receives the message that was pushed to the Message Bus from the DMS.

- The collected annotated document is now pushed to the Validator and if the document is semantically and syntactically valid is forwarded to the Meta-Cloud Data storage (IoT-Registry) for storage.

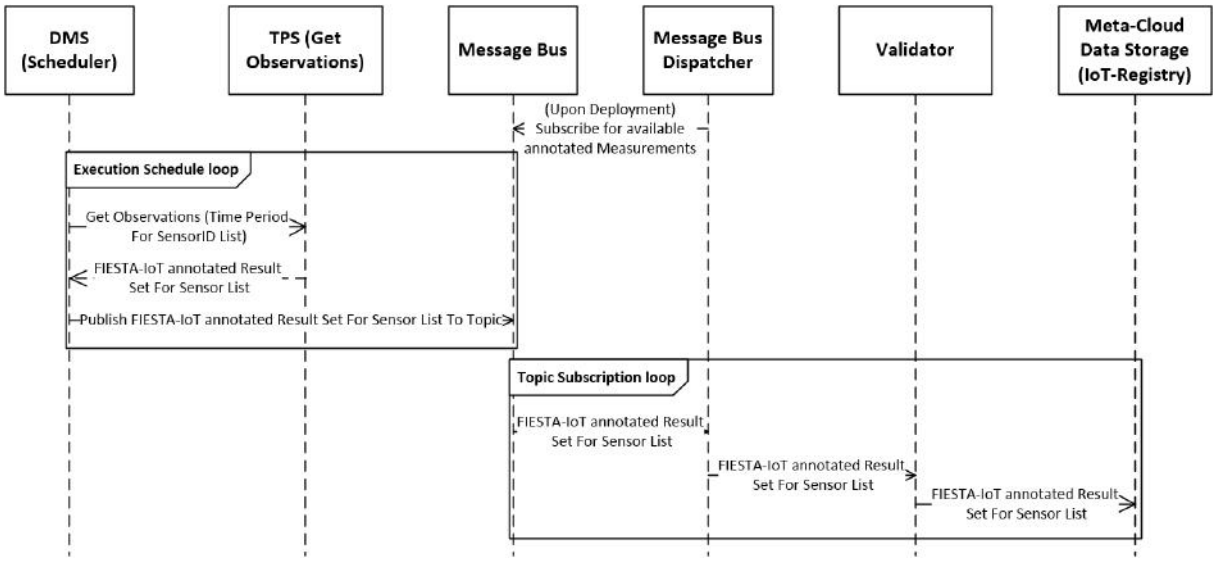

Figure 5 TPI Runtime Sequence Diagram example (getObservations)

2.3 Offered Components and Services

As mentioned above, the TPI is a set of RESTful web services and tools that enables the integration of the testbeds to the FIESTA-IoT platform. These services are offered to the Testbed Provider in order to enable the "plugability" and management of their testbeds in relation with the FIESTA-IoT platform. Along with the TPI components we also have other components that facilitate TPI's functionality, among others, within the platform. These components along with the TPI ones are listed below and are analysed in the next section of the document:

- IoT-Registry (IR)
  - Core services
    - Testbeds
    - Resources
    - Observations
    - Queries
  - Utility services
    - Annotator as a Service
    - Identity Mapper
- TPI Core Services
  - Testbed Provider services (TPS)
  - Data Management services (DMS)
- TPI Auxiliary services
  - Testbed & Resource registration (TRR)
  - Message Bus Dispatcher (MBD)

2.3.1 IoT Registry (IR)

2.3.1.1 IoT Registry Core

The FIESTA-IoT meta-directory, which is the core component in charge of handling (i.e. offering services and storing RDF data) the semantic information that flows across the FIESTA-IoT platform is mainly managed by the so-called "IoT-Registry" module. It basically controls the triple-store that internally holds the aforementioned meta-repository. The IoT-Registry has been implemented as a REST web service, thus providing a fully-fledged set of interfaces that support a CRUD approach. In addition to this, it is worth highlighting that this module relies on the Jena[1] framework for the handling of the triple-stores that take over the storage of the information and its interaction with other components.

As a matter of fact, this module turns out to be the sink where all the information coming from the testbeds (mostly, resource descriptions and observations) is stored. Moreover, it introduces a way to query (filter) the information, through some services that operate under a SPARQL engine. This comes to offer a flexible and standard interface to access the information stored in the triple-store (i.e. federated testbeds' resource descriptions and observations).

In this deliverable, we mainly focus on the subset of services that have to do with the actual interaction between testbed providers and the FIESTA-IoT meta-directory[2]. For a full vision on the API, the reader should refer to [5] and, for an interactive documentation[3]. Henceforth, all the services behind this IR (IoT-Registry) API will hang from the following path: https://platform.fiesta-iot.eu/iot-registry/api.

Now we have introduced the module itself, it is time to describe each of the services that have to do with testbed providers. Nonetheless, most of these offered services will be transparent in the eyes of testbed providers, since the actual proxy (between the platform and them) will be the DMS and not the IoT Registry. To do this, we group the different services based on the actuation domain, providing a brief description of each of them:

- **Testbeds (*/testbeds*)**. On this realm, the TRR module holds this operation, and the following list of services is to be carried out between IR and TRR.
  - **Register Testbed:** When a testbed wants to become part of the FIESTA-IoT federation, it *MUST* be first registered into the platform. Behind this service, two operations are carried out: one held at the TRR side, and the other, much simpler, at the IR, where it is only stored (in the triple-store) a graph with the testbed ID. It goes without saying that a univocal connection between each other is compulsory.
  - **Get the list of registered testbeds.** Fetch the whole set of testbeds currently present in the IR.

[1] https://jena.apache.org/

[2] All the operation will be carried out through external components, like DMS or TRR, so external providers will not be able to directly use most of IoT-Registry's services.

[3] https://platform.fiesta-iot.eu/iot-registry/docs/api.html

    - **Get a testbed description (from the IR)**. Retrieve the semantic description of the node that has defined upon a testbed registration (subject/predicate/object).
    - **Get a testbed's list of resources**. Retrieve the list of resources that have been registered under a particular testbed ID.
- **Resources (*/resources*)**. All the information concerning resources is managed here. In this case, the DMS will be between the IR and testbed providers.
    - **Register a resource (or a group of them).** As its own name hints, this service is in charge of storing a resource from a FIESTA-IoT compliant description document. It is worth highlighting that only validated documents will be up to be registered (for more information, refer to [5]).
    - **Get the list of registered resources.** Fetch the whole set of resources present in the IR at the moment of executing the service.
    - **Get a resource description from an individual ID**. Alike for testbeds, one can fetch the semantic information associated to a resource.
- **Observations (*/observations*):** Whereas the resources realm has its own domain, the API offers a mostly alike set of services for the data generated by the sensors. One more time, this group will not be directly available to regular users, and the regular injection of data must be done through the TPS/DMS channel.
    - **Store an observation.** A document containing the information pertaining to an observation (or a burst of them) is processed by the IR and saved in the corresponding triple-store.
    - **Get the list of stored observations.** Fetch the whole set of observations stored up to the moment of executing the service.
    - **Get the description of an observation.** From a particular ID, the semantic information about any of the nodes that shape an observation can be retrieved.
- **Queries (*/queries*):** Aside the unidirectional process of injecting data to be stored into the triple-stores managed by the IR, the API permits the extraction of information
    - **Run a SPARQL query.** The most common query language for retrieving data in semantic-based storage systems is SPARQL. For this reason, we offer the possibility to manually execute SPARQL queries against the IR module so that the inherent response will be sent back to the user.
    - **Get a list of predefined queries (descriptions).** Potentially, not a high ratio of users might have a good background in semantics. Hence, the IR incorporates a space where widespread SPARQL queries are stored in a kind of open catalogue so that non-skilled users might directly grab them for a further use.
    - **Save a query to the catalogue.** We have left the door open to saving any SPARQL into the so-called "catalogue". This way, everybody is welcome to share their queries and help others request data from the IR.
    - **Run a particular query from the catalogue.** After fetching all the queries, one can directly execute one of these queries and get the actual result of the SPARQL itself.

- **Update a particular query.** Being an open system, it is likely to happen that multiple similar queries are stored, thus almost replicating their functionality. Therefore, the possibility of modifying them is something that users might really need.
- **Delete a particular query.** The same way that we allow the modification of queries, we also permit their elimination. Of course, only authorized users are able to execute this operation (as well as the previous one).

2.3.1.2 IoT Registry (IR) Utilities

Along with all the services provided by the iot-registry module that deals with the push/pull of data to to/from the triple-stores, we have also included some extra features in order to make experimenters' and testbed providers' lives easier.

2.3.1.2.1 Annotator as a Service

As was described in [5], the default option for converting the datasets from testbeds' native format, in case they followed a proprietary approach, to FIESTA-IoT semantic model is through the implementation of tailored annotators to each of the potentially heterogeneous testbeds. However, the iot-registry supports, as a helping tool, a generic translator that can create actual FIESTA-IoT-compliant resource descriptions and observations, provided that the input data matches the format defined for these web services (see below). However, it is worth highlighting that the nature between these web services is to provide a generic way to annotate information, thus complex stuff might not be well suited through this approach.

In a nutshell, the operation of these services works like this: the users sends a POST message to the IR module, which generates the corresponding RDF and FIESTA-IoT compliant object from the information received. Then, an output document will be sent back to the users, who will be in charge of doing whatever with it (e.g. registering a resource, an observation, etc.).

This feature introduces three different services, described below:

- Annotate a resource
  - Testbed providers can address the registration of resources by means of this service, where the API will generate a FIESTA-IoT compliant resource description from a straightforward non-semantic object.
- Annotate a testbed (plus resources)
  - On top of the annotation of a resource (or a group of them), we have also defined the possibility of binding an array of assets to their actual testbed owner.
- Annotate an observation
  - Alike the previous case, testbed providers could also generate FIESTA-IoT compliant data from one of our services. In this last case, annotated observations could be as well made thanks to this API.

2.3.1.2.2 Identity mapper

An atomic process that is carried out upon every testbed/resource/observation registration is the “anonymization” of its legacy identifier. For deeper information about this operation, the reader should refer to [5]. In essence, FIESTA creates a unique identifier instead of the legacy one used by the underlying testbeds. Nonetheless, this operation has an “inverse”, so users are able to retrieve the original ID or vice versa. Therefore, these are the two services:

- (Native) Testbed to FIESTA-IoT
  - Actual operation carried out by the IR upon a resource registration or injection of observations (in case they define individuals). Taking the node IRI as input, the API returns the “codified” identifier.
- FIESTA-IoT To testbed (Native)
  - Reverse operation to retrieve the legacy testbed’s identifier from that of FIESTA-IoT’s.

2.3.2 Testbed Provider Services (TPS)

FIESTA-IoT has specified a list of services of which at least one (get or push) 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 in an FIESTA-IoT annotated document once.
- **getObservations**
  - This service provides the values of a specific Sensor list for a specific time-period in an FIESTA-IoT annotated document once.
- **pushLastObservations**
  - This service initiates a stream at the Testbed side which pushes continuously the latest values of a specific Sensor list to a specific endpoint in an FIESTA-IoT annotated document.
- **pushSingleObservation**
  - This service pushes continuously the latest value of a specific Sensor to a message bus with the Sensor ID as queue topic.
- **stopPushOfObservations**
  - This service stops the push of observations initiated by the “pushLastObservations” and “pushSingleObservation”. This service MUST be implemented in combination with the “pushLastObservations” and “pushSingleObservation” services.

2.3.3 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 (TPS) in order to retrieve the data of the registered resources.

These services are:

- **subscribeToObservations**
  - This service gets the values of a specific Sensor list from TPS and pushes them to a specific endpoint based on a specific execution schedule. In every execution when all the sensor measurements have to be reported the time period starts from the last execution.
- **unsubscribeFromObservation**
  This service unregisters a list of Sensors from an endpoint which has been initiated with the subscribeToObservations.
- **subscribeToObservationStream**
  - This service instructs the TPS to **push** the values of a specific Sensor list to a specific endpoint (pushObservationsStreamProxy) as soon as they have new values.
- **subscribeToObservationStreamWithTopic**
  - This service instructs the TPS to push 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.
- **unsubscribeFromObservationStream**
  - This service stops the pushing of Observations from TPS of a specific sensor list to an end point. Essentially it stops the subscribeToObservationStream and subscribeToObservationStreamWithTopic.
- **pushObservationsStreamProxy**
  - This service provides an Endpoint in order the Testbeds to push their annotated measurements to the FIESTA-IoT message bus. Essentially this service creates a "proxy" between the Testbed Provider (TPS) and the Message Bus. This service is used in combination with the subscribeToObservationStream.
  -

In Figure 6 below we can see a simplified diagram of the different DMS-TPS service interaction for applying the different DMS functionalities described above.

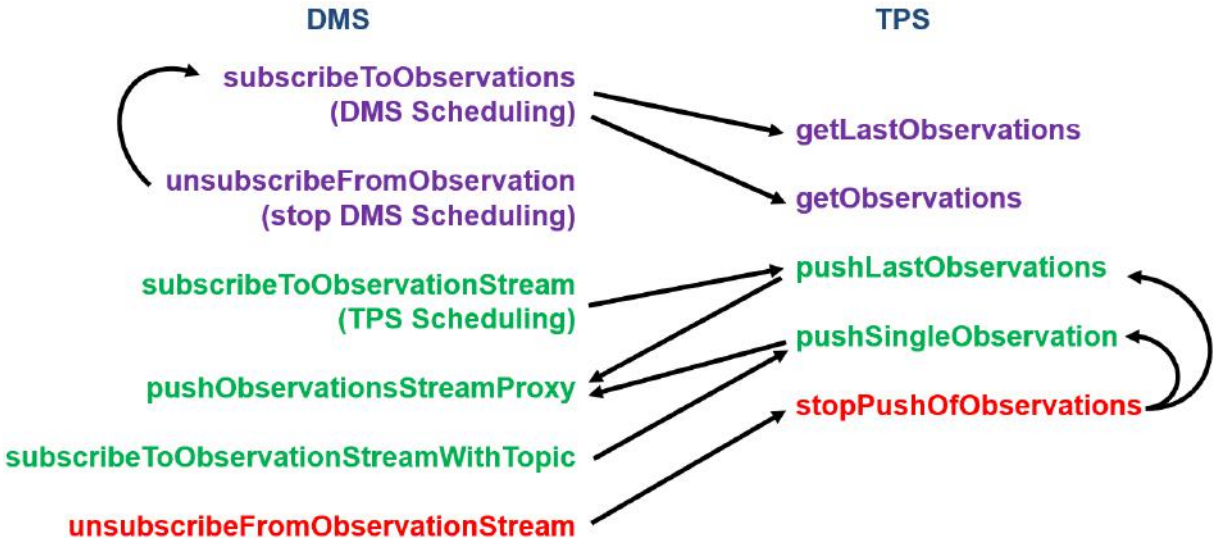

Figure 6 DMS-TPS service interactions

In order to control the required internal scheduling of the DMS and to facilitate the functionality of subscribeToObservations and unsubscribeFromObservation additional services has been defined and are exposed from the DMS component. These services are the following:

- **getAllScheduledJobs**
  - This service provides the ability to retrieve all scheduled jobs of a specific user on the DMS.
- **deleteScheduledJob**
  - This service enables the deletion of a specific scheduled job with the given ID.
- **pauseScheduledJob**
  - This service provides the ability to pause a scheduled job.
- **resumeScheduledJob**
  - This service provides the ability to resume a scheduled job.
- **getScheduledJobs**
  - This service Returns all the jobs that are scheduled on the DMS.
- **getMetadataForScheduledJob**
  - This service provides the ability to retrieve the metadata of the scheduled job from the given id
- **deleteAllScheduledJobs**
  - This service enables the deletion of all the scheduled jobs on DMS.
- **getCurrentlyExecutingJobs**
  - This service provides the ability to get all currently executing jobs. Note that this method returns all jobs that are executed at the exact point when the rest service is requested.

2.3.4 Testbed & Resource registration (TRR)

FIESTA-IoT has specified a list of services that will enable to register testbed and resource the services exposed by the testbed in order to retrieve the data of the registered testbed, resources. These services are:

- **createRegisterDevices**
  - This service provides the ability to register devices manually from a specific user.
- **createRegisterDevicesByText**
  - This service provides the ability to register devices by input text in RDF format from a specific user.
- **createRegisterDevicesByUpload**
  - This service provides the ability to register devices by upload document in RDF format from a specific user.
- **createRegisterTestbeds**
  - This service provides the ability to register testbed by input testbeds information from a specific user.
- **getAllTestbedsByUserID**
  - This service provides the ability to get all the registered testbeds registered from a specific user.
- **getAllTestbedsRegisterByRegisterIDList**

    - This service provides the ability to get all testbeds by input list of registerID.
- **getAllTestbedsRegisterIDsByUserID**
    - This service provides the ability to get all registerIDs by input specific userID.
- **getTestbedByRegisterID**
    - This service provides the ability to get a testbed by registerID by input registerID.
- **getAllRegisterTestbeds**
    - This service provides the ability to get all testbeds by input params page and size default page 0 and size 20.
- **getRegisterTestbeds**
    - This service provides the ability to get a testbed by ID.

2.3.5 Message Bus Dispatcher (MBD)

The Message Bus Dispatcher is a relatively simple component which collects the messages pushed to the message bus (see Figure 1 above) and forwards them to the IoT-Registry. Upon deployment the MDB subscribes to the Message Bus queue that the annotated measurements are pushed from DMS and TPS. As soon as a new measurement is pushed to the Message Bus the MBD gets informed and collects it in order to forward it to the IoT-Registry either directly or thru the validator as shown in Figure 1 above. The MDB receives the messages from the Message Bus in a Text format and it forwards them following the appropriate message header. When MBD collects a message from the Message Bus it also removes it from the queue.

3 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.

3.1 Testbed & Resource registration User Interface (TRR)

In this section, we introduce the User Interfaces that are provided by FIESTA-IoT to facilitate the testbed and resource registration process within the FIESTA-IoT platform.

3.1.1 Testbed Registration

The Testbed Registration UI can be found at the FIESTA-IoT portal under the Testbed Provider Menu (see Figure 7 below) item named the “Register Testbed”. Register Testbed page (see Figure 8 below) will show the list of registered testbeds and enable user to register a new testbed. The admin user can view all the registered testbeds, whereas the testbed providers only see their own testbeds that they have registered.

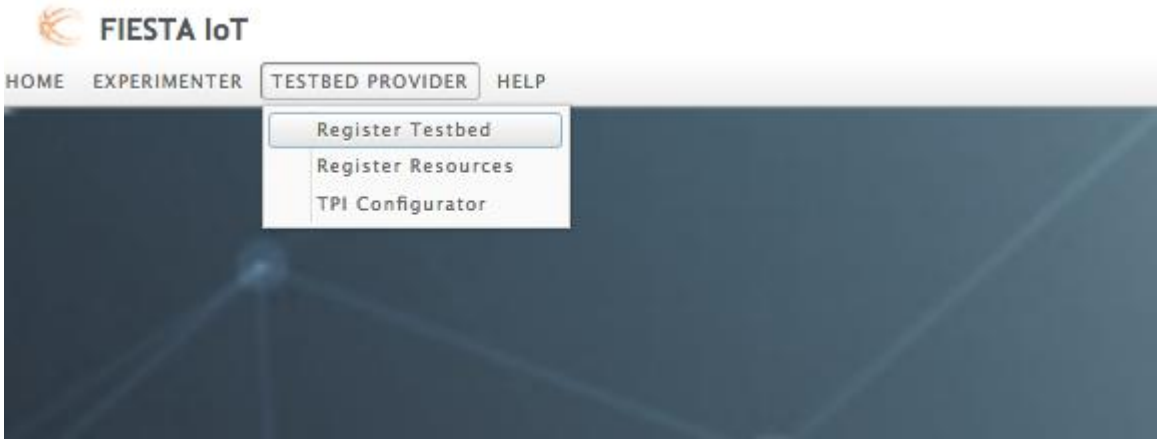

Figure 7 Testbed Registration at the FIESTA-IoT portal

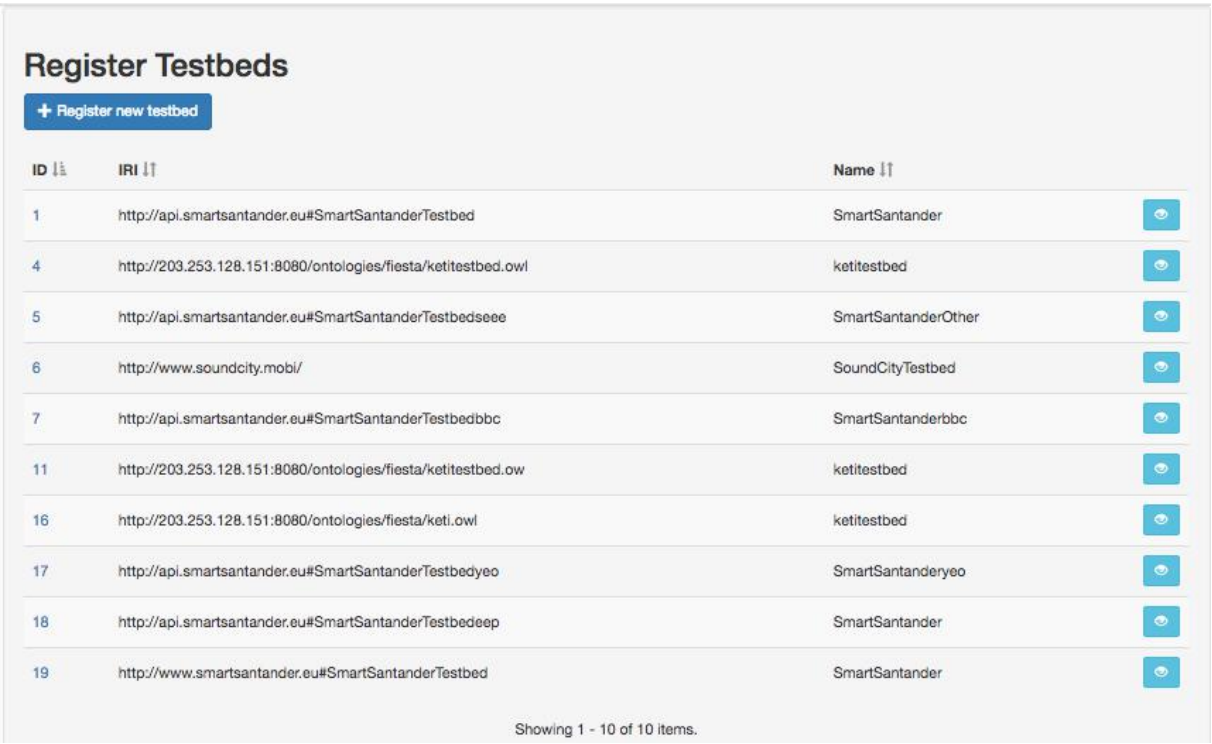

Figure 8 Testbed Registration UI

The user can register a new testbed by clicking “Register new testbed” button, a register testbed page (see Figure 8 above) will be shown where the user inputs testbed related information (Figure 9 below):

- Input IRI, this is unique for each registered testbed.
- Input Annotated Resource Observation, Annotated Resource Description user must specify data format by selecting a format from combo-box list.
- User can validate resource observation and resource description by clicking validate button.
- User must input latitude, longitude.
- Other remaining textbox are optional.

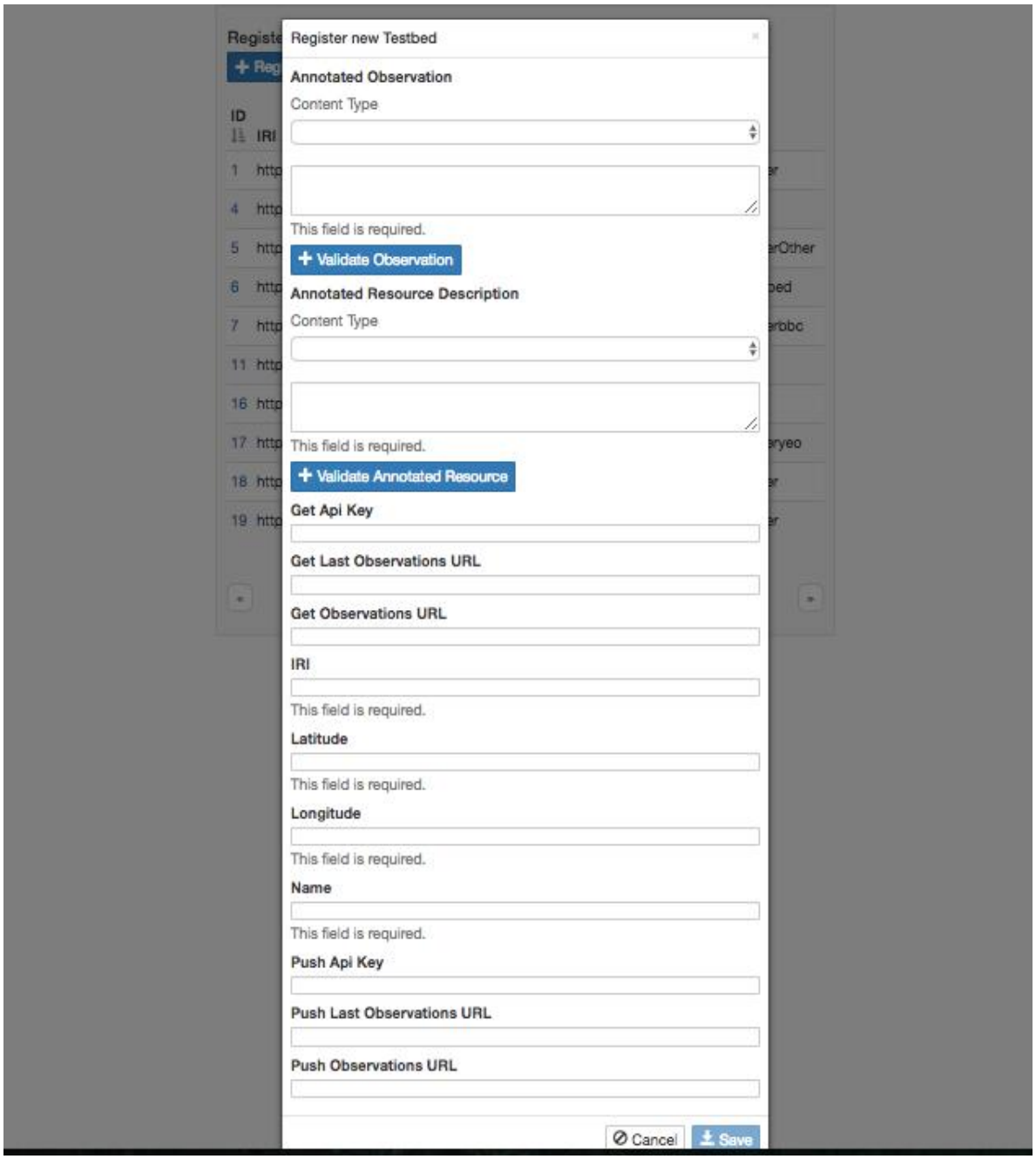

Figure 9 Registering a new testbed

When the user clicks the save button, if the registration process is successful the register testbed page will disappear and a success message is displayed along with the identifier of the registered testbed (see Figure 10 below).

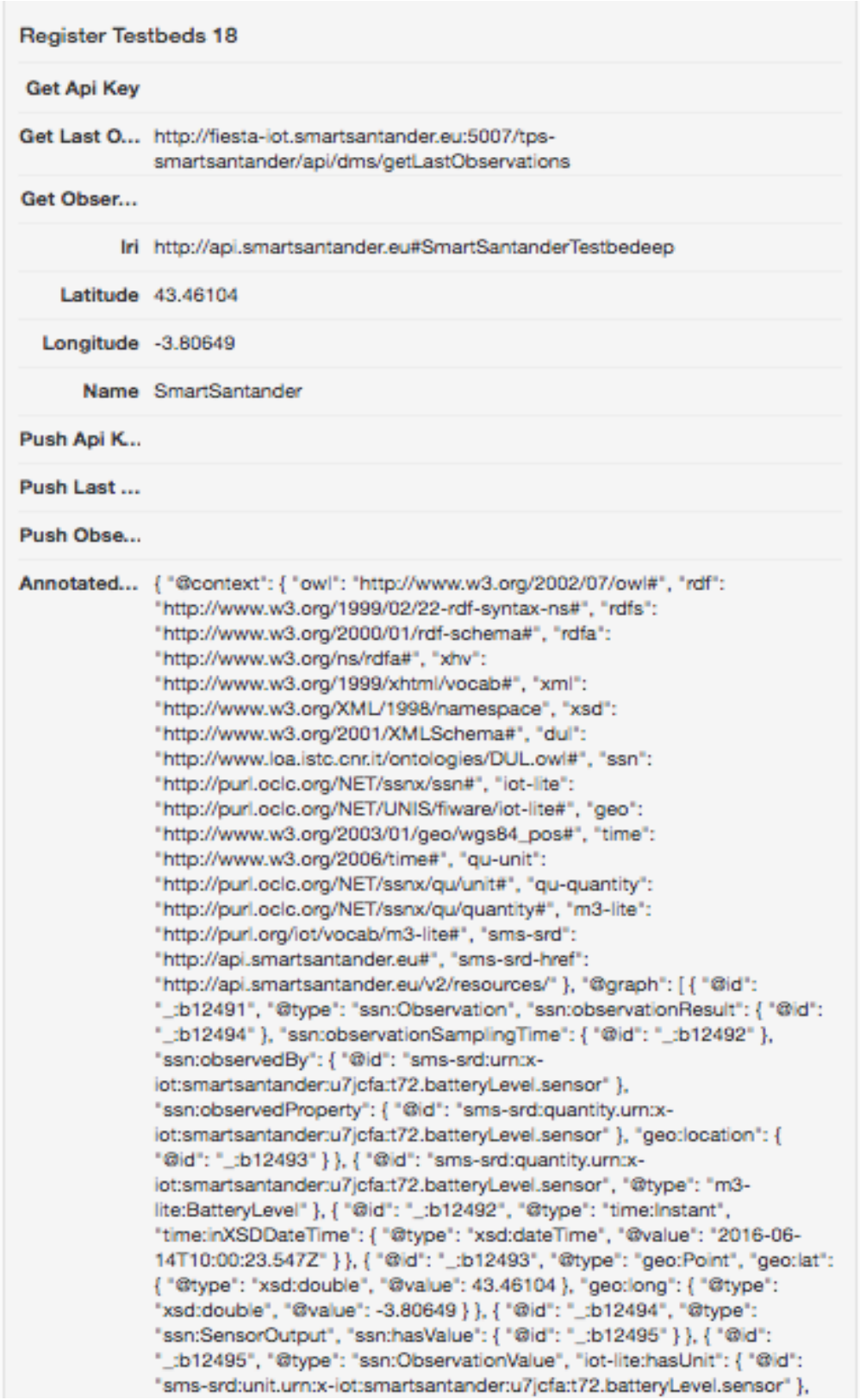

Figure 10 Registering a new testbed

3.1.2 Resource Registration

After registering a testbed with the specified IRI, a user can use this IRI for registering resources related to the testbed. Register Resources can be found at the FIESTA-IoT portal under the Testbed Provider Menu item named the “Register Resources”. Once the user chooses this option, resource registration can be done through three options (see Figure 11 below):

- Register Devices Manual,
- Register Device by Text,
- Register Devices by Upload

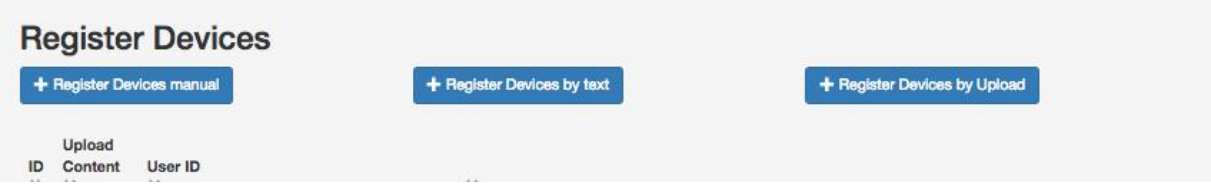

Figure 11 Registering resources

3.1.2.1 Register Devices Manually

When registering devices manually, the user must select a testbed (IRI) for which he/she wants to register devices manually (see Figure 12 below).

Figure 12 Register Devices Manual

After selecting the registered testbed IRI, user clicks on ‘Register Devices manual’ button and a form is displayed (see Figure 13 below).

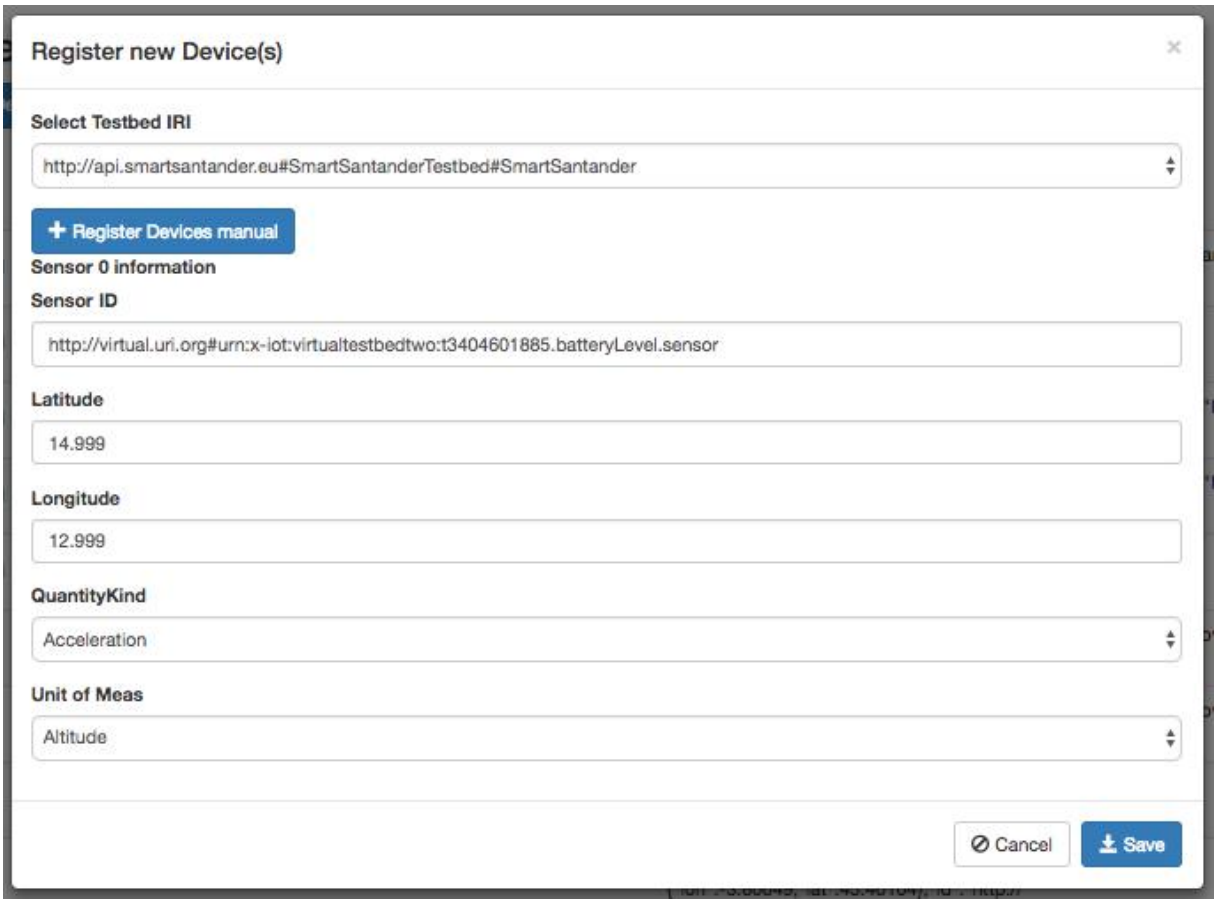

Figure 13 Register Devices Manual

After entering the information related to device manual the user clicks the save button to complete the devices manual registration process. Upon success, the user will return back to the main options window of 'Register Resources' (see Figure 14 below).

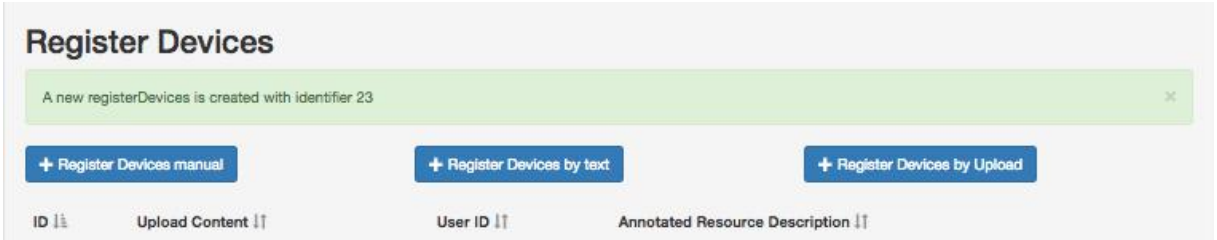

Figure 14 Register Devices Manual success

3.1.2.2 Register Devices by text

When registering devices by text, the user must specify the format and then input the text into the text-area (see Figure 15 below).

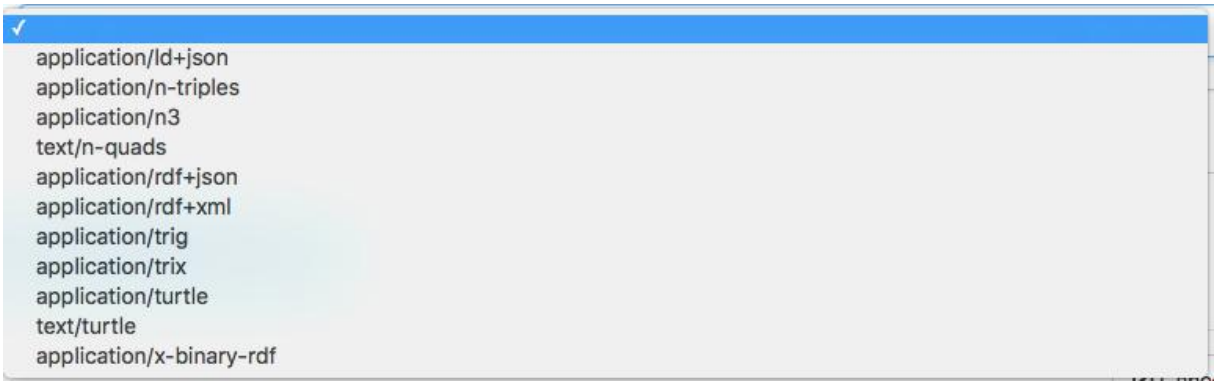

Figure 15 specifying the registration format

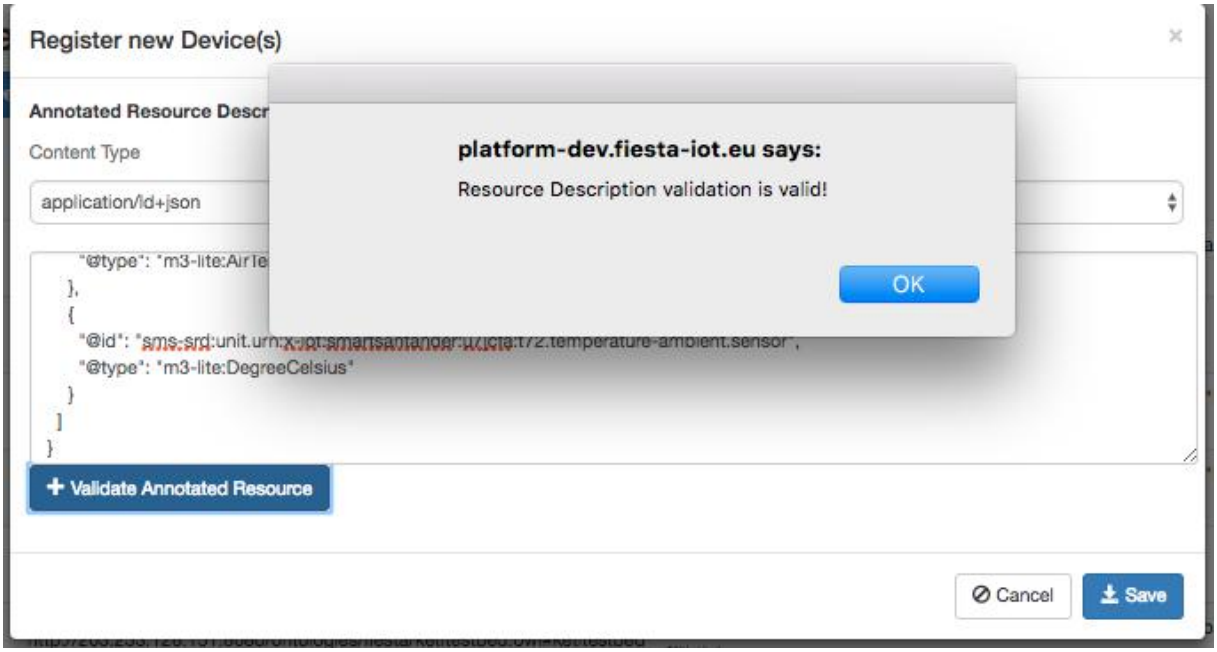

Figure 16 Register Devices by text

After providing the text for registering a device, the user clicks the save button (Figure 16 above). Upon success, the user will return back to the main options window of 'Register Resources' (see Figure 17 below).

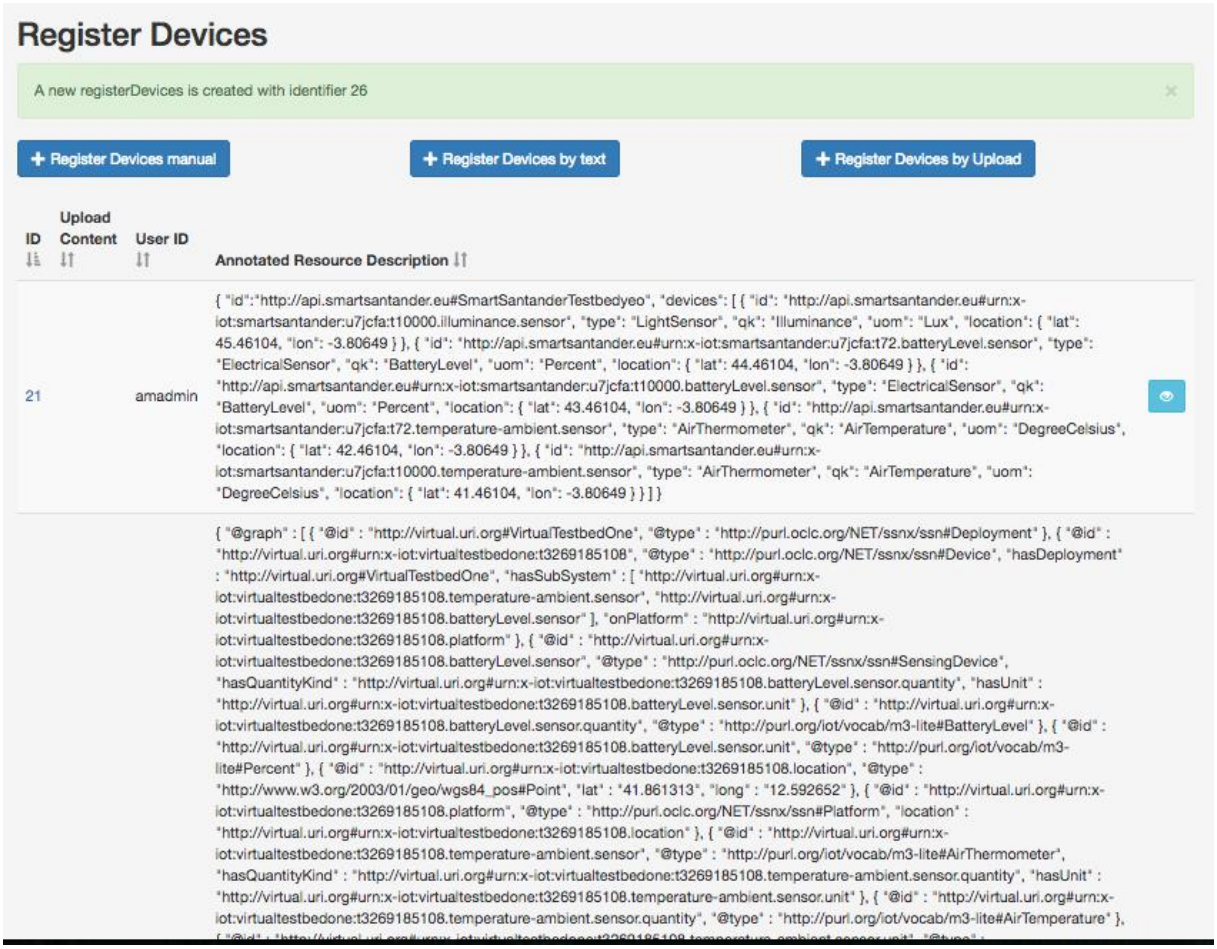

Figure 17 Register Devices by text success

3.1.2.3 Register Devices by Upload

When registering devices by Upload, the user must select the format of the document and then upload the file to register resource (see Figure 18 below). The current max size of the file upload is set to 10MB.

Figure 18 Register Devices by Upload

The system will only register resources after validating the document. If the process was successful on save, the window register by upload will disappear and user will return to the Register Resources page with a green message (see Figure 19 below).

Figure 19 Register Devices by Upload success

3.2 TPI Configuration and management

As mentioned in section 2, FIESTA-IoT will offer a TPI configuration UI that will enable the testbed provider to discover the semantically registered resources of his/her testbed, by utilizing the IoT-Registry services. Moreover it will manage the data retrieval process, by utilizing the TPI DMS services.

The TPI Configurator UI can be found at the FIESTA-IoT portal under the Testbed Provider Menu (see Figure 20 below) by clicking the “TPI Configurator”.

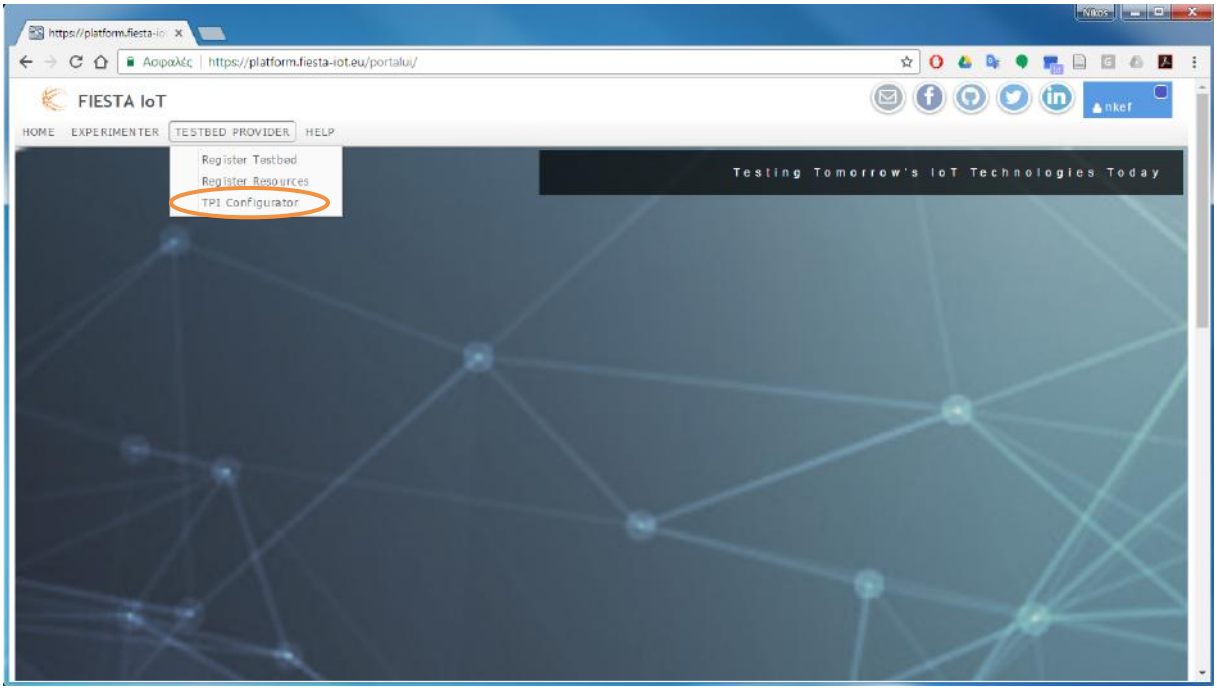

Figure 20 TPI Configurator @ the FIESTA-IoT portal.

As soon as the tool opens it identifies the user and automatically searches for Testbeds bound with the logged-in User ID. The available testbeds are listed at the Testbed dropdown list of the tool's Discovery tab (see Figure 21 below).

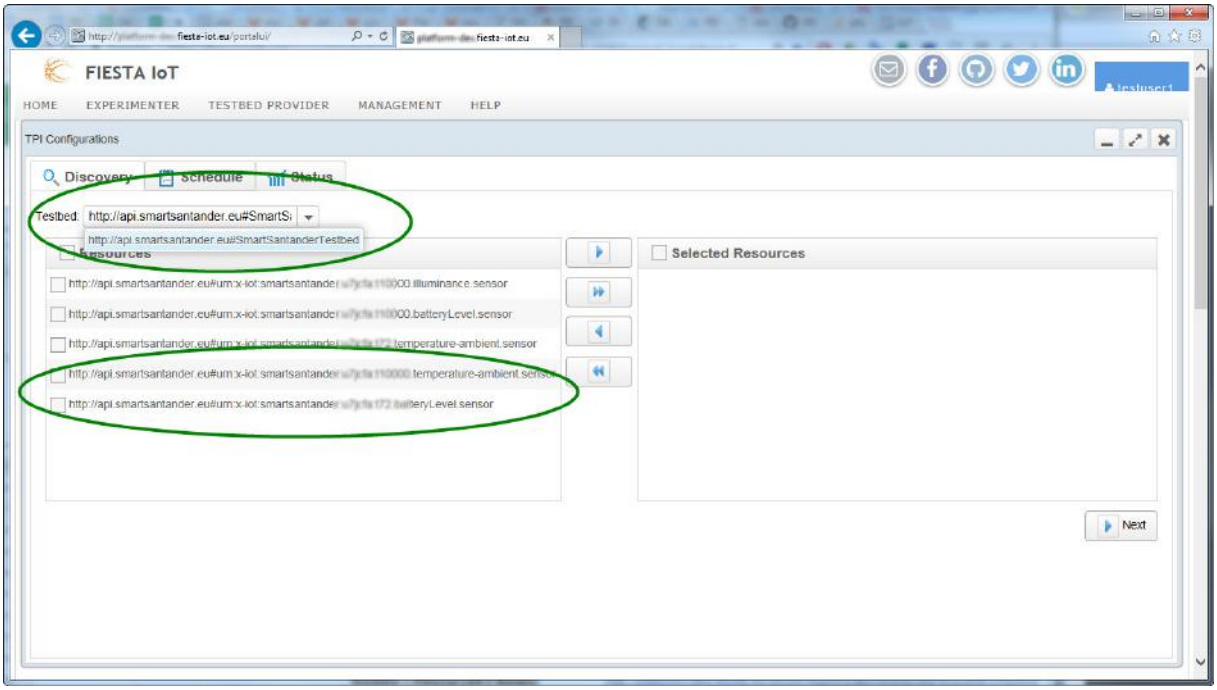

Figure 21 TPI configurator Testbed & Resource Discovery

When a testbed is chosen the registered resources appear at the resource list where the user can select in order to proceed with their data retrieval scheduling (see Figure 21 above and Figure 22 below) by hitting next.

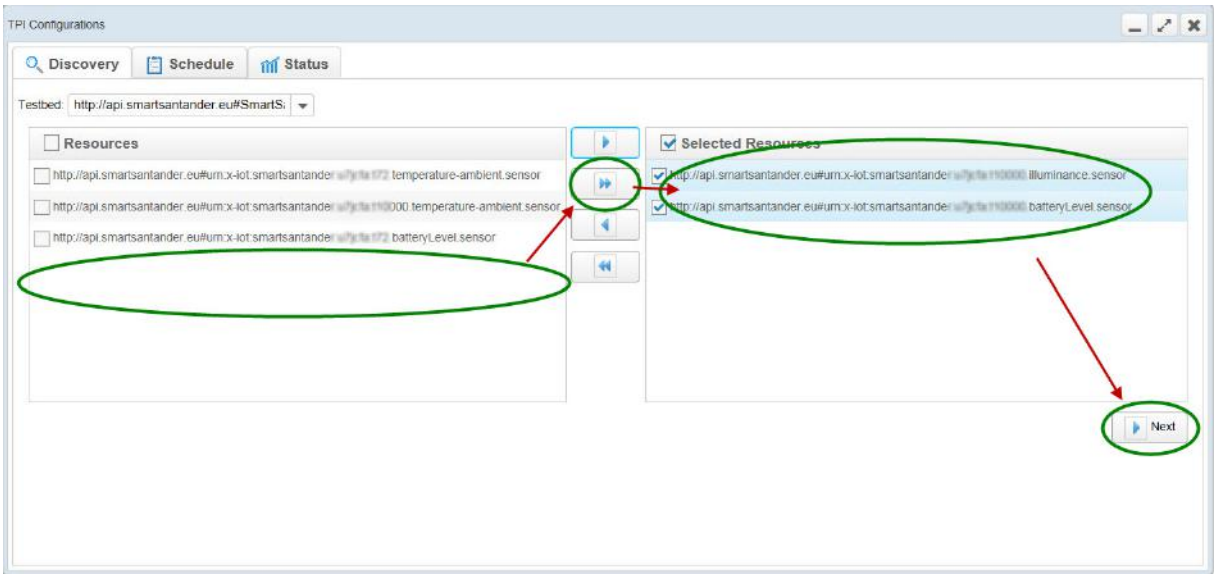

Figure 22 TPI configurator resource selection

The Testbed provider can now define a specific schedule to retrieve their measurements. This is achieved by defining it in the “Schedule” tab as shown in Figure 23 below. In the schedule tab the user can specify the properties of the schedule to be defined and these are:

- a name for the schedule,
- to choose the Testbed ‘s exposed service (TPS) (see Figure 23 below) that has already been defined in the Testbed registration process. This Testbed URI will expose one of the TPS services (i.e. get observation, push observation etc.),

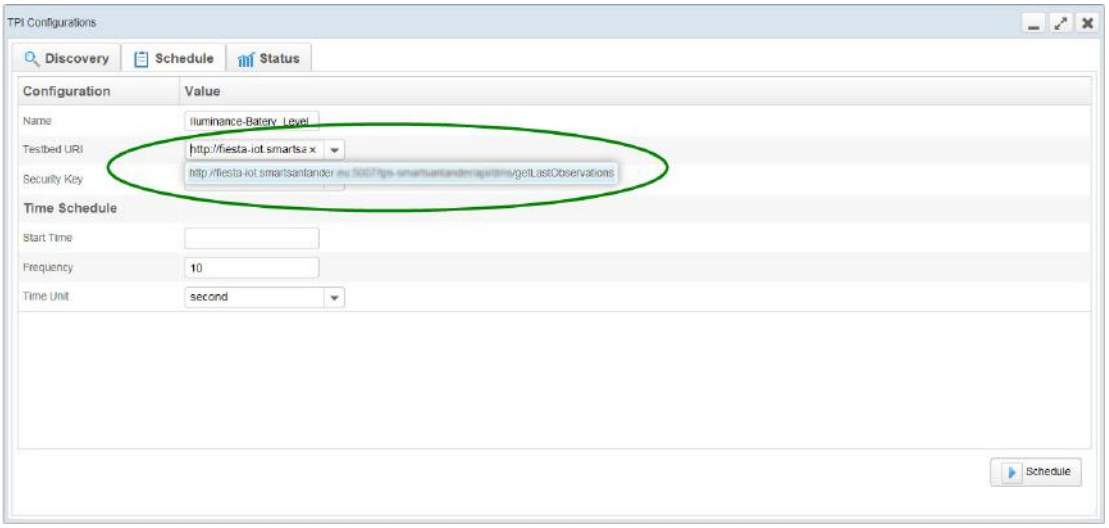

Figure 23 Schedule Tab – Selecting Testbed (TPS) exposed service URI

- Select the Security Key (see Figure 24 below), identified in the Testbed registration process, in order to access the Testbed URI above and

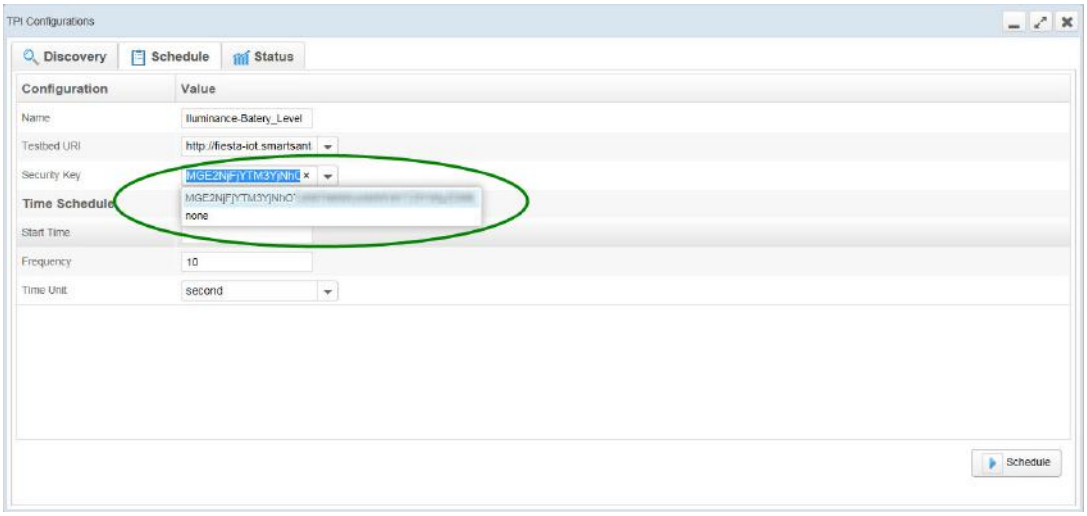

Figure 24 Schedule Tab – Selecting Testbed (TPS) service Security Key

- To specify a timeschedule (if the get methodology for TPS has been implemented) for the daretrieval (get) to be executed from the DMS to TPS component. This is achieved by identifying:

- The start time (optional): which defines a historical date in order to adittionally retrieve historical measurements for a resource (i.e. to retrieve the data from last month set the start time date 30 days in the past). Note that this feature is not curently implemented.
- The frequency and the time unit of it which defines the periodicity frequency of the measurements retrieval (get) from the Testbed's TPS.

After completing the required flow and the definition of the requested properties, the testbed provider needs to hit the schedule button (see Figure 25 below) 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 (depending on the TPS service implemented from the Testbed TPS).

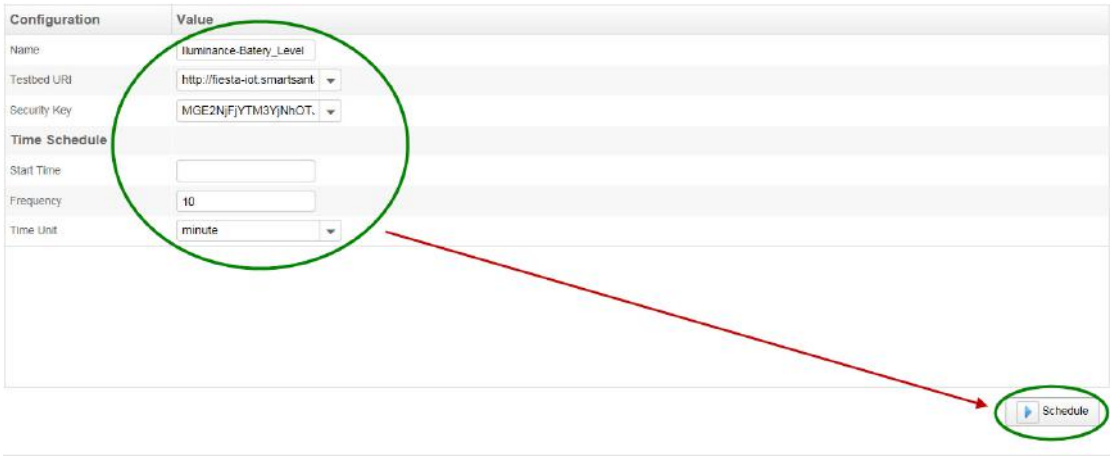

Figure 25 Schedule Tab – schedule button

This step will take the Testbed Provider to the status tab (see Figure 26 below) where an overview of all the scheduled jobs can be found. In case the Experimenter needs to stop a job the job should be picked from the Status list and the stop button should be clicked (see Figure 26 below). This stops the schedule for the specific job of retrieving measurements from TPS (in case the get method has been implemented) or instructs TPS to stop pushing measurements to DMS (in case the push methodology was chosen).

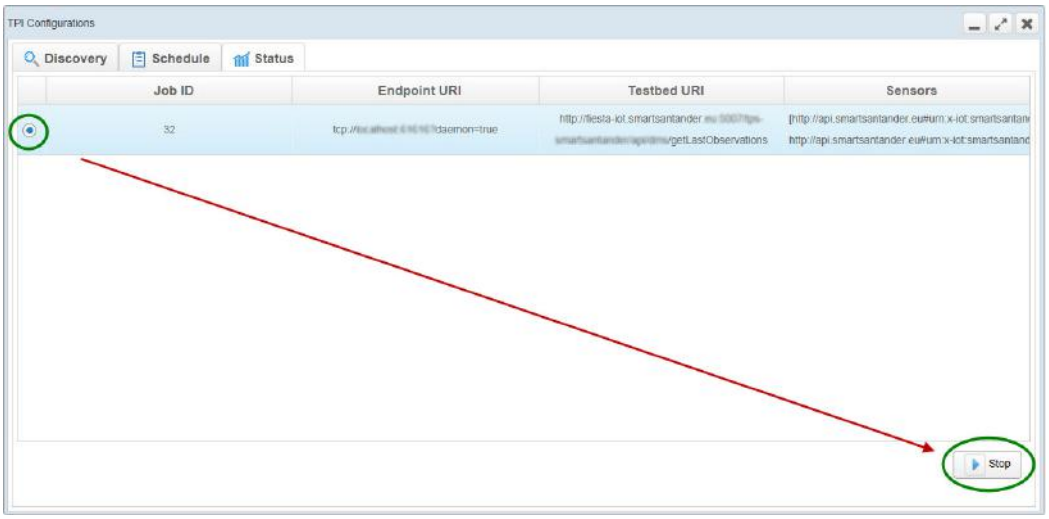

Figure 26 Status Tab – stop a scheduled process

4 TPI AND PREREQUISITE COMPONENTS API SPECIFICATION

4.1 IoT Registry (IR)

4.1.1 IoT Registry Core

4.1.1.1API Definition

Table 1 below summarizes the services supported by the REST API defined over the IR module. Again, for a full description of all the web services, the reader should refer to the API documentation webpage: https://platform.fiesta-iot.wu/iot-registry/docs/api.html. Besides, it is worth highlighting one more time that the use of these web services will be closed to external users and will be only part of the communication between the IR module and other internal components.

Table 1 List of primitives comprising the implemented Semantic Registry API

<<interface>> IrInterface --- GET: /testbeds → Get the list of testbeds POST: /testbeds → Register a testbed (to the triple-store) GET: /testbeds/{id} → Get a testbed description (from the semantic graph) GET: /testbeds/{id}/resources → Get a testbed's underlying resources GET: /resources → Get the list of resources POST: /resources → Register a resource (or a group of them) GET: /resources/{id} → Get a resource description (from the semantic graph) GET: /observations → Get the list of observations POST: /observations → Store an observation GET: /observations/{id} → Get an observation description (from the semantic graph) GET: /queries/execute/{db}/{query} POST: /queries/execute/{db} GET: /queries/store POST: /queries/store GET: /queries/store/{id} PUT: /queries/store/{id} DELETE: /queries/store/{id}

Once we have introduced the list of services available, we describe in Table 2 their operation, highlighting the input (either in the message body for POST/PUT operations or in the own HTTP header in case it is possible).

Table 2 Implemented Semantic Registry API definition

/testbeds		
Method	**Request body/URI param**	**Output**
GET		List of testbeds (RDF)
POST	<String> Testbed IRI	HTTP 201 Created if the process is successful; otherwise, an HTTP 4xx message is returned
/testbeds/{id}		

Method	Request body/URI param	Output
GET	{id} → String (FIESTA node ID)	Semantic information about the testbed (RDF)
/testbeds/{id}/resources		
Method	**Request body/URI param**	**Output**
GET	{id} → String (FIESTA node ID)	List of resources that "hang" from testbed selected through its {id}
/resources		
Method	**Request body/URI param**	**Output**
GET		List of resources (RDF)
POST	<RDF> Resource(s) description(s)	HTTP 201 Created if the process is successful; otherwise, an HTTP 4xx message is returned
/resources/{id}		
Method	**Request body/URI param**	**Output**
GET	{id} → String (FIESTA node ID)	Semantic information about the resource (RDF)
/observations		
Method	**Request body/URI param**	**Output**
GET		List of observations (RDF)
POST	<RDF>Annotated observation	HTTP 201 Created if the process is successful; otherwise, an HTTP 4xx message is returned
/observations/{id}		
Method	**Request body/URI param**	**Output**
GET	{id} → String (FIESTA node ID)	Semantic information about the observation associated to the {id} (RDF)
/queries/execute/{db}/{query}		
Method	**Request body/URI param**	**Output**
GET	{db} → String (resources/observations/global) {query} → SPARQL query	SPARQL output
/queries/execute/{db}		
Method	**Request body/URI param**	**Output**
POST	[HEADER] {db} → String (resources/observations/global) [BODY] → SPARQL query (Content-Type: text/plain)	SPARQL output

`/queries/store`		
Method	**Request body/URI param**	**Output**
GET		List of stored SPARQL queries (defined below)
POST	JSON object to describe a SPARQL (application/json)	HTTP 201 Created if the process is successful; otherwise, an HTTP 4xx message is returned
`/queries/store/{id}`		
Method	**Request body/URI param**	**Output**
GET	[HEADER] {id} → ID of the SPARQL (in the catalogue)	JSON object to describe a SPARQL (application/json)
PUT	[HEADER] {id} → ID of the SPARQL (in the catalogue) [BODY] JSON object to describe a SPARQL (application/json)	HTTP 200 OK if the process is successful; otherwise, HTTP 404 message is returned
DELETE	[HEADER] {id} → ID of the SPARQL (in the catalogue)	HTTP 200 OK if the process is successful; otherwise, HTTP 404 not found if the resource is not found

4.1.1.2Object Definition

First and foremost, all the semantic information (marked as RDF in Table 2) can be serialized in all the formats supported by JENA, defined in the “Content-type” and “Accept” fields of the HTTP header, depending whether the document is an output or input parameter (or both at the same time). Below we find the list of supported RDF formats:

- application/ld+json
- application/n-quads
- application/n-triples
- application/n3
- application/rdf+json
- application/rdf+thrift
- application/rdf+xml
- application/trig
- application/trix
- application/trix+xml
- application/turtle
- application/x-trig
- application/x-turtle
- null/rdf
- text/csv
- text/n-quads
- text/n3

- text/nquads
- text/plain
- text/rdf+n3
- text/trig
- text/turtle

Without leaving these acceptable formats, users can specify (in the “Accept” field of the HTTP header), the serialization type of the SPARQL output between the following list of elements:

- application/sparql-results+json
- application/sparql-results+xml
- application/sparql-results+thrift
- application/json
- application/xml
- text/tab-separated-values
- text/csv
- text/xml
- text/tsv

Finally, we have also specified a JSON object to describe a SPARQL sentence that will be stored into the IR module. As can be inferred from its definition, it basically embraces the raw query to identify it plus a brief description that can others understand what it is actually doing.

```
{
  "value": <String>, /* SPARQL query */
  "name": <String>,
  "description": <String>
}
```

4.1.2 IoT Registry Utilities

Table 3 below introduces the REST API offered as part of the set of utilities included in the IR portfolio. On the other hand, Table 4 digs into the details of each of them. We recall here that the base path to get access to these resources is the following one: https://platform.fiesta-iot.eu/iot-registry/api .

Table 3 IR utilities services (summary)

<<interface>> IrUtilitiesInterface --- POST: /utils/annotator/device POST: /utils/annotator/testbed POST: /utils/annotator/observation GET/POST: /utils/identifier/to_fiesta_iot GET/POST: /utils/annotator/to_testbed

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

Table 4 IR utilities table (complete)

/utils/annotator/device		
Method	Request body/URI	Output
POST	Resource description JSON object (see Section 4.1.2.1.1)	Annotated resource description
/utils/annotator/testbed		
Method	Request body/URI	**Output**
POST	Testbed description JSON object (see Section 4.1.2.1.2)	Annotated resource description connected to a testbed
/utils/annotator/observation		
Method	Request body/URI	Output
POST	Observation JSON object (see Section 4.1.2.1.3)	Annotated observation
/utils/identifier/to_fiesta_iot		
Method	Request body/URI	Output
GET	type=resource/observation/ endpoint/testbed value=Legacy Testbed ID	FIESTA ID
POST	Array of legacy testbeds IDs (see Section)	FIESTA ID
/utils/identifier/to_testbed		
Method	Request body/URI	Output
GET	type=resource/observation/ endpoint/testbed value=FIESTA-IoT ID	Legacy Testbed ID
POST	Array of FIESTA-IoT IDs (see Section)	Legacy Testbed ID

4.1.2.1Object definition

In this Section we proceed to define the JSON objects that are used as input parameters of the IR utilities.

4.1.2.1.1 Resource description JSON object

Table 5 Resource description JSON object


```
{
  "id": <String>,
  "location": {
    "lat": <double>,
    "lon": <double>
  },
  "testbed": <String>,
  "sensing_devices": [
    {
      "id": <String>,
      "qk": <String>,
      "uom": <String>
    },
    {
      "id": <String>,
      "type": <String>,
      "qk": <String>,
      "uom": <String>
    }
  ]
}
```

4.1.2.1.2 Testbed description JSON object

Table 6 Testbed description JSON object.

```
{
  "id": <String>: [
    {
      "id": <String>,
      "location": {
        "lat": <double>,
        "lon": <double>
      },
      "sensing_devices": [
        {
          "id": <String>,
          "qk": <String>,
          "uom": <String>
        },
        {
          "id": <String>,
          "type": <String>,
          "qk": <String>,
          "uom": <String>
        }
      ]
    },
    {
      "id": <String>,
      "location": {
        "lat": <double>,
        "lon": <double>
      },
      "type": <String>,
      "qk": <String>,
      "uom": <String>
    }
  ]
```



```
}
```

4.1.2.1.3 Observation JSON object

Table 7 Observation JSON object

```
{
  "observed_by": <String>,
  "location": {
    "lat": <double>,
    "lon": <double>
  },
  "quantity_kind": <String>,
  "value": <*>, /*Depends on the unit*/
  "format": <String>,
  "unit": <String>,
  "timestamp": <Date>}
```

4.2 TPI Services (TPS)

4.2.1 API Definition

Table 8 below illustrates the main API primitives that support the Testbed Provider Services functionalities, while Table 9 provides more details about each one of the functions that comprise the API.

Table 8 List of primitives comprising the Testbed Provider Services API

<<interface>>
TpiServicesInterface

POST:**getLastObservations** (getLastObservationsPayload:JsonObject):
String annotatedMeasurements,
POST:**getObservations** (getObservationsPayload:JsonObject):
String annotatedMeasurements,
POST:**pushLastObservations** (pushLastObservationsPayload:JsonObject):String
POST:stopPushOfObservations (stopPushOfObservationsPayload: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 9 below:

Table 9 Testbed Provider Services API definition

Service Name	Input	Output	Info
getLastObservations	JsonObject getLastObservations Payload	Response (String annotatedMeasurements, HttpHeaders Content-Type)	This service provides the latest values of a specific Sensor list within an annotated document once. It also provides the annotated document's Content-Type. The acceptable content types are listed below.

getObservations	JsonObject getObservationsPayload	Response (String annotatedMeasurements, HttpHeaders Content-Type)	This service provides the values of a specific Sensor list for a specific time period within an annotated document once. It also provides the annotated document's Content-Type. The acceptable content types are listed below.
pushLastObservations	JsonObject pushLastObservations Payload	Response (String successMessage)	This service pushes continuously the last values of a specific Sensor list to a specific endpoint.
stopPushOfObservations	JsonObject stopPushOfObservations Payload	Response (String successMessage)	This service stops the pushing of the observations for a list of sensors to an endpoint.
pushSingleObservation	String endPointURI, String sensorID	Response (String successMessage)	This service pushes continuously the last value of a specific Sensor to a message bus with the Sensor ID as queue topic.

The accepted content types (Content-Type) mentioned above are:

- application/ld+json
- application/n-quads
- application/n-triples
- application/n3
- application/rdf+json
- application/rdf+thrift
- application/rdf+xml
- application/trig
- application/trix
- application/trix+xml
- application/turtle
- application/x-trig
- application/x-turtle
- null/rdf
- text/csv
- text/n-quads
- text/n3
- text/nquads
- text/plain
- text/rdf+n3
- text/trig
- text/turtle

4.2.2 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:

Table 10 GetLastObservations Payload object

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

Note: the Date has ISO 8601 date and time format ("yyyy-MM-ddThh:mm:ss") in UTC and the start date precedes the stop date in time. If the startDate and the stopDate appear in other format, they have to be converted to the above one.

Definition of pushLastObservationsPayload object:

Table 11 PushLastObservations Payload object

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

Definition of stopPushOfObservations object:

Table 12 StopPushOfObservations object

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

4.3 Data Management Services (DMS)

4.3.1 API Definition

Table 13 below illustrates the main API primitives that support the TPI Management functionalities, while Table 14 provides more details about each one of the functions that comprise the API.

Table 13 List of primitives comprising the TPI Data Management Services API

<<interface>> TpiDataManagementServicesInterface --- POST: **unsubscribeFromObservationStream** (stopPushOfObservationsPayload:JsonObject): Response POST:**subscribeToObservations** (subscribeToObservationsPayload:JsonObject): Response POST:**subscribeToObservationStream** (subscribeToObservationStreamPayload:JsonObject): Response POST:**subscribeToObservationStreamWithTopic** (subscribeToObservationStreamWithTopicPayload:JsonObject): Response POST:**unsubscribeFromObservation** (unsubscribeFromObservationPayload:JsonObject): Response Post: **pushObservationsStreamProxy**(hh: HttpHeaders, annotatedMeasurementsDocument:String): Response

A FIESTA-IoT implementation SHALL implement methods of the TPI Data Management Services API as specified in Table 14 below:

Table 14 TPI Data Management Services API definition

Service Name	Input	Output	Info
unsubscribeFromObservationStream	JsonObject unsubscribeFromObservationStreamPayload	Response (String successMessage)	This service stops the Observations pushing.
subscribeToObservations	JsonObject subscribeToObservationsPayload	Response (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	Response (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	Response (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	Response (String successMessage)	This service unregisters a list of Sensors from an endpoint.
pushObservationsStreamProxy	HttpHeaders hh, String annotatedMeasurementsDocument	Response (String successMessage)	This service provides an Endpoint in order the Testbeds to push their annotated measurements (annotatedMeasurementsDocument) to the FIESTA-IoT message bus. It also requires as input the “Content-Type” of

			the document and it retrieves it from the Http header. The acceptable content types are listed below.

The accepted content types (Content-Type) mentioned above are:

- application/ld+json
- application/n-quads
- application/n-triples
- application/n3
- application/rdf+json
- application/rdf+thrift
- application/rdf+xml
- application/trig
- application/trix
- application/trix+xml
- application/turtle
- application/x-trig
- application/x-turtle
- null/rdf
- text/csv
- text/n-quads
- text/n3
- text/nquads
- text/plain
- text/rdf+n3
- text/trig
- text/turtle

4.3.2 Object Definition

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

Definition of subscribeToObservationsPayload object:

Table 15 SubscribeToObservations Payload object

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


Note:

1. the Date has ISO 8601 date and time format ("yyyy-MM-ddThh:mm:ss") in UTC
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 and month. It represents the time step, e.g. 5 seconds
5. The startDate can also be empty. If it is not defined then an immediate call to the testbedURI is performed to get the obsevations and the startDate is equal to the time at that point. The next call is then scheduled for startDate + frequency (in timeUnit counted).

Definition of unsubscribeFromObservationStreamPayload and unsubscribeFromObservationPayload objects:

Table 16 unsubscribeFromObservation(Stream) Payload object

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

Definition of subscribeToObservationStreamPayload object:

Table 17 subscribeToObservationStreamPayload Object

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

Definition of subscribeToObservationStreamWithTopicPayload object:

Table 18 subscribeToObservationStreamWithTopicPayload Object

```
{
    "sensorIDs": [<String> sensorIDs],
    "testbedURI": <String> theTestbedURI,
    "topicName": <String> theTopicName
}
```

4.4 Testbed & Resource registration (TRR)

4.4.1 API Definition

Table 19 below illustrates the main API primitives that support the Testbed Provider Registration Services functionalities, while Table 14 provides more details about each one of the functions that comprise the API.

Table 19 List of primitives comprising Testbed and Resource Registration API

<<interface>>
RegisterTestbedsResource

POST **createRegisterTestbeds**(registerTestbeds:JsonObject)

<<interface>>

TestbedResource

GET **getAllTestbeds**(page:Integer, size:Integer):List<JsonObject>
GET **getRegisterTestbeds**(id:Long):JsonObject
POST **getAllTestbedsByUserID**(testbedRegisterInputDTO:JsonObject):JsonObject
POST **getAllTestbedsRegisterByRegisterIDList**(testbedRegisterInputDTO:JsonObject):List<JsonObject>
POST **getAllTestbedsRegisterIDsByUserID**(testbedRegisterInputDTO:JsonObject):List<JsonObject>
POST **getTestbedByIRI**(testbedRegisterInputDTO:JsonObject):JsonObject
POST **getTestbedByIRIAndName**(testbedRegisterInputDTO:JsonObject):JsonObject
POST **findTestbedByRegisterID**(testbedRegisterInputDTO:JsonObject):JsonObject

<<interface>>

RegisterDevicesResource

POST **createRegisterDevices**(registerDevices:JsonObject):JsonObject
POST **createRegisterDevicesByText**(registerTestbdResourceWithText:JsonObject):JsonObject
POST **createRegisterDevicesByUpload**(registerTestbdResourceWithUpload:JsonObject):JsonObject

A FIESTA-IoT implementation SHALL implement methods of the Testbed Provider Services API as specified in Table 20 below:

Table 20 TPI Data Management Services API definition

Service Name	Input	Output	Info
createRegisterTestbeds	JsonObject registerTestbeds	JsonObject	This service provides the way register testbed
getAllTestbeds	page integer, size integer	List <JsonObject>	This service getting all register testbeds with paging default page:0, size:20
getRegisterTestbeds	id Long	JsonObject	This service get testbed by id
getAllTestbedsByUserID	JsonObject testbedRegisterInputDTO	List <JsonObject>	This service getting all testbeds by userID
getAllTestbedsRegisterByRegisterIDList	JsonObject testbedRegisterInputDTO	List<JsonObject>	This service get all testbeds by registerID list
getAllTestbedsRegisterIDsByUserID	JsonObject testbedRegisterInputD	List <JsonObject>	This service get all testbeds by userID

	TO		
getTestbedByIRIAndName	JsonObject testbedRegisterInputDTO	JsonObject	This service get testbed by iri and name
findTestbedByRegisterID	JsonObject testbedRegisterInputDTO	JsonObject	This service get testbed by registerID
createRegisterDevices	JsonObject registerDevices	JsonObject	This service register devices manual
createRegisterDevicesByText	JsonObject registerTestbdResourceWithText	JsonObject	This service register devices by text
createRegisterDevicesByUpload	JsonObject registerTestbdResourceWithUpload	JsonObject	This service register devices by upload

4.4.2 Object Definition

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

Definition of registerTestbeds object:

Table 21 registerTestbeds object

```
{

  "annotatedObservation": <String> annotatedObservation,
  "annotatedResourceDescription": <String> annotatedResourceDescription,
  "getApiKey": <String> getApiKey,
  "getLastObservationsURL": <String> getLastObservationsURL,
  "getObservationsURL": <String>getObservationsURL,
  "iri":<String> iri,
  "latitude": <String> latitude,
  "longitude":<String> longitude,
  "name":<String> name,
  "pushApiKey":<String> pushApiKey,
  "pushLastObservationsURL":<String> pushLastObservationsURL,
  "pushObservationsURL":<String> pushObservationsURL,
  "resourceDescriptionContentType":<String> resourceDescriptionContentType,
  "resourceObservationContentType": <String> resourceObservationContentType,
  "resourceType":<String> resourceType
}
```

Definition of testbedRegisterInputDTO object:

Table 22 testbedRegisterInputDTO object

```
{
  "expectedFieldsAsResult": [
    <String> fieldName
  ],
  "iri": <String> iri,
  "name": <String> name,
  "registerID": <String> registerID,
  "registerIDList": [
    <String> registerID
  ],
  "userID": <String> userID
}
```

Definition of registerDevices object:

Table 23 registerDevices object

```
{
   "devices": [
    {
      "id": <String> id,
      "lat": <float> lat,
      "lon": <float> lon,
      "qk": <String> qk,
      "uom": <String> uom
    }
  ],
  "registerTestbeds": {
    "id": <Long> id,
    "iri": <String> iri,
    }
}
```

Definition of registerTestbdResourceWithText object:

Table 24 registerTestbdResourceWithText object

```
{
  "annotatedResourceDescription": <String> annotatedResourceDescription,
  "contentType":  <String> contentType

}
```

Definition of registerTestbdResourceWithUpload object:

Table 25 registerTestbdResourceWithUpload object

```
{
  "contentType": <String> contentType,
  "uploadContent": [
    <String> uploadContentInBase64Encode
  ]

}
```


5 BACKGROUND IMPLEMENTATION TECHNOLOGIES

In this section we list various technologies that have been used for some of the different component implementations. We mainly list the technologies for processing engines and message bus usage. Having in mind the goals of Fiesta-IoT and the advantages of the required tools we adopted the following technologies: we chose ActiveMQ as our service bus and JMS as messaging technology of choice. ActiveMQ is ideal 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. In the following sections we provide a short description of the mentioned technologies.

5.1 Message Bus

The **Apache ActiveMQ**[4] is an open-source[5] 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 (Enterprise Service Bus) 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 client or server. ActiveMQ is ideal for high performance clustering, machine-to-machine (M2M) and peer-based communication.

The **Java Message Service (JMS)**[6] [7] API is a Java Message Oriented Middleware API for sending messages between two or more clients. It is an implementation to handle the Producer-consumer problem. JMS is a part of the Java Platform, Enterprise Edition, and is defined by a specification developed under the Java Community Process as JSR 914. It is a messaging standard that allows application components based on the Java Enterprise Edition (Java EE) to create, send, receive, and read messages. It allows the communication between different components of a distributed application to be loosely coupled, reliable, and asynchronous. JMS supports point-to-point and publish/subscribe models. In the point-to-point messaging

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

[5] Released under the Apache 2.0 license

[6] http://docs.oracle.com/javaee/6/tutorial/doc/bnceh.html

system, messages are routed to an individual consumer that maintains a queue of "incoming" messages. This messaging type is built on the concept of message queues, senders, and receivers. Each message is addressed to a specific queue, and the receiving clients extract messages from the queues established to hold their messages. Publish/subscribe enables publishing messages to a particular message topic. Subscribers may register interest in receiving messages on a particular message topic. In this model, neither the publisher nor the subscriber knows about each other.

5.2 Process Engine/Scheduler

The **Quartz Job Scheduler**[7] is an open source[5] 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 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.

6 TPI PROTOTYPE IMPLEMENTATION

In this section we provide an installation and basic usage manual of the prototype implementation of the above specified components and APIs. At this point it worth to mention that although the API specifications and implementations are in a mature state as the project evolves the implementations and APIs will evolve as well. This is why FIESTA-IoT consortium has authored a Handbook [8] that is a “living document” and will get updated as the platform evolves. This Handbook is a public document and offered to the FIESTA-IoT platform users.

6.1 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-

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

iot.eu/platform/core/. The latest version of the components are under the "develop branch"[8] .

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: provides the collection of all of the components related with the testbed provider interface
    - tpi.api.tps : Testbed provider services API sceleton
    - tpi.api.dms: Testbed provider interface data management tps.instance: includes the FIESTA-IoT internal TPS iplementations
- ui: provides all the modules related to the User Interface
  - ui.tpi.configurator: Configuration UI for TPS-DMS interaction.
  - ui.testbed.registry: The testbed registration user interface.
  - ui.resource.registry: The resource registration user interface.
- messagebus: provides all the modules related to the FIESTA-IoT message bus
  - messagebus.dispacher: the component that subscribes for the annotated measurements to the message bus to push them to the IoT-Registry.
- utils: provides utilities related with the platform
  - utils.commons: include the common utils/objects used in more than 2 projects/components

Additionally to the FIESTA-IoT GitLab, which is private, we are offering some components (i.e. TPS) thru GitHub[9] in order to enable external users to utilize. The components that are also offered thru GitHub are mentioned explicitly in each of their section below.

6.2 Common Information

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

6.2.1 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.

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

[9] https://github.com/fiesta-iot

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.

6.2.2 Install & Run

The prototype 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[10] 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.

6.3 Components

6.3.1 IoT-Registry (IR)

The produced component (from the build process above) is the **iot-registry.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]/iot-registry

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

[10] http://wildfly.org/

6.3.1.1 Containers and Libraries

The Table 26 below 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.

Table 26 Containers and libraries used for the prototype implementation

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

6.3.2 DMS

6.3.2.1 System Requirements

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

6.3.2.2 Containers and Libraries

Table 27 below 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.

Table 27 Containers and libraries used for the prototype implementation

Container / Library / Framework	**Version**
WildFly	10.0.0.Final
Java Platform, Standard Edition	1.8.0_25
Maven	3.1.1
Activemq	5.13.2
Resteasy	3.0.16.Final
fiesta-commons	0.0.1

logback	1.0.11
javamelody	1.45.0
quartz-scheduler	2.2.1
mysql-connector	5.1.39

6.3.3 TPS

A TPS instance may be implemented in any language and or environment the Testbed provider chooses to as long as it follows the above API as a REST web service. Should a Testbed provider choose to implement it in Java we provide a skeleton Java project in order to help bootstrapping the process. Information where this project can be found and to be used is provided in the following sections.

6.3.3.1 Code Availability and Structure

The Testbed Provider Interface components are offered at FIESTA-IoT GitHub repository[11]. The latest version of the components are under the "develop branch".

The project is divided into two main packages. The "*eu.fiestaiot.tpi.api.tps.rest*" and "*eu.fiestaiot.tpi.api.tps.impl.dataservices*":

1. **eu.fiestaiot.tpi.api.tps.rest**

Within this package, the "TpiApiTestbedProviderServices.java"[12] class can be found which is the entry point for the Services described at the TPS API above. The two main methods of interest are:

- public Response getLastObservations(String getLastObservationsPayload)
- public Response getObservations(String getObservationsPayload)

These methods analyse the payload and forwards it to the implementation of them, which resides in the following package.

2. **eu.fiestaiot.tpi.api.tps.impl.dataservices**

In this package, we have a list of the implementations classes of the different web services exposed from the TPS instance. The Testbed provider needs to implement at least one of them (get or push). The four classes are:

- GetLastObservationsImpl.java[13]
- GetObservationsImpl.java[14]
- PushLastObservationsImpl.java[15]
- StopPushOfObservationsImpl.java[16]

[11] https://github.com/fiesta-iot/testbed.tpi/tree/develop

[12] /tpi.api.tps/src/main/java/eu/fiestaiot/tpi/api/tps/rest/TpiApiTestbedProviderServices.java

[13] /tpi.api.tps/src/main/java/eu/fiestaiot/tpi/api/tps/impl/dataservices/GetLastObservationsImpl.java

[14] /tpi.api.tps/src/main/java/eu/fiestaiot/tpi/api/tps/impl/dataservices/GetObservationsImpl.java

[15] /tpi.api.tps/src/main/java/eu/fiestaiot/tpi/api/tps/impl/dataservices/PushLastObservationsImpl.java

6.3.3.2 System Requirements

The TPS component described above is deployed within a WildFly container and is using Maven as project management.

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

6.3.3.3 Install & Run

The prototype 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 **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.

The produced (from the build process above) project is the **tpi.api.tps.war** file. 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[17] 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 URLs:

- **http://[HOST]:[PORT]/tpi.api.tps/rest/tps/getLastObservations**
- **http://[HOST]:[PORT]/tpi.api.tps/rest/tps/getObservations**
- **http://[HOST]:[PORT]/tpi.api.tps/rest/tps/pushLastObservations**
- **http://[HOST]:[PORT]/tpi.api.tps/rest/tps/stopPushOfObservations**

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

[16] /tpi.api.tps/src/main/java/eu/fiestaiot/tpi/api/tps/impl/dataservices/StopPushOfObservationsImpl.java

[17] http://wildfly.org/

6.3.3.4 Containers and Libraries

Table 28 below 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.

Table 28 Containers and libraries used for the prototype implementation

Container / Library / Framework	**Version**
WildFly	10.0.0.Final
Java Platform, Standard Edition	1.8.0_25
Maven	3.1.1
Resteasy	3.0.16.Final
logback	1.0.11

6.3.4 Message Bus Dispatcher

6.3.4.1 System Requirements

The produced (from the build process above) project name is the **messagebus.dispatcher.war** file.

6.3.4.2 Containers and Libraries

Table 29 below 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.

Table 29 Containers and libraries used for the prototype implementation

Container / Library / Framework	**Version**
WildFly	10.0.0.Final
Java Platform, Standard Edition	1.8.0_25
Maven	3.1.1
Activemq	5.13.2
Resteasy	3.0.16.Final
fiesta-commons	0.0.1
logback	1.0.11
javamelody	1.45.0

7 CONCLUSIONS

This deliverable has illustrated the specification and implementation of the FIESTA-IoT TPI and the required components, 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.

A main conclusion from the TPI specification process is that such interfaces specify two sets of functionalities: i) functionalities targeting the configuration and management of the testbeds, and ii) 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 is well documented and offered to the participants of the open call processes, given that these participants will have to implement the TPS 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.

8 REFERENCES

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

[2] FIESTA-IoT, “Deliverable D3.3 (D3.2.1): Specification and implementation of common Testbed interfaces”, 2016

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

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

[5] FIESTA-IoT, “Deliverable 3.2: Semantic models for testbeds, interoperability and mobility support and best practices”, 2016

[6] FIESTA-IoT, “Deliverable 4.5: Tools and Techniques for Managing Interoperable Data sets”, 2017

[7] Wikipedia, “Java Messaging Service”, available at https://en.wikipedia.org/wiki/Java_Message_Service

[8] FIESTA-IoT, “Handbook for Experimenters and Extensions”, Release 1, 2017

FIESTA-IoT 2017

