HORIZON 2020
The EU Framework Programme for Research and Innovation

FIESTA-IoT
www.fiesta-iot.eu

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

D4.2 EaaS Model Specification and Implementation V2

Document Id:	FIESTAIoT-WP4-D4.2-EaaS_Model_Specification_and_Implementation-300617-Draft
File Name:	FIESTAIoT-WP4-D4.2-EaaS_Model_Specification_and_Implementation-300617-Draft.doc
Document reference:	Deliverable 4.2
Version:	Draft
Editor:	Aqeel Kazmi, Martin Serrano
Organisation:	NUIG-DERI, Insight Center for Data Analytics
Date:	30 / 06 / 2017
Document type:	Deliverable
Dissemination level:	PU

DOCUMENT HISTORY

Rev.	Author(s)	Organisation(s)	Date	Comments
V01	Aqeel Kazmi, Hung Nguyen	NUIG	2017/04/20	Initial version with ToC
V02	Ramnath Teja	KETI	2017/05/09	Updates to section 6
V03	Aqeel Kazmi	NUIG	2017/05/10	Updated ToC as per WP4 partner suggestions
	Luis Sanchez, Jorge Lanza, David Gomez	UC	2017/05/16	Contribution to Section 3
	Nikos Kefalakis	AIT	2017/05/16	Added Domain Specific Language (FEDSpec) and Experiment Registry Management sections
	Rachit Agarwal	INRIA	2017/05/17	Defining an Experiment thru DSL, FAT Integration with FIESTA-IoT.
V07	Aqeel Kazmi	NUIG	2017/05/17	Merged various versions.
	Luis Sanchez, Jorge Lanza, David Gomez	UC	2017/05/17	Contribution to Section 3
	Tarek Elsaleh/Alireza Ahrabian	UNIS	2017/05/19	Living Lab workflow. Update of FAT JSON input.
	Ramnath Teja	KETI	2017/05/22	Updates to section 6
V08	Aqeel Kazmi	NUIG	2017/05/25	Synched all versions. Fixed references, Added content to section 1, 2, and 8 Ready for QA and TR
	Tiago Teixeira	Unparallel	2017/06/20	TR
	Mengxuan Zhao	EGM	2017/06/23	QR
	Ronald Steinke	FOKUS	2017/06/27	TR
V09	Aqeel Kazmi	NUIG	2017/06/27	TR & QR comments addressed in general & section 1, 2, and 8
	Rachit Agarwal	INRIA	2017/06/27	Updates in Section 4.3
	Luis Sanchez	UC	2017/06/28	Updates in Section 3
	Ramnath Teja	KETI	2017/06/29	Updates in Section 6
	Nikos Kefalakis	AIT	2017/06/29	Updates in Section 4 & 5
V10	Aqeel Kazmi	NUIG	2017/06/30	Synched all V09 documents, Re-arranged references, Preparing draft for EC submission
Draft	Aqeel Kazmi	NUIG	2017/06/30	Accepted all changes, Comments removed, draft ready for submission

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TERMS AND ACRONYMS

API	Application Programmable Interface
DSL	Domain Specific Language
EaaS	Experiment as a Service
EEE	Experiment Execution Engine
ERM	Experiment Registry Management
FAT	FIESTA-IoT Analytics Toolkit
FEDSpec	FIESTA-IoT Experiment Model Object
FEMO	FIESTA-IoT Experiment Model Objects
FIRE	Future Internet Research & Experimentation
FISMO	FIESTA-IoT Service Model Object
IoT	Internet of Things
RB	Resource Broker
RM	Resource Manager
TDB	Triplestore DataBase

1 INTRODUCTION

1.1 Executive Summary

In FIESTA-IoT, Experimentation as a Service (EaaS) model refers to a new and innovative way of allowing researchers and experimenters to define and execute data-intensive experiments on top of heterogeneous set of IoT testbeds. This is enabled by aggregating data streams and services from multiple IoT platforms (or testbeds) using a common data model namely FIESTA-IoT ontology. Having ensured the interoperability among multiple testbeds, the data and services are made available only in a testbed-agnostic manner which allows the experimenters to conduct testbed-agnostic experiments. In FIESTA-IoT we interpret the EaaS model as assisting experimenters and developers in designing interoperable IoT applications, more precisely, by semantically annotating data, deducing meaningful knowledge from sensor data, combining applicative domains to build smarter IoT applications and experiments.

This deliverable is dedicated to the specification and implementation of the EaaS model, as required in order to facilitate the researchers, experimenters, and developers to define and execute IoT testbed-agnostic experiments. The EaaS specification is driven by a number of requirements, including the need for specifying the modelling language for modelling experiments associated with IoT data and resources along with tools for specifying, parsing and enacting this language, the experiment workflow that describes the process followed by an experimenter to create and execute an experiment, and the Meta-Cloud framework methodology. The review of complementary techniques and concepts, such as interoperability of services, Domain Specific Languages (DSL) and data workflows, that led to the EaaS model specification have been included in the first version of the deliverable [7] and therefore not in this document in order to avoid repetition.

In order to support the EaaS innovative concept in FIESTA-IoT, the Meta-Cloud architecture and its implementation have been carried out. The use of the Meta-Cloud is explained through the experiment and data workflows. The Meta-Cloud is employing the EaaS model, a cornerstone component which has been designed and implemented as a Domain Specific Language (DSL). This implementation is called FIESTA-IoT Experiment Description Specification (FEDSpec) and is capable of hosting all the defined experiments of a specific experimenter. The Experiment Registry Management (ERM) tool (UI & API) has been designed and implemented that allows experimenters to easily manage and interact with FEDSpec. An API has been designed to deal and easily interact with the EaaS Model. The integration workflow of the FIESTA-IoT Analytics Toolkit (FAT) and Experiment Execution Engine (EEE) is specified. In addition, a concrete use case namely the LivingLab experiment is implemented. Additionally, this deliverable presents and discusses the work carried out in order to support Node-RED for automatically generating experiment FEDSpec. Node-RED allows experiment modelling by using its visual programming nature.

This deliverable satisfies and fulfils a number of both functional and non-functional requirements specified in deliverable 2.1 [2]. In particular, it meets the requirements related to discovery of IoT resources and observations, ways to retrieve sensor streams, integration of multifarious IoT resources, and specification of experiments.

1.2 Audience

This document 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 the FIESTA-IoT Platform.
- **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.
- **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 and work with the FIESTA-IoT platform.
- **Standardization bodies** will have access to this deliverable as it will be a public document and therefore the specifications and tools developed can be standardized following the involvement and reach a wider adoption.

1.3 Structure

In addition to the introductory section, the deliverable is structured as follows:

- **Section 2** provides an overview of the EaaS model. It then provides the placement of the EaaS model within the FIESTA-IoT architecture.
- **Section 3** provides the implementation details of core components of the FIESTA-IoT Meta-Cloud a.k.a IoT-Registry, which is the core of the FIESTA-IoT platform.
- **Section 4** describes the FIESTA-IoT experiment management that is facilitated by the usage of a Domain Specific language (DSL) which is capable of hosting all the experiments of a specific user.
- **Section 5** focuses on the Experiment Registry Management (ERM) component that facilitates the experiment storage, retrieval and discovery.
- **Section 6** provides details on automatically generating the experiment DSL using Node-RED tool.
- **Section 7** focuses on the integration of FIESTA-IoT Analytics Toolkit (FAT) component in FIESTA-IoT platform. In addition, it describes the Living Lab Experiment.
- **Section 8** finally concludes this deliverable.

1.4 Updates from the previous version

This deliverable is the second and final iteration of Deliverable D4.1 [7]. It reflects the updates to the EaaS model and related tools with respect to D4.1. These updates are summarized below:

- Introductory sections regarding FIESTA-IoT scope and WP4 overview have been removed in order not to repeat the content.
- The section regarding background technologies and related work has been removed since there were no updates.
- New section on EaaS model placement and description is added to set the context of work presented in this deliverable.
- FIESTA-IoT Meta-Cloud section has been updated with new content that describes the architecture and implementation of the internals of IoT Registry.
- EaaS Model section has been updated with details over the latest version of FEDSpec.
- Experiment Registry Management section is updated and details are provided about the latest functionalities that have been implemented since previous version of the deliverable.
- Section 6 contains updates over the automatic generation of DSL using the Node-RED tool.
- Experiment workflow section contains details about FIESTA-IoT Analytics Toolkit integration with EEE. In addition, the LivingLab experiment is presented and discussed.
- Finally, the conclusion section is updated in order to reflect the changes made in this deliverable.

2 EAAS MODEL PLACEMENT WITHIN FIESTA-IOT

The purpose of the EaaS model is to facilitate testbed-agnostic access to data streams and services. The EaaS model consists of a number of concepts (tools) developed within the FIESTA-IoT infrastructure. The core elements include the FIESTA-IoT Meta-Cloud (IoT-Registry), experiment modelling language (FEDSpec), and the Experiment Registry Management (ERM). In addition, it includes the specification of experiment workflow and Node-RED integration for automatically generating experiment DSLs.

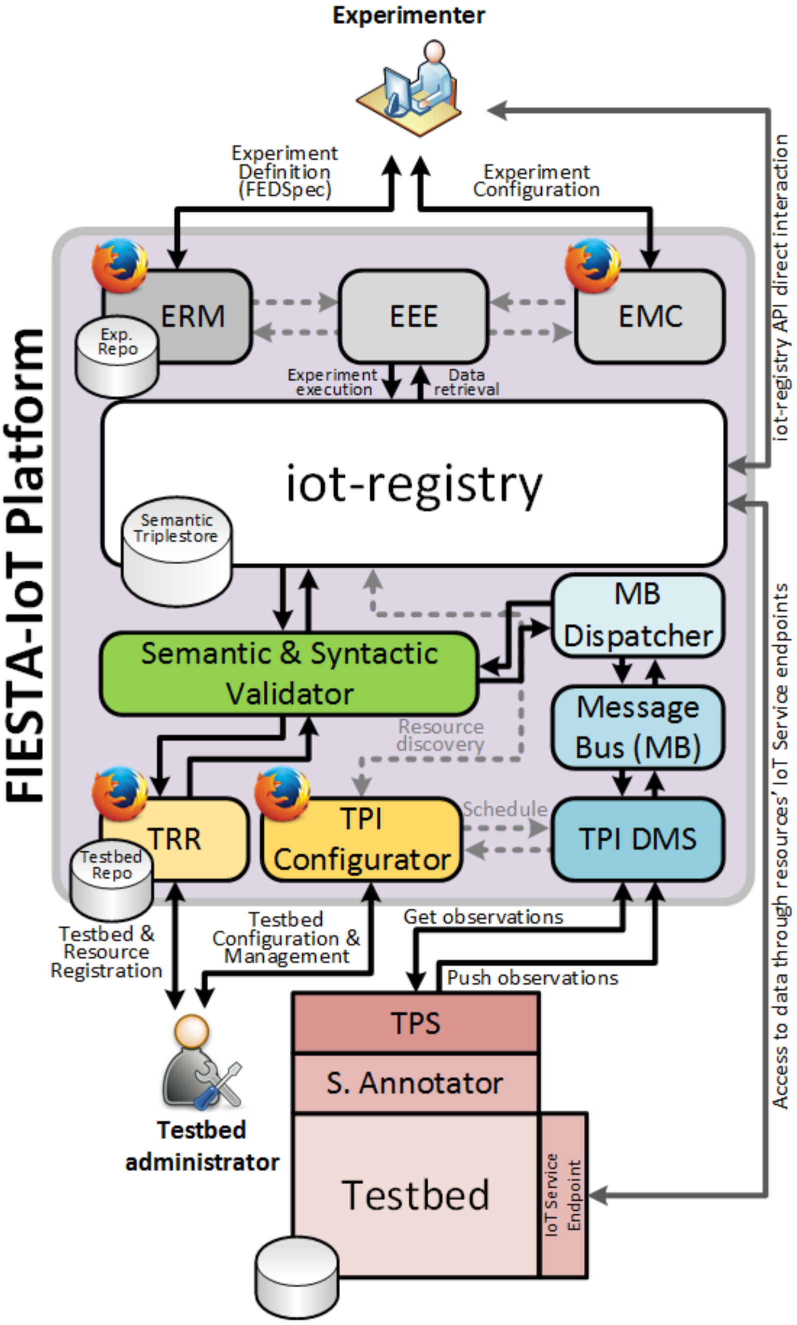


Figure 1. FIESTA-IoT Platform Architectural Overview

The IoT Registry (FEISTA-IoT Meta-Cloud), as this can be seen in Figure 1 above, is the main component within the FEISTA-IoT infrastructure. The main function of IoT registry is to store and maintain all the information e.g. the resource descriptions and observations provided by the integrated testbeds. This information is accessible by the experimenter directly via the API that IoT registry has exposed or through creating and executing an experiment. The ideal workflow is that the experimenter defines an experiment using the FEDSpec model then registers and stores this experiment using the Experiment Registry Management (ERM) tool. The Experiment Management Console (EMC) component is used by an experimenter to see details of an experiment and change the status e.g. start, stop or schedule. The Experiment Execution Engine (EEE) executes the experiment as per its schedule and parameters specified in FEDSpec. Finally, the results are returned (to a specified location) after the successful execution of an experiment. This whole process (experiment workflow) is provided and discussed in section 7.1.

3 FIESTA-IOT META-CLOUD

In the previous version of the deliverable [7], the abstract methodology and architecture to implement the FIESTA-IoT Meta-Cloud was described. In this version, the implementation of the core components identified will be described. Indeed, the actual implementation of these components is the core of the FIESTA-IoT platform that is currently running and supporting the experimentation and integration of testbeds that is ongoing nowadays.

Among the identified functional components, the following ones have been implemented within a single software development so called *IoT-Registry*:

- **Query engine** enables to query a subset of data that we are interested in. A SPARQL query engine is used, the one available within the Jena framework. It corresponds to the Querying step from SEG 3.0 [7].
- **Meta-cloud data endpoint** will provide access to semantically annotated data (e.g., SPARQL endpoint, web services).
- **Registry** enables registering a new resource, testbed or service. It is included within the Service step from SEG 3.0.
- **Discovery** enables discovering already registered resources, testbeds or services. It is included within the Service step from SEG 3.0.
- **Repositories**. They are the actual placeholders where the semantic triples representing the resources and observations are stored.

In the remaining of the chapter we provide a detailed overview of the internals of the *IoT-Registry* which complements the description of the services and APIs that it exports, which are thoroughly described in [9] and [1].

3.1 IoT-Registry architecture

The *IoT-Registry*'s main function is to keep and maintain all the (semantic) resource descriptions and observations which the underlying testbeds, federated within FIESTA-IoT, have provided. On top of this "collector" behaviour, we have built a fully-fledged REST API that allows the interplay between users (FIESTA-IoT admins, testbed providers, experimenters or observers) and the databases that store the information. Below we describe every single feature carried out by this key module. Figure 2 below shows the internal architecture of the implemented IoT-Registry.

The core of the component is formed by the Triplestore Database (TDB) that provides the storage capacity for aggregating the Resource Descriptions from the devices belonging to the federated testbeds as well as the Observations that these devices are constantly producing. Both the Resource Descriptions and the Observations are semantic documents that uses RDF serialization to describe them. In this sense, the TDB implements the information model that is specified by the FIESTA-IoT ontology [4], [5], and [13].

However, this is only the repository part of the component and the actual intelligence of the module is implemented on the other sub-systems. The first of these functional modules is the Data Endpoint which is the responsible for exporting the SPARQL endpoint of the TDB into a web-based API. It, thus, mainly acts as a proxy getting the

SPARQL queries that are carried into the body of the HTTP requests, injecting them into the native SPARQL endpoint of the TDB and getting back the corresponding response within the HTTP response packet.

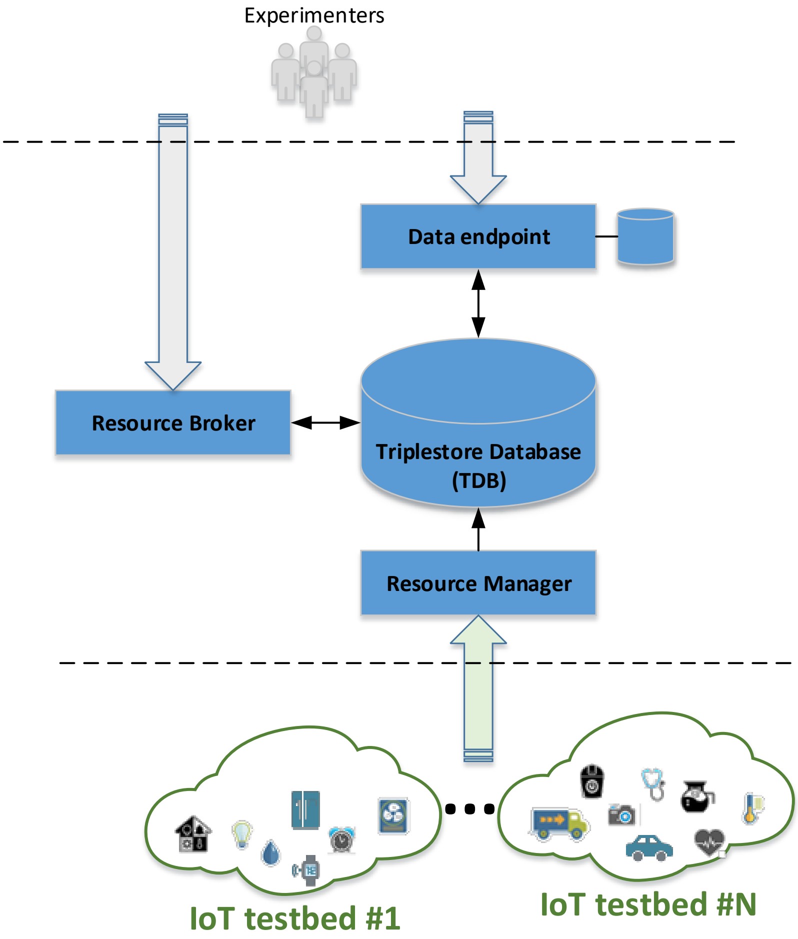


Figure 2. IoT-Registry internal architecture

The remaining two components, namely the Resource Manager (RM) and the Resource Broker (RB), are the ones that transform the IoT-Registry from a regular Semantic Datastrore into an enabler of the Web of Things paradigm. In this sense, the two key concepts behind this behaviour is the testbed-agnostic nature of the FIESTA-IoT platform and the service-oriented character of the IoT ARM [3] which underlies all the FIESTA-IoT Platform architecture. In this sense, the key idea is that the services exposing the resources from the underlying testbeds can be accessed using a truly web-oriented style. For this to happen, the IoT-Registry firstly "impersonates" the underlying resources (meaning that it exports a homogenized URI under the FIESTA-IoT domain namespace) for each of the federated testbeds and afterwards, provides a brokering mechanism that enables unified and proxied access to the underlying resources and the service endpoints that are used to expose them.

3.2 IoT-Registry components description

This section describes in detail the main insights from the implementation of the different modules that comprises the *IoT-Registry*.

3.2.1 Semantic data model

As for the semantic information compatible with this module, it is worth referring to [5] to get a clear reference to what kind of datasets the *IoT-Registry* has to deal with. Based on the acquaintance achieved with the analysis of the in-house testbeds, together with the support of some of the most well-known off-the-shelf ontologies on Sensor Networks, such as SSN[1] (Semantic Sensor Network), we gave rise to the FIESTA-IoT ontology[2], tailored to span various IoT domains, e.g. Smart Cities, Smart Building, Smart Health Care or Smart Agriculture, just to cite a few.

This data model is loaded when the IoT-Registry is deployed at the application server. The model underlies all the semantic features offered by the IoT-Registry such as the query engine, internal validation of triples and documents, etc.

3.2.2 Storage structure of the TDB

As it has been previously mentioned, the information that is stored at the FIESTA-IoT Meta-Cloud relates to two different, but tightly bound, realms. On the one hand, the descriptions of the resources that form the underlying testbeds (i.e. the IoT devices) and, on the other hand, the observations made by that sensors[3].

Similarly, the internal structure of the TDB follows a similar approach. In this sense, the implementation of the Jena-based query engine has two different graphs that are virtually merged into a third one, as can be seen in Figure 3 below.

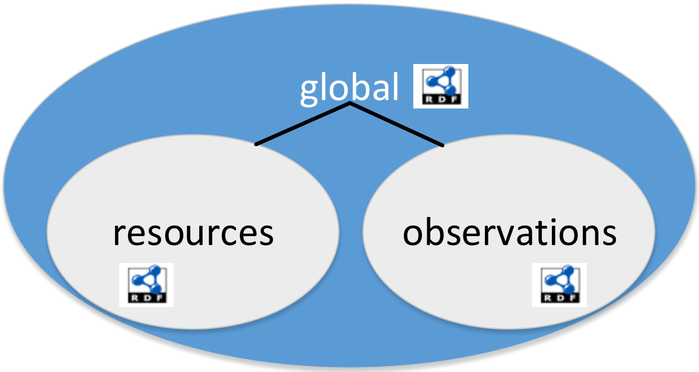


Figure 3. IoT-Registry TDB internal structure

The *resources* and *observations* graphs store respectively the resource descriptions of the IoT devices and the observations that they generate.

As it can be seen in Figure 4 below, the linked graphs that the instances of each of these items form, are mostly independent and, indeed, can be queried individually if the experimenter is interested only on information related to one of them. For example, the experimenter can perfectly look for the Service that is exposing any of the IoT devices using the typical what (i.e. physical phenomenon observed) and where (i.e. location) discovery criteria. For this search, only the resources graph

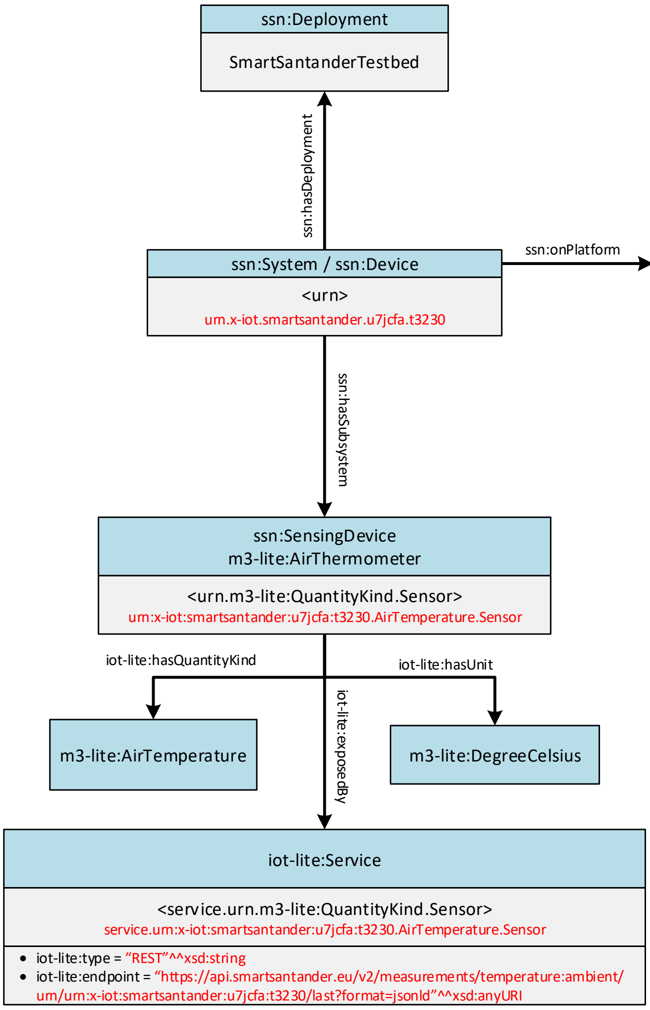
[1] https://www.w3.org/2005/Incubator/ssn/ssnx/ssn

[2] http://ontology.fiesta-iot.eu/ontologyDocs/fiesta-iot/doc

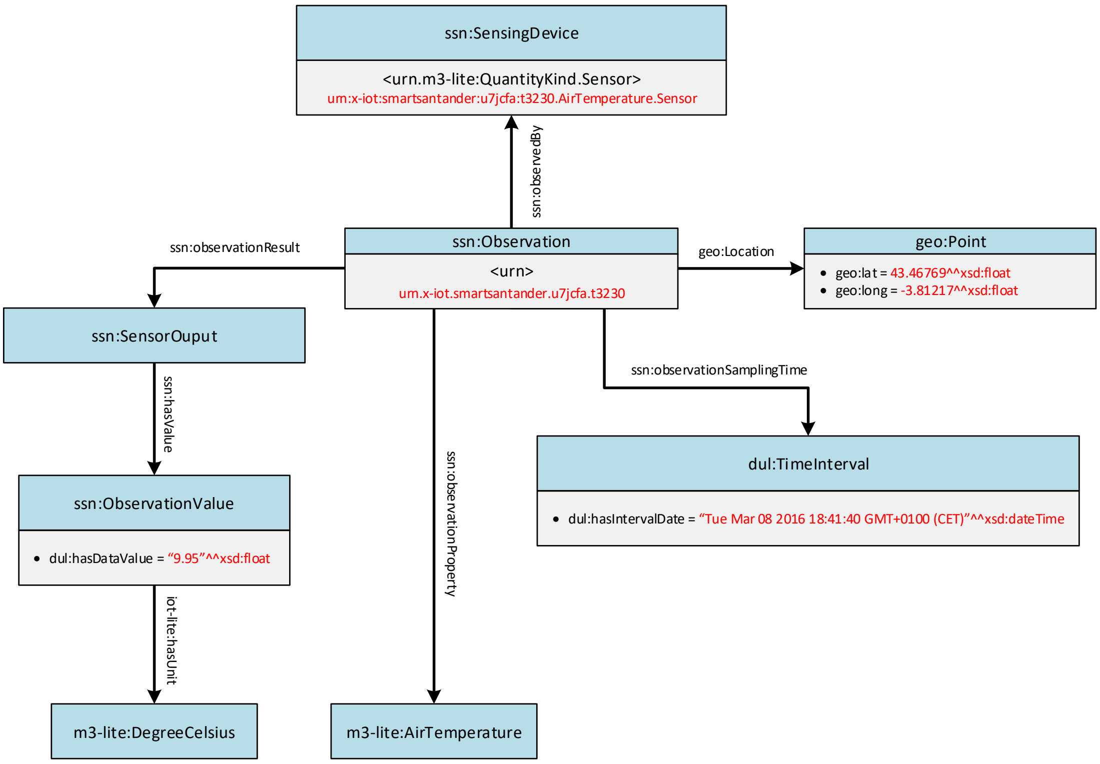
[3] At the time of writing this document, testbeds only provide sensor-based assets. Nonetheless, in case new platform came with different devices (e.g. actuators or tags), the ontology would be modified in order to foster these new elements.

should be explored by the query engine, thus, optimizing the discovery and access performance. A similar example can be imagined if the experimenter is interested on the data contained on the actual measurements collected by the sensors at the underlying testbeds as they are also self-contained in terms of geo-location, timestamp and phenomenon observed.

However, in the cases on which the experimenter is looking for extra information about the IoT device that has produced a measurement (e.g. coverage, accuracy, sensing procedure or other metadata), this information can only be obtained from the resources graph and if the two graphs weren't virtually bound, the experimenter would have to execute two different queries against each of the two graphs. The solution adopted caters for the flexibility of allowing optimized queries when they target only one of the graphs but at the same time allows more complex queries targeting information from both of them.


ssn:Deployment
SmartSantanderTestbed
ssn:hasDeployment
ssn:System / ssn:Device
<urn>
urn.x-iot.smartsantander.u7jcfa.t3230
ssn:onPlatform
ssn:Platform
<platform.urn>
platform.urn.x-iot.smartsantander.u7jcfa.t3230
ssn:hasSubsystem
geo:Location
ssn:SensingDevice
m3-lite:AirThermometer
<urn.m3-lite:QuantityKind.Sensor>
urn:x-iot:smartsantander:u7jcfa:t3230.AirTemperature.Sensor
geo:Point
<location.urn>
location.urn.x-iot.smartsantander.u7jcfa.t3230
• geo:lat = 43.46769^^xsd:float
• geo:long = -3.81217^^xsd:float
iot-lite:hasQuantityKind
iot-lite:hasUnit
iot-lite:exposedBy
m3-lite:AirTemperature
m3-lite:DegreeCelsius
iot-lite:Service
<service.urn.m3-lite:QuantityKind.Sensor>
service.urn:x-iot:smartsantander:u7jcfa:t3230.AirTemperature.Sensor
• iot-lite:type = "REST"^^xsd:string
• iot-lite:endpoint = "https://api.smartsantander.eu/v2/measurements/temperature:ambient/urn/urn:x-iot:smartsantander:u7jcfa:t3230/last?format=jsonld"^^xsd:anyURI

(a)


ssn:SensingDevice
<urn.m3-lite:QuantityKind.Sensor>
urn:x-iot:smartsantander:u7jcfa:t3230.AirTemperature.Sensor
ssn:observedBy
ssn:observationResult
ssn:Observation
<urn>
urn.x-iot.smartsantander.u7jcfa.t3230
geo:Location
geo:Point
geo:lat = 43.46769^^xsd:float
geo:long = -3.81217^^xsd:float
ssn:SensorOuput
ssn:hasValue
ssn:observationProperty
ssn:observationSamplingTime
dul:TimeInterval
dul:hasIntervalDate = "Tue Mar 08 2016 18:41:40 GMT+0100 (CET)"^^xsd:dateTime
ssn:ObservationValue
dul:hasDataValue = "9.95"^^xsd:float
iot-lite:hasUnit
m3-lite:DegreeCelsius
m3-lite:AirTemperature

(b)

Figure 4. (a) Resource description annotation example. (b) Observation annotation example

3.2.3 Data Endpoint

SPARQL is known to be the most common and widely used RDF query language. Therefore, it is sensible to offer a fully-fledged SPARQL interface, as part of the *IoT-Registry* module that allows this kind of semantic queries on the FIESTA-IoT platform. The Data Endpoint (DE) module implements this functionally by enabling a direct SPARQL endpoint.

The DE is a conformant SPARQL protocol service as defined in the SPARQL Protocol for RDF (SPROT). It allows users to query a knowledge base via the SPARQL language. Results are returned in any of the common data representation formats, namely JSON, XML, CSV, etc.

The default endpoint runs the query on the "global" graph, including both Resources and the linked Observations. However, it is also possible to limit the scope of the query to just the Resources or the Observations graph by identifying the graph in the request URL.

Moreover, it also offers a system for the storage of queries so that its execution can be programed without having to include the complete SPARQL sentence at every request. This additional functionality would make it easier to share knowledge between experimenters or testbeds and smooth the learning curve when it comes to cope with the FIESTA-IoT ontology and its underlying dataset representation.

Finally, an additional functionality has been added to the DE so that the stored SPARQL queries can be dynamically adapted and used as templates rather than as static queries. To achieve such a feature, the REST API wrapping the DE allows for some variables to be replaced with input parameters in the GET/POST requests based on a set of conventions. The syntax is based on a proprietary defined wildcard: *%%%*, that will never appear in a regular SPARQL query, which delimits a parameter name. Then, by adding the parameter as a URL query parameter, the assigned value will replace it before executing the query.

This feature has been added with a twofold objective. On the one hand, it promotes sharing queries, thus giving rise to a sort of “crowd-sourced” catalogue. Moreover, it enables the creation of optimized queries resolving recurrent demands from experimenters. This way it will be possible to create a “best-practices” catalogue open to experimenters. On the other hand, this option reduces the overhead and eases the action of executing multiple times the same SPARQL sentence as caching can be used to enhance the query engine performance.

3.2.4 Resource Manager

The Resource Manager (RM) exposes the single-entry point for all the testbeds to register their IoT Resources’ descriptions. Its main role is to homogenize the descriptions received from the different testbeds. After syntactically checking the annotated descriptions and guaranteeing that they are compliant with the FIESTA-IoT ontology, the RM transforms the URI for all the resource descriptions in order to make them belong to the FIESTA-IoT namespace. This process basically consists of overwriting the bindings that points to the original testbeds’ domains included in the annotated resource descriptions. These bindings are transformed to the common meta-platform domain so that every entity identifier and/or IoT Service endpoint, independently of which testbed they belong to, are exposed as if they belonged to a unique graph, namely the federation graph.

Therefore all the semantically annotated descriptions generated by the testbeds are stored in the Triplestore Database following the testbed agnostic paradigm followed within FIESTA-IoT. Once the necessary adaptations to the resource descriptions have been done and internally recorded for future use by the RB, the RM stores them into the TDB.

While the communication interface between the RM and the TDB will be based on semantic requests, the interface with the testbeds is based on standard HTTP encapsulating semantically annotated documents.

3.2.5 Testbed agnostic namespace transformation

One of the key objectives for the whole FIESTA-IoT project is to specify and implement techniques for testbed agnostic access to data sets stemming from multiple heterogeneous IoT platforms. The IoT-Registry is at the core of these techniques and there is a particular procedure executed within the RM that serves towards this objective.

When any of the federated testbeds register their IoT devices or send their observations to the IoT-Registry for storage, the triples of the annotated documents that represent them are bound to the respective namespace of the testbed through the URI that identifies each of the nodes in the graph (e.g. http://smart-ics.ee.surrey.ac.uk/fiesta-iot/deployment#smart-ics, http://iotocean.org/ontologies/ketiOntology.owl#fil_510.co2, http://api.smartsantander.eu#ft4cev:t5115.fillLevel-wasteContainer.obs-1375, etc.).

Keeping this kind of URIs would go against the testbed agnostic nature that is expected from FIESTA-IoT platform. Moreover, as it will be presented when describing the insights of the Resource Broker (RB), keeping the original namespaces of the underlying testbeds would make impossible the realization of a true Web of Things which is another key objective of the IoT-Registry.

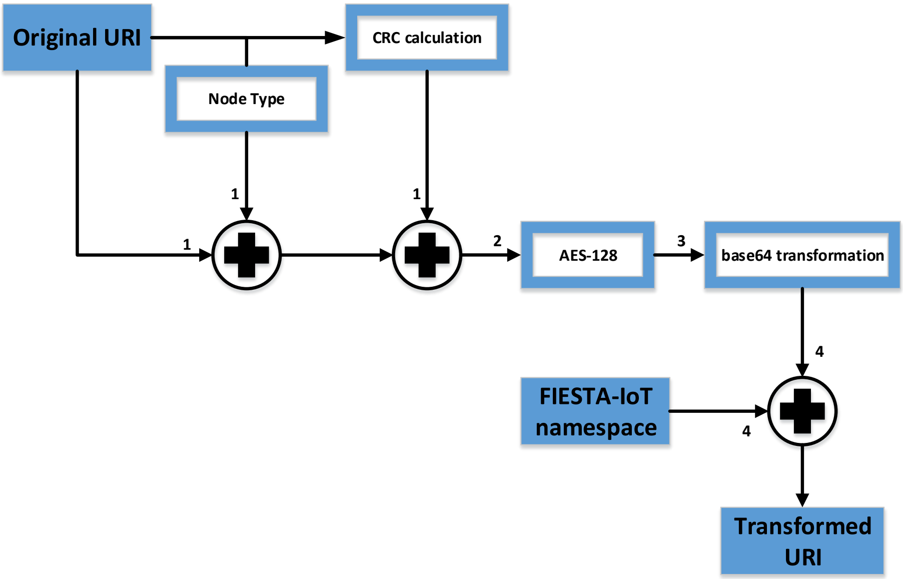


Figure 5. URI namespace transformation flow diagram

Figure 5 above shows the process implemented within the RM for the transformation of the URIs for all the nodes stored at the TDB graph database.

1. The original URI is appended (actually additional characters are added as prefixes of the original URI) with a Cyclic Redundancy Check (CRC) and an integer which represents the node type (0 if the node is a testbed, 1 if the node is an observation, and 2 for the rest).
2. The resulting string is hashed using AES-128 block-cypher.
3. The resulting byte array is transformed to base64 representation.
4. The resulting string is appended to the corresponding FIESTA-IoT namespace resulting in the corresponding transformed URI.

For example, the resulting transformed URI for one of the testbeds original URI (*http://api.smartsantander.eu#SmartSantanderTestbed*) is *https://platform.fiesta-iot.eu/iot-registry/api/testbeds/a1yp9GcKEPw37Bx5rslgRI4QLSNCwEwBatCIOe_W0dHZCmzj2WmkExz3qoNuvWg1pueAXn1Li0JrNjvBiQwV3Q==*.

3.2.6 Resource Broker

Apart from the direct extraction of data from the TDB through the DE by executing SPARQL queries, the FIESTA-IoT platform also supports the access to the services that directly expose the underlying IoT devices. In this sense, the FIESTA-IoT ontology allows the option[4] for testbed providers to include, within the resource

[4] Despite being part of the semantic data model (i.e. ontology), devices do not have to include a linkable endpoint through which experimenters might get data from.

description of their IoT devices, information about the Services that expose this IoT device.

However, for the experimenter that makes use of FIESTA-IoT Platform to carry out an experiment that uses these services, accessing them directly on their original testbed would require an extra effort (i.e. for understanding each testbed internal mechanisms). Moreover, this would break the testbed agnostic paradigm that FIESTA-IoT has as one of its main objectives.

The Resource Broker is the component in charge of enabling the access to IoT devices' Services while keeping the testbed agnostic nature of FIESTA-IoT and also homogenizing the way of accessing them for the experimenter.

As it has been described in the previous section, graph nodes URIs are transformed for them to belong to the unified FIESTA-IoT namespace. This transformation makes the service endpoint to target the FIESTA-IoT namespace and more specifically the IoT-Registry. The RB intercepts the requests made to the transformed URIs and forwards it to the corresponding testbed endpoint. This process is carried out internally at the RB so that for the experimenter it is completely transparent and it gets the service result without having to care about the specific testbed requirements. In this respect, it is the RB the one that is implemented to deal with that specific requirements imposed by each of the underlying testbeds.

3.3 Prototype implementation and deployment

In this section we provide an installation and basic implementation description of the *IoT-Registry*. At this point, it is worth to mention that although the specification of the APIs offered by the *IoT-Registry* have been described in [9]. Moreover, although IoT-Registry implementation is 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 [1] that is a "living document" and will get updated as the platform evolves. This Handbook is a public document which is offered to the FIESTA-IoT platform users.

3.3.1 Source code availability and structure

The *IoT-Registry* component is offered at FIESTA-IoT GitLab repository which is named "IoT-Registry" and is available at: https://gitlab.fiesta-iot.eu/platform/iot-registry. The latest version of the components is under the "develop branch"[5] .

The repository is organized in 3 categories/folders and the *IoT-Registry* component is placed as follows:

- conf: provides all the necessary configuration parameters.
- raml: provides the on-line documentation of the IoT-Registry APIs
- src/main: contains the source code for the different building blocks of the IoT-Registry

[5] https://gitlab.fiesta-iot.eu/platform/iot-registry/tree/develop

3.3.2 Component deployment

A detailed installation and deployment guide for this component is available at the FIESTA-IoT GitLab. In the following subsections, we will summarize the main aspects to be taken into account for the deployment of the *IoT-Registry*.

3.3.2.1 System Requirements

The IoT-Registry component is deployed within a WildFly container and use Maven for project management. 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.

Additionally, IoT-Registry uses an internal MySQL database for the persistence of the SPARQL queries. This additional feature of the DE module, enables storing of the queries to be passed to the Jena-based engine.

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 prototype are given in the section below.

3.3.2.2 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 initialize 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) **iot-registry.war** file from the **target** directory (**$PROJECT_HOME/target/**), into the **standalone/deployments** directory of the WildFly[6] distribution, in order to be automatically deployed.

[6] http://wildfly.org/

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/api

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

3.3.2.3 API Usage

The details about the usage of the *IoT-Registry* offered APIs is described in [9] together with the explanation on how to discover and access to the interoperable datasets stored at the *IoT-Registry* in a testbed agnostic manner.

In any case, on-line documentation is available on https://platform.fiesta-iot.eu/iot-registry/docs/api.html.

3.3.2.4 Containers and Libraries

Table 1 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 1 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
fiesta-commons	0.0.1
mysql-connector	5.1.40
resteasy	3.1.1.Final
hibernate	5.1.0.Final
jena	3.2.0
jts	1.13

4 DOMAIN SPECIFIC LUNGUAGE MODEL

4.1 Overview

FIESTA-IoT experiment management is facilitated by the usage of a Domain Specific language (DSL) that is capable of hosting all the experiments of a specific User. This language is called FIESTA-IoT Experiment Description Specification (FEDSpec) and its core structure is depicted in Figure 6 below. In this section we are going to present the FEDSpec DSL as well as an example of how to use it and the current supported entities from the FIESTA-IoT Experiment Execution Engine (EEE).

4.2 FIESTA-IoT Experiment Model Object (FEDSpec)

FIESTA-IoT Experiment Description Specification (FEDSpec) is an XML document whose structure is described in the XML Schema Definition (XSD) provided in Table 9 of APENDIX I below. FEDSpec, as shown in Figure 6, can host all the defined experiments of an Experimenter by including multiple FIESTA-IoT Experiment Model Objects (FEMO).

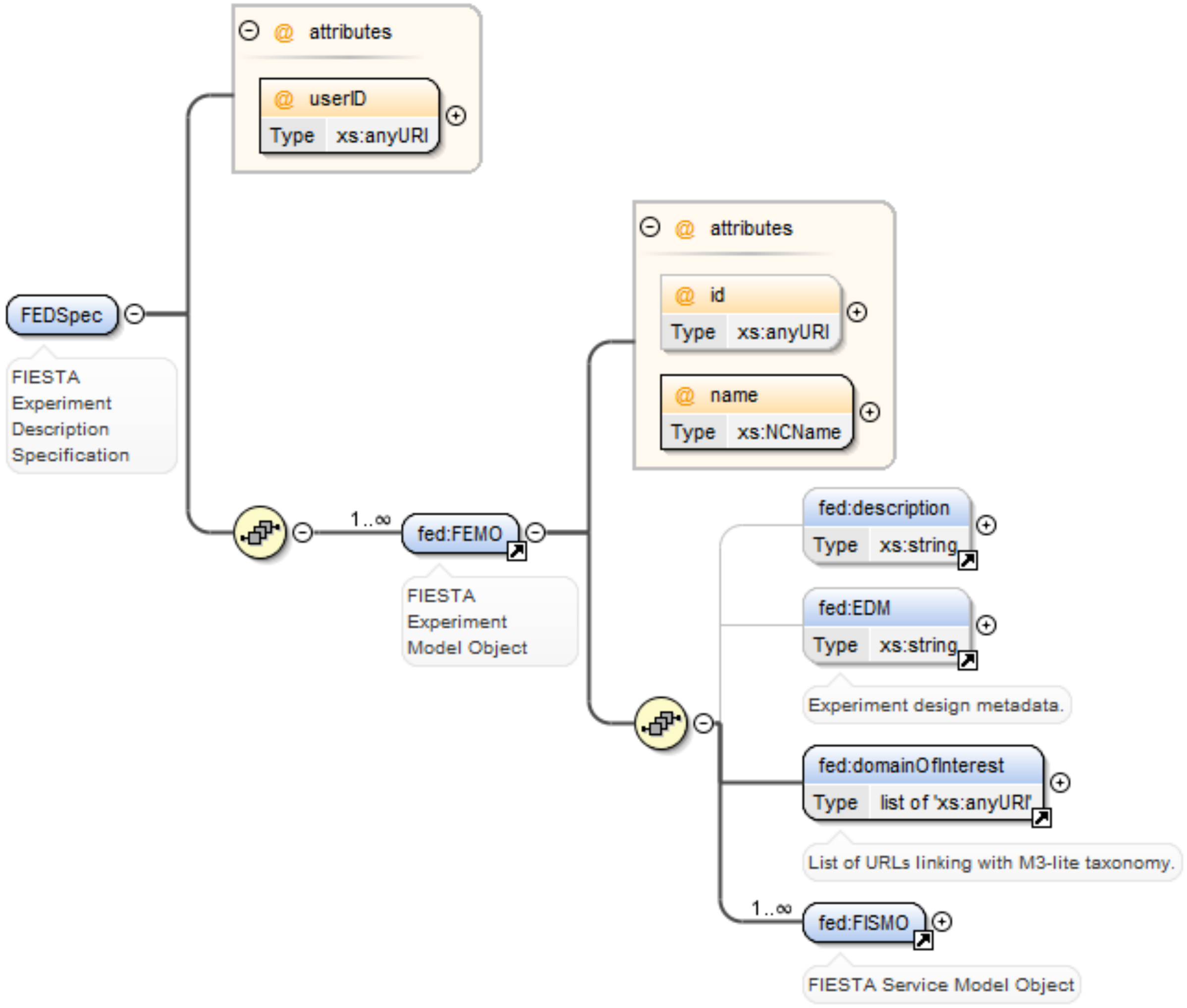


Figure 6. FEDSpec Schema graph

FEMO is responsible of holding the description of a single experiment and consists of:

- The **description**: (Optional) where a short textual description of the experiment can be added.
- The **Experiment Design Metadata (EDM)**: (Optional) where the graphical metadata of the experiment editor can be stored (i.e. node-red) that facilitate the conversion of a FEMO to a graphical representation of the experiment to an Editor (these metadata are editor specific).
- The **domain of interest**: where we can have the list of domains of interest of the experiment, which can be used for discovery purposes, based on the M3-lite taxonomy.
- The **FIESTA-IoT Service Model Object (FISMO)**: which is the main and most important experiment descriptive entity, one or many of which can be included in a FEMO object and thus create a complete experiment.

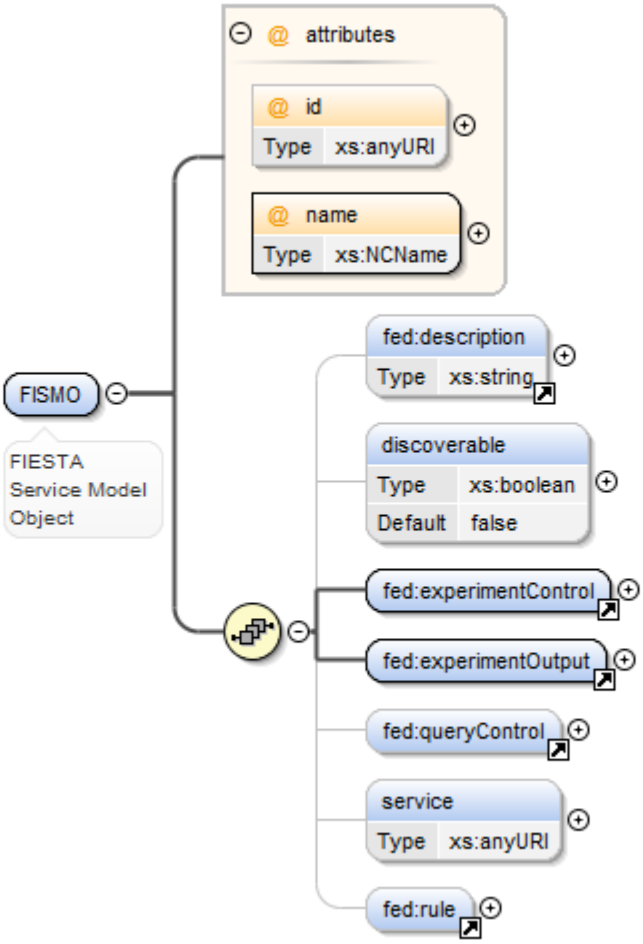


Figure 7. FISMO Schema graph

FISMO, as shown in Figure 7 above, consists of the following entities:

- The **description**: (Optional) where a short textual description of the experiment's Service can be added.
- The **discoverable**: (Optional) here a Boolean defines if the experiment is discoverable or not.

- The **service**: (Optional) here the URL for accessing the “iot-lite:Service” can be hosted for providing access to the semantic datasets.
- The **experiment control**: which controls how the Service should be processed; in particular, it specifies the conditions under which the service should be invoked (e.g., specifying a periodic schedule, defining specific visualizations, etc.). This control is facilitated by the following entities (shown in Figure 8 below):
    - **Scheduling**: (Optional) which may be defined to specify a periodic schedule for query execution for a specific subscription. The schedule of the Service could be defined, if required, by identifying:
        - Start time: (Optional) the time that the service should be started.
        - Periodicity: (Optional) how often the service will be executed.
        - Stop time: (Optional) when the service should be stopped.
    - **Trigger**: (Optional) the URL that should be invoked in order to trigger the execution of the Service if required
    - **Report if empty**: (Optional) If true, a Result Set is always sent to the subscriber when the query is executed. If false, a Result Set instance is sent to the subscriber only when the results are non-empty.

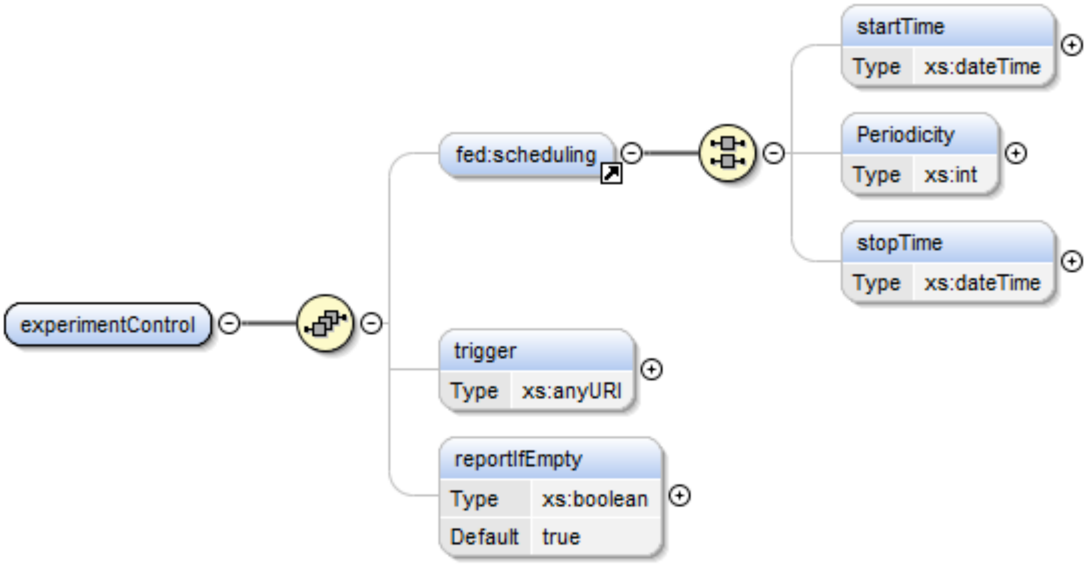


Figure 8. Experiment Control Schema graph

- The **experiment output**: (shown in Figure 9 below) which defines the required information for the instantiation of the output of an experiment. The output could be provided in various ways i.e. visualization (widget) at a webpage or as a file. The output configuration is facilitated by the following entities:
    - The **location**: which hosts the URI where the output should be sent.
    - The **file**: (Optional) where the file type of the output can be identified The current valid list of files types that are supported are:
        - *"text/plain",*

- *"text/tab-separated-values",*
- *"text/csv",*
- *"application/sparql-results+json",*
- *"application/sparql-results+xml",*
- *"application/sparql-results+thrift",*
- *"application/json",*
- *"text/xml"* and
- *"application/xml"*.

The Experimenter should choose either one of them.

- The **widget**: (Optional) which defines the required information for the instantiation of the presentation GUI as defined at the initial user request at the experiment definition time. For example, the widget that is going to be used for representing the data like a speedometer, a spreadsheet, a map or a diagram.

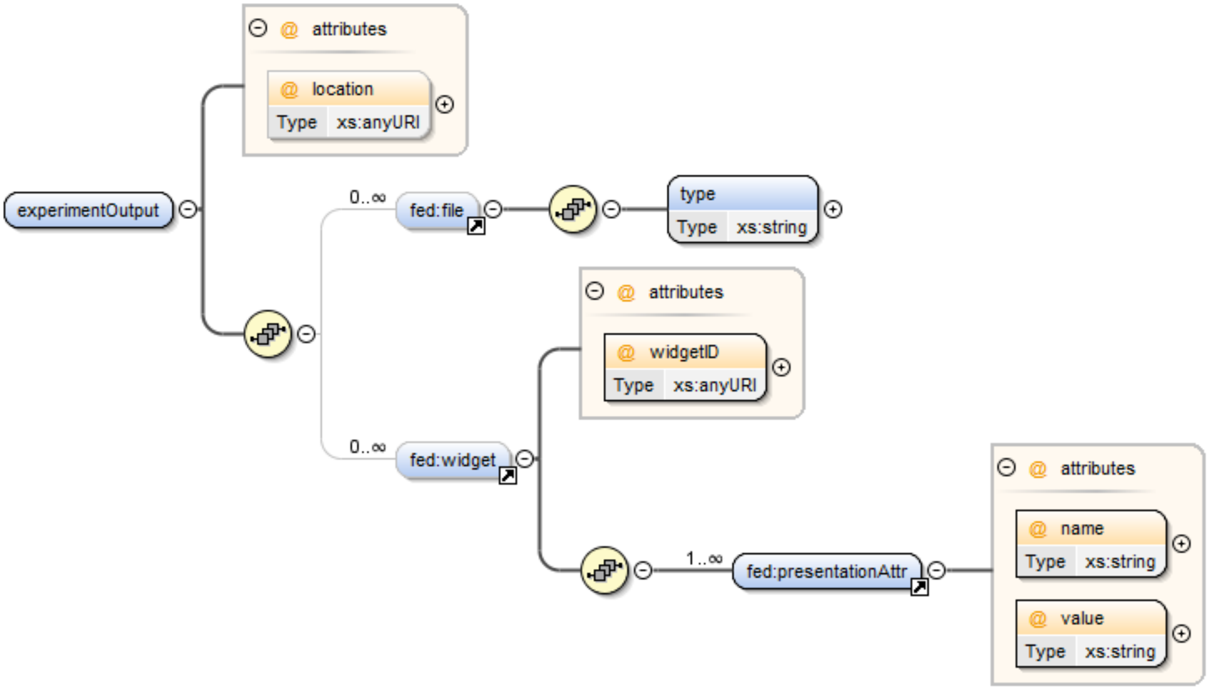


Figure 9. Experiment Output Schema graph

- The **query control**: is responsible to host the description of the required query that should be executed in order to provide the results/data from this experiment Service. Along with the query itself it holds additional entities that enable the discovery and dynamic configuration of the query. The query control configuration is facilitated by the following entities (shown in Figure 10 below):
  - **Quantity kind**: (optional) a list of quantity kind URIs, based on the m3-lite taxonomy, to identify the type of measurements involved in this query.

- **Static location**: (optional) the geo-coordinates of the experiment/query that will facilitate the experiment discovery.
- **Query interval**: (optional) provides the time interval that the query should collect data from by explicitly specifying the start/stop date/time (fromDateTime - toDateTime). Alternatively it can hold the duration in seconds from the present to the past in order to dynamically calculate the start/stop date/time. The latter is used if the experiment wants to get at each execution the values of the last X hours, X days etc.
- **Query request**: this entity is the most important entity of the FISMO object and holds the W3C query data (SPARQL[7]) that should be executed in order to retrieve the results of the experiment Service.

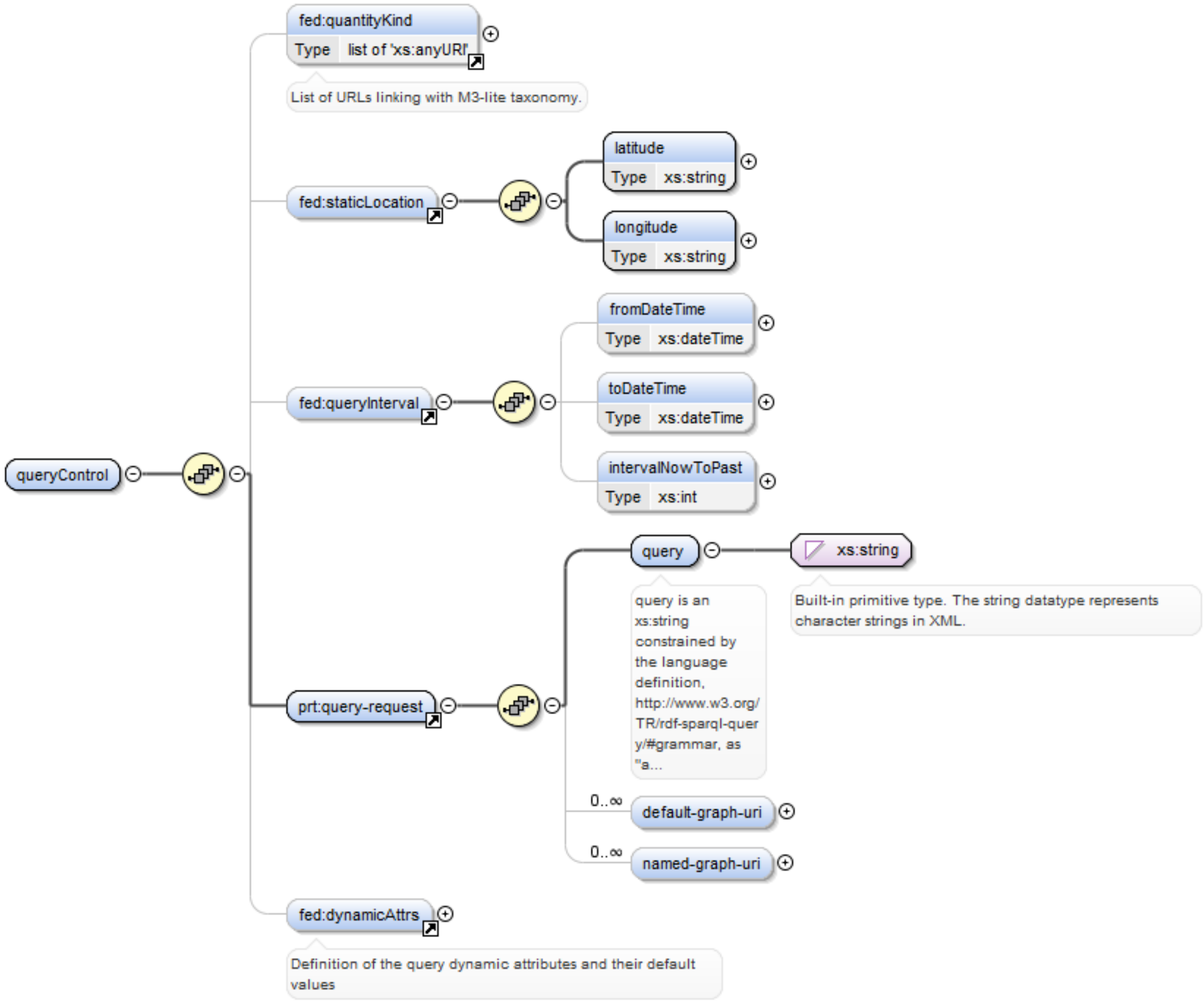

fed:quantityKind
Type list of 'xs:anyURI'
List of URLs linking with M3-lite taxonomy.
fed:staticLocation
latitude
Type xs:string
longitude
Type xs:string
fromDateTime
Type xs:dateTime
toDateTime
Type xs:dateTime
intervalNowToPast
Type xs:int
fed:queryInterval
queryControl
query
xs:string
query is an xs:string constrained by the language definition, http://www.w3.org/TR/rdf-sparql-query/#grammar, as "a...
Built-in primitive type. The string datatype represents character strings in XML.
prt:query-request
0..∞
default-graph-uri
0..∞
named-graph-uri
fed:dynamicAttrs
Definition of the query dynamic attributes and their default values

Figure 10. Query Control Schema graph

- **Dynamic attributes**: (optional) this entity provides the ability to dynamically update attributes of the query at the runtime of an experiment (i.e. in a mobile application to provide the CO2 level of the current location the experiment geo-coordinates should be updated in each experiment execution). Dynamic attributes provides predefined

[7] https://www.w3.org/TR/sparql11-query/

attributes (the predefined dynamic attributes) and experimenter defined attributes (list of dynamic attribute) shown below:

- **Predefined dynamic attributes**: (optional) it provides a list of dynamic attributes with a predefined name and a user defined initial value. These attributes currently include the:
    - dynamic query interval which further contains:
        - fromDateTime,
        - toDateTime and
        - intervalNowToPast which is defined as milliseconds from now to past. If this is set then fromDateTime and toDateTime are not used.
    - dynamic geo-location: This contains latitude and longitude information in the string format.
- **Dynamic attribute**: (optional) it provides a list of dynamic attributes with a user-defined name and a user defined initial value.

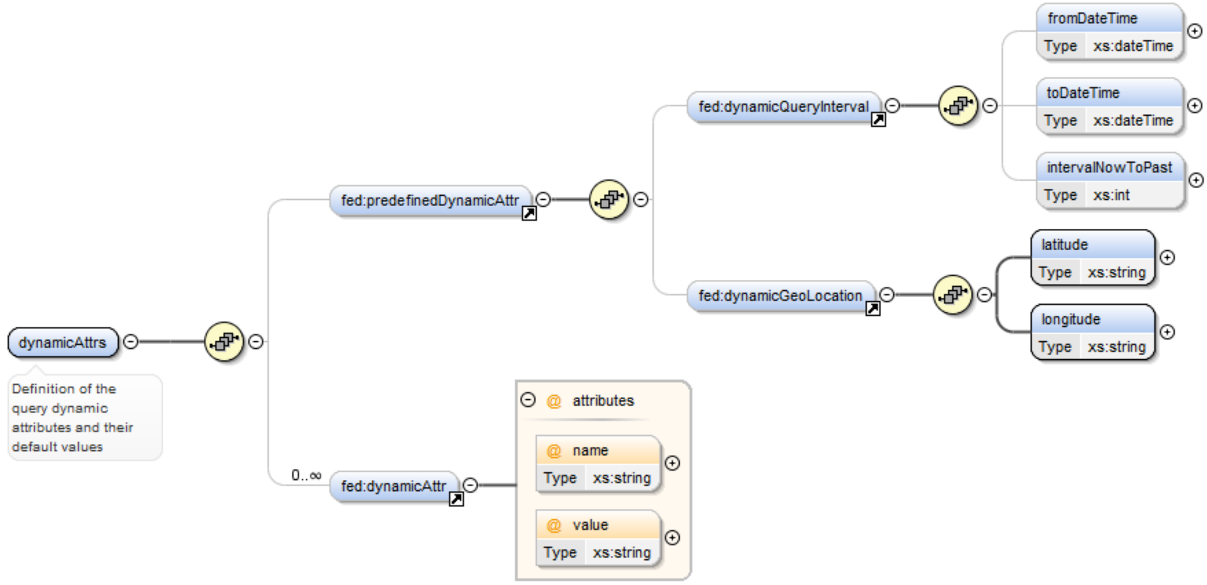


Figure 11. Dynamic Attributes Schema graph

- The **rule**: (optional) which hosts a rule that should be applied on top of the service results. The rule configuration is facilitated with the help of the following entities (shown in Figure 12 below):
    - The **name**: a textual representation of the applied rule name
    - The **rule definition**: a textual representation of the applied rule code which can be a Jena rule or a SPARQL CONSTRUCT.
    - **Domain Knowledge**: the URI stating the domain knowledge of the rule
    - **Quantity kind**: a list of quantity kind URIs based on the m3-lite taxonomy.

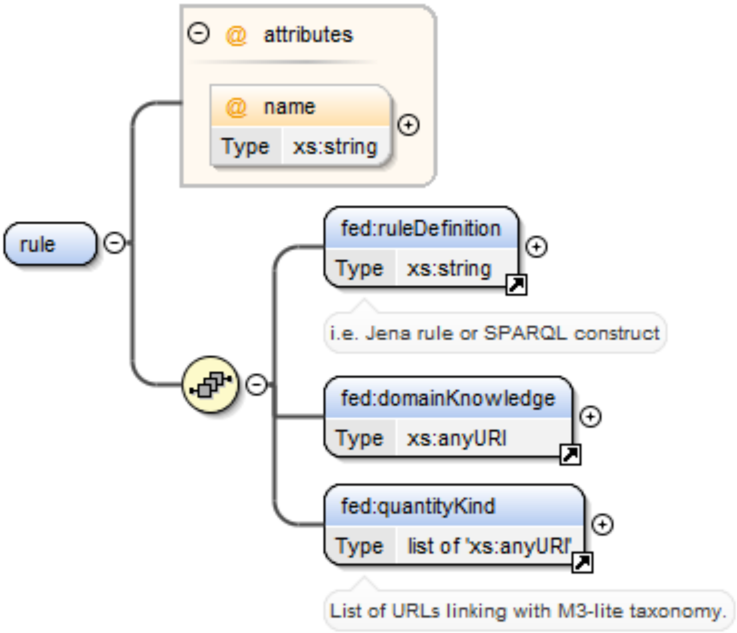


Figure 12. Rule Schema graph

4.3 Defining an experiment through Domain Specification Language (DSL)

In the following example, we are going to provide a simple experiment definition that is consumed by the Experiment Execution Engine (EEE) and produce the required results. A simple FISMO template is provided below. Experimenters should consider replacing “#*#” with experiment requirements.

```
<fed:FISMO name="#NAME#">
 <fed:description>#DESCRIPTION#</fed:description>
 <fed:discoverable>#DISCOVERABLE#</fed:discoverable>
 <fed:experimentControl>
  <fed:scheduling>
   <fed:startTime>#STARTTIME#</fed:startTime>
   <fed:Periodicity>#PERIODICITY#</fed:Periodicity>
   <fed:stopTime>#STOPTIME#</fed:stopTime>
  </fed:scheduling>
  <fed:reportIfEmpty>#REPORTIFEMPTY#</fed:reportIfEmpty>
 </fed:experimentControl>
 <fed:experimentOutput location="#URLLOCATION#">
  <fed:file>
   <fed:type>#FILETYPE#</fed:type>
  </fed:file>
  <fed:widget widgetID="eu.fiesta_iot.analytics.toolkit">
   <fed:presentationAttr name="requestBody" value="#WVALUE#"/>
  </fed:widget>
 </fed:experimentOutput>
 <fed:queryControl>
  <prt:query-request>
   <query><![CDATA[
          #[1/1] visualization type: 'Gauge' and sensors
```

```
             #QUERY#
       ]]>
   </query>
  </prt:query-request>
  <fed:dynamicAttrs>
   <fed:predefinedDynamicAttr>
    <fed:dynamicQueryInterval>
     <fed:fromDateTime>#FROMDATETIME#</fed:fromDateTime>
     <fed:toDateTime>#TODATETIME#</fed:toDateTime>
     <fed:intervalNowToPast>#INTERVALNOWTOPAST#</fed:intervalNowToPast>
    </fed:dynamicQueryInterval>
    <fed:dynamicGeoLocation>
     <fed:latitude>#LATITUDE#</fed:latitude>
     <fed:longitude>#LONGITUDE#</fed:longitude>
    </fed:dynamicGeoLocation>
   </fed:predefinedDynamicAttr>
   <fed:dynamicAttr name="#DA_NAME1#" value="#DA_VALUE1#"/>
   <fed:dynamicAttr name="#DA_NAME2#" value="#DA_VALUE2#"/>
  </fed:dynamicAttrs>
 </fed:queryControl>
</fed:Fismo>
```

To further explain,

- #NAME#: should be the name of the FISMO that experimenter want to have. For example:
  `<fed:FISMO name="2ndUseCase">`

- #DESCRIPTION#: should be the description of the FISMO to understand what is the FISMO is about. For example:
  `<fed:description>Over time all noise observations for a given location</fed:description>`

- #DISCOVERABLE#: should be either “true” or “false”. As explained before this attribute is used by the EEE to know whether the FISMO can be shown in the “Other Available FISMO IDs for subscriptions” tab in the Experiment Management Console. Using this other experimenters could reuse the query and scheduling information (subscribers should giving a new location that would send the results to them). For example:
  `<fed:discoverable>true</fed:discoverable>`

- #STARTTIME#: it is the time when the scheduling should start. In case, the #STARTTIME# in the past, the current time will be used and in case #STARTTIME# is in future, the given time will be used by the EEE to schedule the FISMO. The #STARTTIME# should be in DATETIME format.
  `<fed:startTime>2016-11-08T18:50:00.0Z</fed:startTime>`

- #PERIODICITY#: it is the period after which the EEE should re-trigger the execution of the FISMO. It is in INTEGER format and denotes the Seconds. For example:

```
<fed:Periodicity>250</fed:Periodicity>
```

- #STOPTIME#: it is the time when the EEE should stop executing the FISMO. In case, the #STARTTIME# in the past, and is less than #STARTTIME# an error is raised by the EEE. Thus it is advisable that the #STOPTIME# is in future and is greater than #STARTTIME# The #STOPTIME# should be in DATETIME format. For example:
  ```
  <fed:stopTime>2017-11-08T18:49:59.0Z</fed:stopTime>
  ```

- #REPORTIFEMPTY#: It is the Boolean value ("true" or "false") that should be set in order for the experimenter to not to receive empty resultsets obtained after the execution of the query. A value true mean that if the resultset was null then do also report it. By default the value is false. For example:
  ```
  <fed:reportIfEmpty>false</fed:reportIfEmpty>
  ```

- #URLLOCATION#: it should be the location where the results of the query should be returned. This is usually a valid URL location. Please note that an experimenter cannot update this parameter once the FEDSpec is submitted. It is assumed that the experimenter has developed the functionality behind this link where EEE can upload the results in a "multipart" files format. In the current implementation of EEE, it is also assumed that this is a REST based API that implements HTTP POST with following REQUEST parameters
  connection : keep-alive

  Content-Type: multipart/form-data; boundary=--timestamp

  An example URLLOCATION is as below:

  ```
  <fed:experimentOutput
  location="http://example.org/ExperimentServer/store/"></fed:exp
  erimentOutput>
  ```

- #FILETYPE#: this parameter defines the response content-type. As described above valid content-types that are received are "text/plain", "text/tab-separated-values", "text/csv", "application/sparql-results+json", "application/sparql-results+xml", "application/sparql-results+thrift", "application/json", "text/xml" and "application/xml". The Experimenter should choose either one of them. Please note that we do not set the extension of the file itself. For example:
  ```
  <fed:file><fed:type>application/xml</fed:type></fed:file>
  ```

- #WVALUE# is a JSON string that is of form:
  ```
  {
   "Method": ["Method 1"," Method 2"," Method 3"],
   "Parameters": ["Parameters 1", "Parameters 2", "Parameters 3"]
  }
  ```

Setting this value will enable EEE to know if the FIESTA-IoT Analytics has to be executed or not. If the experimenters do not want this then they simply just not provide the widget tag. Further, if this is set then the query should be of specific format as well. Please refer to the query template below:

```
Prefix …
Select distinct ?sensingDevice ?dataValue ?dateTime
Where {
…
}
```

Please note the SELECT statement. As per the guidelines set by FISTA-IoT Analytics tool, this SELECT statement should not be modified. The experimenters will receive a CSV file always if this is set. Setting the #FILETYPE# will have no effect. Also note that the permissible values of “Methods” and “Parameters” are:

Method	**Parameters**	**Description**
Outlier	*Thresh*	Value between **0** and **1**, selects the percentage of tail values to remove from the ordered time series data.
FilterData	*Type*	Select between, **“B”** Bandpass Filter, **“L”** Lowpass filter and Highpass filter **“H”**.
	cutoff_1	For the respective filters is the first normalised cutoff frequency, between 0 and 0.5.
	cutoff_2	For bandpass filter only, the second cutoff frequency.
	numtaps	Filter length. Usually select 30.
KMeans	*NumClusters*	The number of clusters to select. An integer value.
PCA	*Mode*	Select either, **“ExpVar”** the explained variance for the different principal components, or **“Comp”** the principal component loadings that is the direction in the data corresponds to the highest variance.
LinReg	*Type*	Select between, **“Param”** the estimated parameters of the regression model, and **“Predict”** the estimate of the output given the test data.
	Dependant	Select the column index corresponding to the dependent variable.
	Ratio	Select the ratio of the training data to test data. Value between 0 and 1.

KNNreg	*Num*	Selects the number of nearest neighbours.
	Dependant	Select the column index corresponding to the dependent variable.
	Ratio	Select the ratio of the training data to test data. Value between 0 and 1.
FFT	N/A	N/A
Periodogram	N/A	N/A
Correlation	N/A	N/A

An example “#WVALUE#” is as shown below:

```
{
 "Method": ["fft","linReg"],
 "Parameters": ["", "Predict"]
}
```

Note that the values in Parameters should have one-one mapping to values in Methods.

- #QUERY#: is the actual SPARQL query that should be executed by the EEE on the Meta-Cloud. For best results, we advise experimenters to not to provide following query and be specific to the needs.

```
select * where {?s ?p ?o.}
```

The above query would hinder the performance of the IoT-Registry component. A valid query would look like:

```
Prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
Prefix iotlite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#>
Prefix dul: <http://www.loa.istc.cnr.it/ontologies/DUL.owl#>
Prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
Prefix time: <http://www.w3.org/2006/time#>
Prefix m3-lite: <http://purl.org/iot/vocab/m3-lite#>
Prefix xsd: <http://www.w3.org/2001/XMLSchema#>
select ?s ?ti ?val
where {
      ?o a ssn:Observation.
      ?o ssn:observedBy ?s.
      ?o ssn:observedProperty ?qkr.
      ?qkr a ?qk.
      Values ?qk {m3-lite:Sound m3-lite:SoundPressureLevelAmbient}
      o ssn:observationSamplingTime ?t.
      ?o geo:location ?point.
      ?point geo:lat "4.346104E1"^^xsd:double.
      ?point geo:long "-3.80649E0"^^xsd:double.
```

```
        ?t time:inXSDDateTime ?ti.
        ?o ssn:observationResult ?or.
        ?or  ssn:hasValue ?v.
        ?v dul:hasDataValue ?val.
} group by (?s) ?ti ?val
```

Note that the #QUERY# should be between "`<![CDATA[#[1/1] visualization type: 'Gauge' and sensors`" and "`]]`".

The following parameters are to be used when dynamic attributes are needed in the query.

- #FROMDATETIME#: This attribute represents the "from" date time a query has to be executed. Within the query please use "%%fromDateTime%%" so that it is replaced with the value provided correctly. This attribute is dynamic so this attribute should be presented as a default value in DATETIME format in the FEDSpec. In the FEDSpec, experimenters can define it as, for example:
  `<fed:fromDateTime>2006-05-04T18:13:51.0Z</fed:fromDateTime>`

- #TODATETIME#: This attribute represents the "to" date time a query has to be executed. Within the query please use "%%toDateTime%%" so that it is replaced with the value provided correctly. This attribute is dynamic so this attribute should be presented as a default value in DATETIME format in the FEDSpec. In the FEDSpec, experimenters can define it as, for example:
  `<fed:fromDateTime>2006-05-04T18:13:51.0Z</fed:fromDateTime>`

- #INTERVALNOWTOPAST#: This attribute represents milliseconds from now to past. This is used in case experimenters do not want to specify #FROMDATETIME# and #TODATETIME#. This attribute is not needed to be represent in the query, however in the query, experimenters should still have "%%fromDateTime%%" and "%%toDateTime%%" so that EEE can process the #INTERVALNOWTOPAST# and replace them accordingly. EEE resolves "%%fromDateTime%%" as "current time - #INTERVALNOWTOPAST#". In the FEDSpec, experimenters can define it as, for example:
  `<fed:intervalNowToPast>300000</fed:intervalNowToPast>`

- #LATITUDE#: This attribute represents the "latitude" in a query that has to be executed. Within the query please use "%%geoLatitude%%" so that it is replaced with the value provided correctly. This attribute is dynamic so this attribute should be presented as a default value in float format in the FEDSpec. In the FEDSpec, experimenters can define it as, for example:
  `<fed:latitude>46.52119378179781</fed:latitude>`

- #LONGITUDE#: This attribute represents the "longitude" in a query that has to be executed. Within the query please use "%%geoLongitude%%" so that it is replaced with the value provided correctly. This attribute is dynamic so this

attribute should be presented as a default value in float format in the FEDSpec. In the FEDSpec, experimenters can define it as, for example:
`<fed:longitude>46.52119378179781</fed:longitude>`

- #DA_NAME_NUMBER# and #DA_VALUE_NUMBER# these attributes go hand in hand. They form a key value pair. It is advised that experimenters use the #DA_NAME_NUMBER# in the query as “%%DA_NAME_NUMBER%%”. Please note change of “#” to “%%”. In the FEDSpec, experimenters can define it as, for example:
  ```
  <fed:dynamicAttr name="qk" value="http://purl.org/iot/vocab/m3-lite#AirTemperature"/>
  <fed:dynamicAttr name="unit" value="http://purl.org/iot/vocab/m3-lite#Degree"/>
  ```

  Note that in the example #DA_NAME1# is “qk” and `DA_VALUE1=http://purl.org/iot/vocab/m3-lite#AirTemperature`

  A sample query that utilizes the dynamic attributes is mentioned below for reference:

  ```
  Prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
  Prefix iotlite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#>
  Prefix dul: <http://www.loa.istc.cnr.it/ontologies/DUL.owl#>
  Prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
  Prefix time: <http://www.w3.org/2006/time#>
  Prefix m3-lite: <http://purl.org/iot/vocab/m3-lite#>
  Prefix xsd: <http://www.w3.org/2001/XMLSchema#>
  Prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
  select ?sensorID (max(?ti) as ?time) ?value ?latitude ?longitude
  where {
        ?o a ssn:Observation.
        ?o ssn:observedBy ?sensorID.
        ?o ssn:observedProperty ?qkr.
        ?qkr rdf:type ?qk.
        Values ?qk {m3-lite:Sound m3-lite:SoundPressureLevelAmbient}
        ?o ssn:observationSamplingTime ?t.
        ?o geo:location ?point.
        ?point geo:lat ?latitude.
        ?point geo:long ?longitude.
        ?t time:inXSDDateTime ?ti.
        ?o ssn:observationResult ?or.
        ?or  ssn:hasValue ?v.
        ?v dul:hasDataValue ?value.
        FILTER ((xsd:double(?latitude) >= "-90"^^xsd:double)
              && (xsd:double(?latitude) <= "90"^^xsd:double)
              && ( xsd:double(?longitude) >= "-180"^^xsd:double)
              && ( xsd:double(?longitude) <= "180"^^xsd:double))
        FILTER(?value>="50"^^xsd:double)
        FILTER(?ti   >   "%%fromDateTime%%"^^xsd:dateTime   &&   ?ti   <
  "%%toDateTime%%"^^xsd:dateTime)
  } group by ?sensorID ?time ?value ?latitude ?longitude
  ```

On top of the FISMO there is a FEMO object. Currently EEE support the following FEMO template.

```
<fed:FEMO name="#FEMONAME#">
        <fed:description>#FEMODESCRIPTION#</fed:description>
        <fed:domainOfInterest>#DOMAINOFINTERESTLIST#</fed:domainOfInterest>
         <fed:FISMO name="#NAME#">
        …
        </fed:FISMO>
</fed:FEMO>
```

Here:

- #FEMONAME#: should be the name of the FEMO that experimenter want to have. For example:
  `<fed:FEMO name="MySecondExperiment">`

- #FEMODESCRIPTION#: should be the description of the FEMO to understand what is the FEMO is about. For example:
  `<fed:description>LargeScale crowdsensing experiment </fed:description>`

- #DOMAINOFINTERESTLIST#: this is the list of the domain of interests that experiment supports. For multiple domain of interests values should be blank space separated. For example.
  `<fed:domainOfInterest>http://purl.org/iot/vocab/m3-lite#Transportation http://purl.org/iot/vocab/m3-lite#Pollution http://purl.org/iot/vocab/m3-lite#City http://purl.org/iot/vocab/m3-lite#Health</fed:domainOfInterest>`

Further on the top of the FEMO object, there exists a FEDSpec object.

```
<fed:FEDSpec xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
             xmlns:fed="http://www.fiesta-iot.eu/fedspec"
             xmlns:prt="http://www.w3.org/2007/SPARQL/protocol-types#"
             xmlns:vbr="http://www.w3.org/2007/SPARQL/results#"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="FILELOCATION"
             userID="#USERID#">
        <fed:FEMO name="#FEMONAME#">
        …
        </fed:FEMO>
</fed:FEDSpec>
```

For complete valid sample please refer to APPENDIX II and more specifically in Table 11.

5 EXPERIMENT REGISTRY MANAGEMENT (ERM)

Experiment Registry Management (ERM) component exposes an API that facilitates the experiment storage, retrieval and discovery. This is achieved by manipulating objects that comply with the FEDSpec schema and its various defined entities.

5.1 API Definition

Table 2 below illustrates the main API primitives that support the Experiment Registry Management functionalities, while Table 3 provides more details about each one of the functions that comprise the API.

Table 2. List of primitives comprising the Experiment Registry Management API

<<interface>> ExperimentRegistryManagementInterface --- POST:**saveUserExperiments**(fedSpec:FEDSpec):String POST:**deleteUserExperiments** (userID:String):String POST:**saveUserExperiment**(femo:FEMO, userID:String):String POST:**deleteUserExperiment** (femoID:String):String POST:**saveUserExperimentServiceModelObject** (fismo:FISMO, femoID:String):String POST:**deleteExperimentServiceModelObject** (fismoID:String):String GET: **getALLUserExperiments** (userID:String):FEDSpec GET:**getAllUserExperimentsDescriptions** (userID:String):ExpDescriptiveIDs GET: **getExperimentDescription** (femoID:String):FemoDescriptiveID GET:**getExperimentModelObject** (femoID:String):FEMO GET:**getExperimentServiceModelObject** (fismoID:String):FISMO GET:**getDiscoverableExperimentServiceModelObjects** (): List<FISMO>

The FIESTA-IoT implements the methods of the Experiment Registry Management API as specified in Table 3 below:

Table 3. Experiment Registry Management API definition

Service Name	Input	Output	Info
saveUserExperiments	FEDSpec fedSpec	String	Used to submit the constructed experiment to the cloud. Requires as input the FIESTA-IoT Experiment Description Specification (FEDSpec) which includes all the User's preferences regarding the Experiment, request lifecycle and visualization. It returns the constructed Experiment ID.
deleteUserExperiments	String userID	String	Used to delete all User experiments. Returns a success message.
saveUserExperiment	FEMO femo, String userID	String	Used to save/update (if the Experiment does not contain a registered ID) a user experiment. Returns a success message.

deleteUserExperiment	String femoID	String	Used to delete a registered Experiment. Requires as input the Experiment ID. Returns a success message.
saveExperimentService ModelObject	FISMO fismo, String femoID	String	Used to Save/update (if the service model object does not contain a registered ID). Returns a success message.
deleteExperimentService ModelObject	String fismoID	String	Used to delete an Experiment Service. Returns a success message.
getALLUserExperiments	String userID	FEDSpec	Used to retrieve All the Experiments defined by a user. It returns an FIESTA-IoT Experiment Description Specification. Requires as input a User ID.
getAllUserExperiments Descriptions	String userID	Exp Descriptive IDs	Used to retrieve the available experiments (a list of experimentID/ServiceName/ServiceDescription triplet) already registered by a specific user. Requires as input a User ID.
getExperiment Description	String femoID	Femo Descriptive ID	Used to retrieve the available services (a list of serviceID/ServiceName/ServiceDescription triplet) already registered by a specific user. Requires as input the Service ID.
getExperimentModel Object	String femoID	FEMO	Used to retrieve the description (FEMO) of an available Experiment. Requires as input the Experiment ID
getExperimentService ModelObject	String fismoID	FISMO	Used to retrieve the description (FISMO) of an available service. Requires as input a Service ID.
getDiscoverableExperimentServiceModelObjects		List <FISMO>	Used to retrieve a list of discoverable Service Model Objects.

In Table 4. Experiment Registry Management API Usagebelow we can find the details for using the ERM API.

Table 4. Experiment Registry Management API Usage

Service Name	Service Usage
saveUserExperiments	POST /experiment.erm/rest/experimentservices/saveUserExperiments HTTP/1.1 Host: platform.fiesta-iot.eu iPlanetDirectoryPro: TOKEN_FROM_USER_CREDENTIALS

	Content-Type: application/xml Cache-Control: no-cache Body: <?xml version="1.0" encoding="UTF-8"?> <fed:FEDSpec xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" ..... </fed:FEDSpec>
deleteUserExperiments	POST /experiment.erm/rest/experimentservices/deleteUserExperiments?userID=USERID HTTP/1.1 Host: platform.fiesta-iot.eu iPlanetDirectoryPro: TOKEN_FROM_USER_CREDENTIALS Cache-Control: no-cache Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW
saveUserExperiment	POST /experiment.erm/rest/experimentservices/saveUserExperiment?userID=USERID HTTP/1.1 Host: platform.fiesta-iot.eu iPlanetDirectoryPro: TOKEN_FROM_USER_CREDENTIALS Content-Type: application/xml Cache-Control: no-cache Body: <?xml version="1.0" encoding="UTF-8" standalone="yes"?> <ns2:FEMO xmlns:ns2="http://www.fiesta-iot.eu/fedspec" ...... </ns2:FEMO>
deleteUserExperiment	POST /experiment.erm/rest/experimentservices/deleteUserExperiment?femoID=FEMOID HTTP/1.1 Host: platform.fiesta-iot.eu iPlanetDirectoryPro: TOKEN_FROM_USER_CREDENTIALS Cache-Control: no-cache Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW

saveExperimentServiceModelObject	POST /experiment.erm/rest/experimentservices/saveExperimentServiceModelObject?femoID=FEMOID HTTP/1.1 Host: platform.fiesta-iot.eu iPlanetDirectoryPro: TOKEN_FROM_USER_CREDENTIALS Content-Type: application/xml Cache-Control: no-cache Body: <?xml version="1.0" encoding="UTF-8" standalone="yes"?> <ns2:FISMO name="2ndUseCase"> ...... </ns2:FISMO>
deleteExperimentServiceModelObject	POST /experiment.erm/rest/experimentservices/deleteExperimentServiceModelObject?fismoID=FISMOID HTTP/1.1 Host: platform.fiesta-iot.eu iPlanetDirectoryPro: TOKEN_FROM_USER_CREDENTIALS Cache-Control: no-cache Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW
getALLUserExperiments	GET /experiment.erm/rest/experimentservices/getALLUserExperiments?userID=USERID HTTP/1.1 Host: platform.fiesta-iot.eu iPlanetDirectoryPro: TOKEN_FROM_USER_CREDENTIALS Cache-Control: no-cache
getAllUserExperimentsDescriptions	GET /experiment.erm/rest/experimentservices/getAllUserExperimentsDescriptions?userID=USERID HTTP/1.1 Host: platform.fiesta-iot.eu iPlanetDirectoryPro: TOKEN_FROM_USER_CREDENTIALS Cache-Control: no-cache
getExperiment	GET

Description	/experiment.erm/rest/experimentservices/getExperimentDescription?femoID=FEMOID HTTP/1.1 Host: platform.fiesta-iot.eu iPlanetDirectoryPro: TOKEN_FROM_USER_CREDENTIALS Cache-Control: no-cache
getExperimentModelObject	GET /experiment.erm/rest/experimentservices/getExperimentModelObject?femoID=FEMOID HTTP/1.1 Host: platform.fiesta-iot.eu iPlanetDirectoryPro: TOKEN_FROM_USER_CREDENTIALS Cache-Control: no-cache
getExperimentServiceModelObject	GET /experiment.erm/rest/experimentservices/getExperimentServiceModelObject?fismoID=FISMOID HTTP/1.1 Host: platform.fiesta-iot.eu iPlanetDirectoryPro: TOKEN_FROM_USER_CREDENTIALS Cache-Control: no-cache
getDiscoverableExperimentServiceModelObjects	GET /experiment.erm/rest/experimentservices/getDiscoverableExperimentServiceModelObject HTTP/1.1 Host: platform.fiesta-iot.eu iPlanetDirectoryPro: TOKEN_FROM_USER_CREDENTIALS Cache-Control: no-cache

5.1.1 Object Definition

The FEDSpec XSD, which was described in detail in Section 4.2 above, can be found in Appendix I and more specifically in Table 9 below. In Figure 13 below we can see the ExpDescriptiveIDs schema graph (the XSD is provided in Appendix I below and more specifically in Table 10 below) which consists of:

- A list of **FEMO Descriptive ID**: which is capable of providing a high-level deception of an experiment and It includes:
    - The ID of the experiment
    - The description of the experiment
    - The name of the experiment and

- A list of **FISMO Descriptive ID**: which is capable of providing a high-level deception of a Service Model Object and It includes:
  - The FISMO ID
  - The FISMO description and
  - The FISMO name

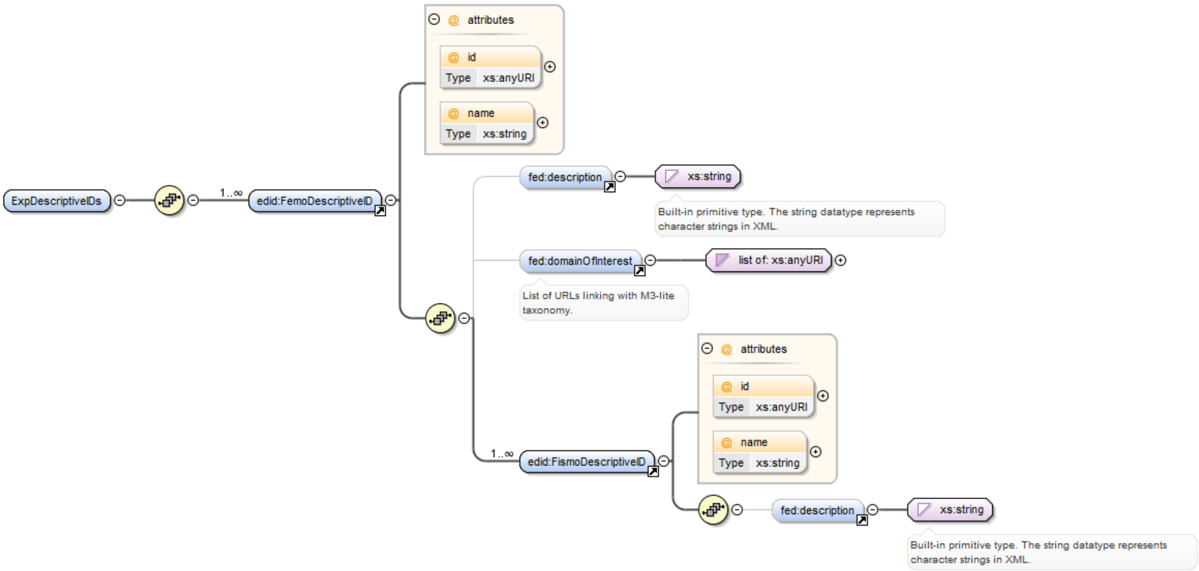


Figure 13. Experiment Descriptive IDs schema graph.

5.2 Exceptions

Methods of the Experiment Registry Management API signal error conditions to the client by means of exceptions. The following exceptions are defined in Table 5. All the exception types in the following table are extensions of a common ExperimentRegistryManagementException base type, which contains one string element giving the reason for the exception.

Table 5 Exceptions associated with the Experiment Registry Management API

Exception Name	Meaning
SecurityException	The operation was not permitted due to an access control violation or other security concern. This includes the case where the service wishes to deny authorization to execute a particular operation based on the authenticated client identity. The specific circumstances that may cause this exception are implementation-specific, and outside the scope of this specification.
FEDSpecValidationException	The Experiment Registry Management failed to successfully validate the FEDSpec comprising the service request. This can be a result of a malformed FEDSpec

	specification.
FemoValidationException	The Experiment Registry Management failed to successfully validate the FEMO comprising the service request. This can be a result of a malformed FEMO specification.
FismoValidationException	The Experiment Registry Management failed to successfully validate the FISMO comprising the service request. This can be a result of a malformed FISMO specification.
NoSuchExperimentID	The Experiment Registry Management failed to identify the Experiment ID i.e. the Experiment ID is not available within the Experiment Registry repository.
NoSuchServiceModelObjectID	The Experiment Registry Management failed to identify the Service Model Object ID i.e. the Service Model Object ID is not available within the Experiment Registry repository.
NoSuchUserID	The Experiment Registry Management failed to identify the User ID i.e. the User ID is not available within the Experiment Registry repository.
ImplementationException	A generic exception raised by the implementation for reasons that are implementation-specific. This exception contains one additional element: a severity member whose values are either ERROR or SEVERE. ERROR indicates that the Experiment Registry Management implementation is left in the same state it had before the operation was attempted. SEVERE indicates that the Experiment Registry Management implementation is left in an indeterminate state.
ResultTooLargeException	An attempt to execute a request resulted in more data than the service was willing to provide.

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

Table 6 Exceptions thrown by the different Experiment Registry Management services

Service Name	Throws
saveUserExperiments	SecurityException FEDSpecValidationException ImplementationException
deleteUserExperiments	SecurityException NoSuchUserID ImplementationException
saveUserExperiment	SecurityException FemoValidationException NoSuchUserID ImplementationException
deleteUserExperiment	SecurityException NoSuchExperimentID ImplementationException
saveUserExperimentServiceModelObject	SecurityException FismoValidationException NoSuchExperimentID ImplementationException
deleteExperimentServiceModelObject	SecurityException NoSuchServiceModelObjectID ImplementationException
getALLUserExperiments	SecurityException NoSuchUserID ImplementationException ResultTooLargeException
getAllUserExperimentsDescriptions	SecurityException NoSuchUserID ImplementationException ResultTooLargeException
getExperimentDescription	SecurityException NoSuchExperimentID ImplementationException ResultTooLargeException
getExperimentModelObject	SecurityException NoSuchExperimentID ImplementationException ResultTooLargeException
getExperimentServiceModelObject	SecurityException NoSuchServiceModelObjectID ImplementationException ResultTooLargeException
getDiscoverableExperimentServiceModelObjects	SecurityException ImplementationException ResultTooLargeException

5.3 Prototype Implementation

In this section, we provide an installation and basic usage manual of the ERM prototype implementation. 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 [1] 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.

5.3.1 Design

The ERM implementation was designed in a way to be able to support future requirements over the User’s experiment definition and discovery. For this reason, we decided to store all the FEDSpec entities into a one to one mapping fashion over a relational database schema which is depicted in Figure 14 below.

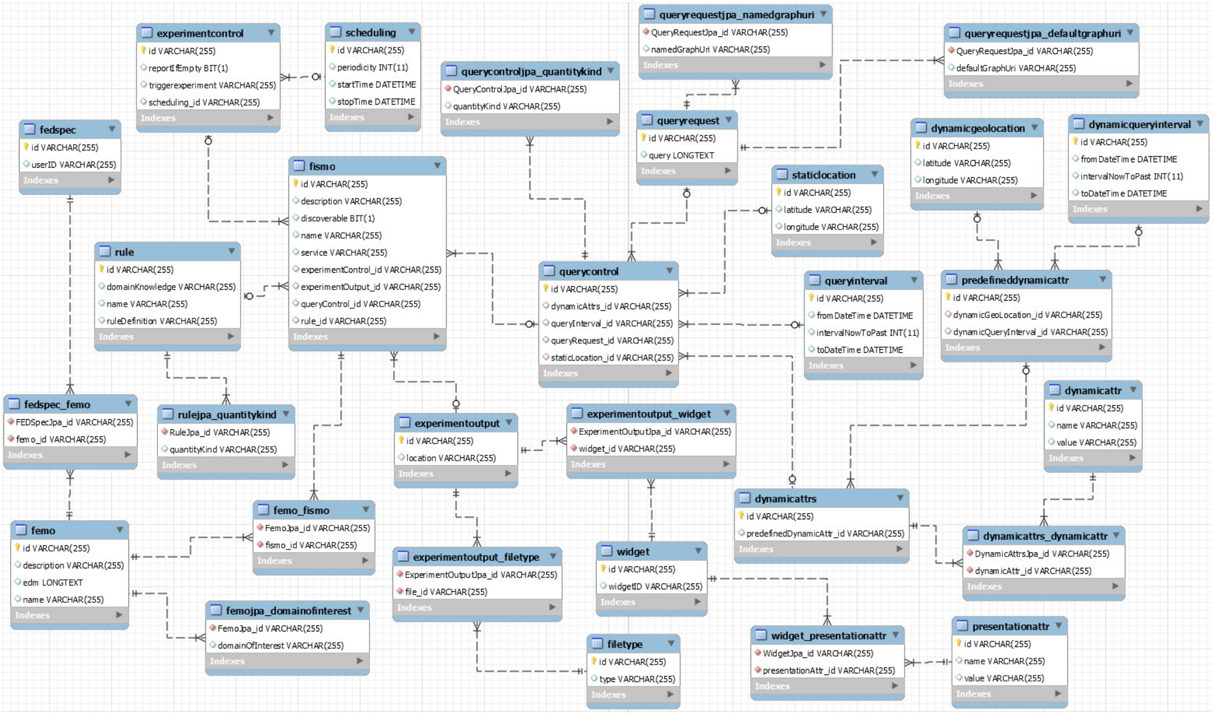


Figure 14 ERM Relational Database Schema.

Moreover this relational database is mapped with the object-oriented domain model of FEDSpec with the help of Hibernate ORM[8].

5.3.2 Source code Availability and Structure

The Experiment Registry Management component is offered at FIESTA-IoT GitLab repository which is named “core” and is available at: https://gitlab.fiesta-

[8] http://hibernate.org/

iot.eu/platform/core/. The latest version of the components is under the "develop branch"[9] .

The repository is organized in 4 categories/folders and the ERM component is placed as follows:

- doc: provides all the related documents with the platform.
- module: provides the core modules of the platform
  - experiment: provides the modules related with the experiments
    - experiment.erm: the Experiment Registry Management module
- utils: provides utilities related with the platform
  - utils.commons: include the common utils/objects used in more than 2 projects/components

5.3.2.1 System Requirements

The ERM component is deployed within a WildFly container and use Maven for 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 prototype are given in the section below.

5.3.2.2 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,

4. make sure that all properties listed in **$PROJECT_HOME/src/main/resources/fiesta-iot.properties** have the appropriate values,
5. copy that file into **$WILDFLY_HOME/standalone/configuration**, and
6. 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.

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

Alternatively copy the produced (from the build process above) **experiment.erm.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]/experiment.erm

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

5.3.2.3 API Usage

The different exposed services described in the ERM API above are exposed at the following URLs:

```
http://[HOST]:[PORT]/experiment.erm/rest/experimentservices/saveUserExperiments
http://[HOST]:[PORT]/experiment.erm/rest/experimentservices/deleteUserExperiments
http://[HOST]:[PORT]/experiment.erm/rest/experimentservices/saveUserExperiment
http://[HOST]:[PORT]/experiment.erm/rest/experimentservices/deleteUserExperiment
http://[HOST]:[PORT]/experiment.erm/rest/experimentservices/saveExperimentServiceModelObject
http://[HOST]:[PORT]/experiment.erm/rest/experimentservices/deleteExperimentServiceModelObject
http://[HOST]:[PORT]/experiment.erm/rest/experimentservices/getALLUserExperiments
http://[HOST]:[PORT]/experiment.erm/rest/experimentservices/getAllUserExperimentsDescriptions
http://[HOST]:[PORT]/experiment.erm/rest/experimentservices/getExperimentDescription
http://[HOST]:[PORT]/experiment.erm/rest/experimentservices/getExperimentModelObject
http://[HOST]:[PORT]/experiment.erm/rest/experimentservices/getExperimentServiceModelObject
```

5.3.2.4 Containers and Libraries

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

Table 7 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
fiesta-commons	0.0.1
logback	1.0.11
mysql-connector	5.1.39
resteasy	3.0.19.Final
hibernate	5.0.7.Final

[10] http://wildfly.org/

5.4 ERM UI Client

5.4.1 Register a new experiment

FIESTA-IoT is currently offering a simple interface in order to Store, Update and delete experiments called Experiment Register Client. The Experiment Register Client can be found at the Experimenter menu of the FIESTA-IoT portal (see Figure 15 below).

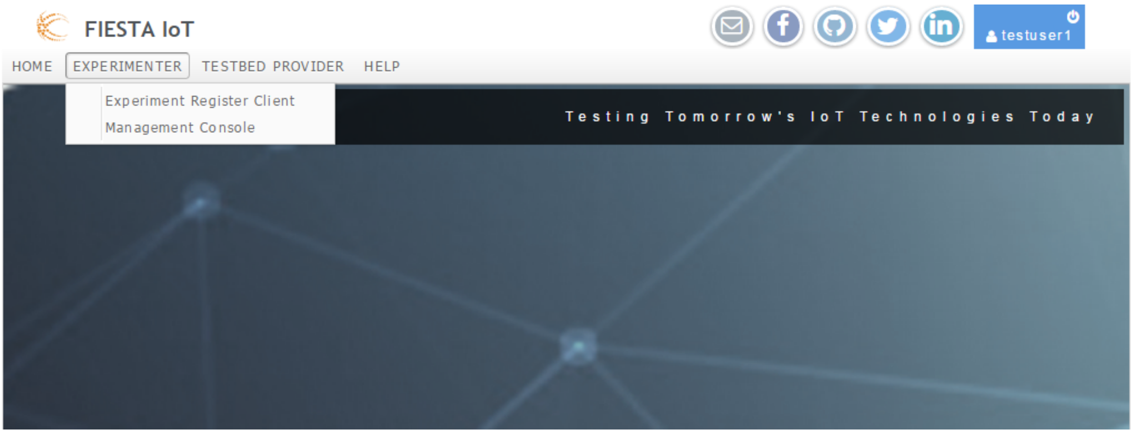


Figure 15. Portal Experimenter Menu

The Experiment Register Client provides the ability to store an experiment at the FIESTA-IoT platform in the form of a FEDSpec. The defined FEDSpec could be as simple as a single service (FISMO) or as complex as multiple experiments (FEMOs). To upload a FEDSpec first one should identify the location of it by hitting the “Open FEDSpec” (see Figure 16 below) and then by hitting the “Save FEDSpec” button. As soon as the FEDSpec is saved the included FEMOS appears in the available experiments list (FEMOS) as shown in Figure 16 below. When uploading a FEDSpec the FEMO/FISMO IDs should be empty, as they will be automatically assigned by the system.

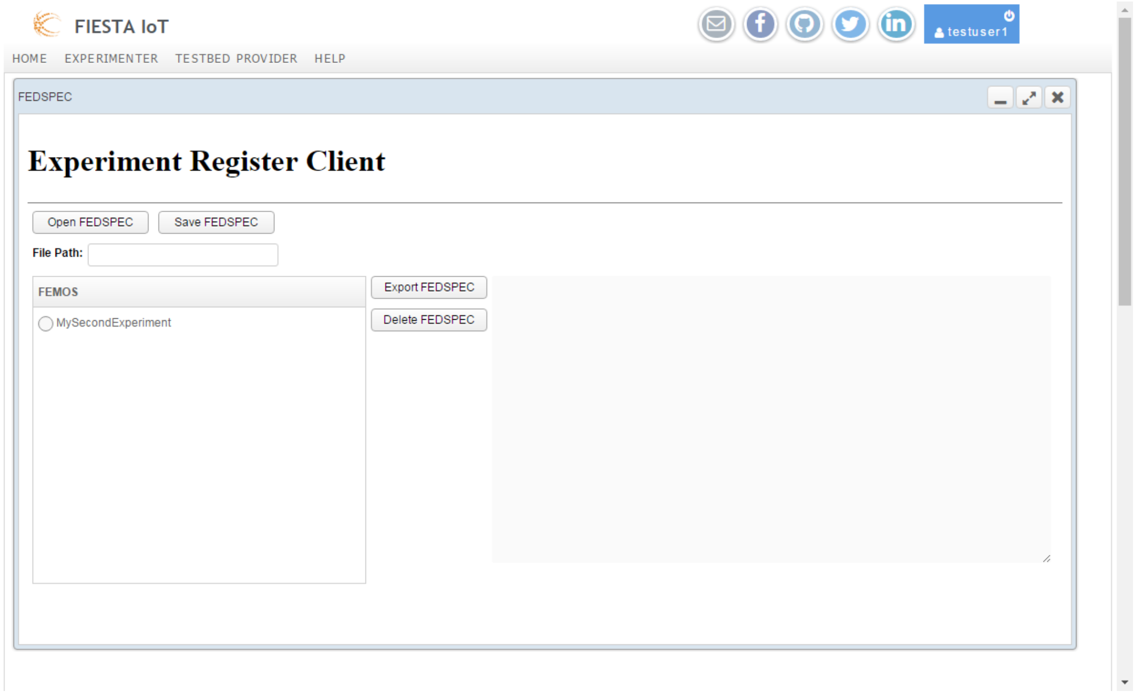
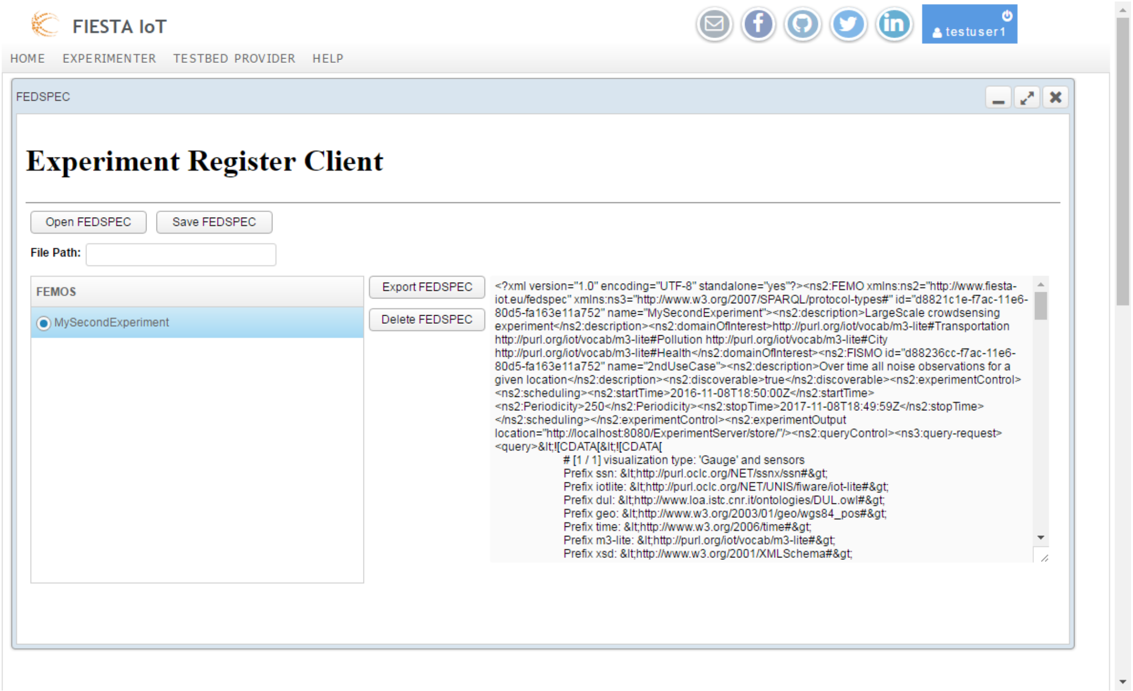


Figure 16. Experiment Register Client

The User, by choosing a FEMO, is capable to have a quick overview of it as shown in Figure 17 below.



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><ns2:FEMO xmlns:ns2="http://www.fiesta-
iot.eu/fedspec" xmlns:ns3="http://www.w3.org/2007/SPARQL/protocol-types#" id="d8821c1e-f7ac-11e6-
80d5-fa163e11a752" name="MySecondExperiment"><ns2:description>LargeScale crowdsensing
experiment</ns2:description><ns2:domainOfInterest>http://purl.org/iot/vocab/m3-lite#Transportation
http://purl.org/iot/vocab/m3-lite#Pollution http://purl.org/iot/vocab/m3-lite#City
http://purl.org/iot/vocab/m3-lite#Health</ns2:domainOfInterest><ns2:FISMO id="d88236cc-f7ac-11e6-
80d5-fa163e11a752" name="2ndUseCase"><ns2:description>Over time all noise observations for a
given location</ns2:description><ns2:discoverable>true</ns2:discoverable><ns2:experimentControl>
<ns2:scheduling><ns2:startTime>2016-11-08T18:50:00Z</ns2:startTime>
<ns2:Periodicity>250</ns2:Periodicity><ns2:stopTime>2017-11-08T18:49:59Z</ns2:stopTime>
</ns2:scheduling></ns2:experimentControl><ns2:experimentOutput
location="http://localhost:8080/ExperimentServer/store/"/><ns2:queryControl><ns3:query-request>
<query>&lt;![CDATA[&lt;![CDATA[
                    # [1 / 1] visualization type: 'Gauge' and sensors
                    Prefix ssn: &lt;http://purl.oclc.org/NET/ssnx/ssn#&gt;
                    Prefix iotlite: &lt;http://purl.oclc.org/NET/UNIS/fiware/iot-lite#&gt;
                    Prefix dul: &lt;http://www.loa.istc.cnr.it/ontologies/DUL.owl#&gt;
                    Prefix geo: &lt;http://www.w3.org/2003/01/geo/wgs84_pos#&gt;
                    Prefix time: &lt;http://www.w3.org/2006/time#&gt;
                    Prefix m3-lite: &lt;http://purl.org/iot/vocab/m3-lite#&gt;
                    Prefix xsd: &lt;http://www.w3.org/2001/XMLSchema#&gt;
```

Figure 17. Experiment Register Client - Experiment Browser

The tools provide also the ability to export a FEMO by hitting the “Export FEDSPEC” button after choosing the FEMO of interest from the provided list. The FEDSpec that will be exported will now contain the FEMO/FISMO IDs assigned from the FIESTA-IoT platform. This will give the Experimenter the ability to update the exported FEMO/FISMO by updating the XML file and saving it again to the Experiment Repository following the same process described above.

6 EXPERIMENT MODELLING ENGINE (EME)

Experiment Modelling Engine is a visual programming tool provided by FIESTA-IoT for the ease of experiment modelling to users that came from non-programming background. This is achieved by using one of the established visual programming tool Node-RED.

6.1 Node RED

Node-RED is a programming tool for wiring together hardware devices, APIs and online service in new and interesting ways. It provides a browser-based editor that makes it easy to wire together flows using the wide range of nodes in the palette that can be deployed to its runtime in a single-click.

FIESTA-IoT customized Node-RED called Experiment Modelling Engine (EME) is used to perform experiment storage, retrieval, discovery and termination by using the visual programming nature of the Node-RED. The Experiment Modelling Engine comes with four basic FIESTA-IoT Nodes

1. Service Node
2. FEMO Node
3. FISMO Node
4. Query Control Node

Each **Service Node** can perform one of the twelve operations on FEDSpec schema which can be categorized into either storage, retrieval, discovery or termination (refer to section 5 - Experiment Registry Management, Table 3 - Experiment Registry Management API definition). A Service Node can take input from two kinds of Nodes; FEMO Node and FISMO Node. A Service Node taking input from the FEMO Node receives a FEMO template and when taking input from the FISMO Node, the Service Node receives FISMO template. The Service Node can be configured to perform selected operation by taking direct input at the node or by taking input from the wired node as shown in Figure 18 below.

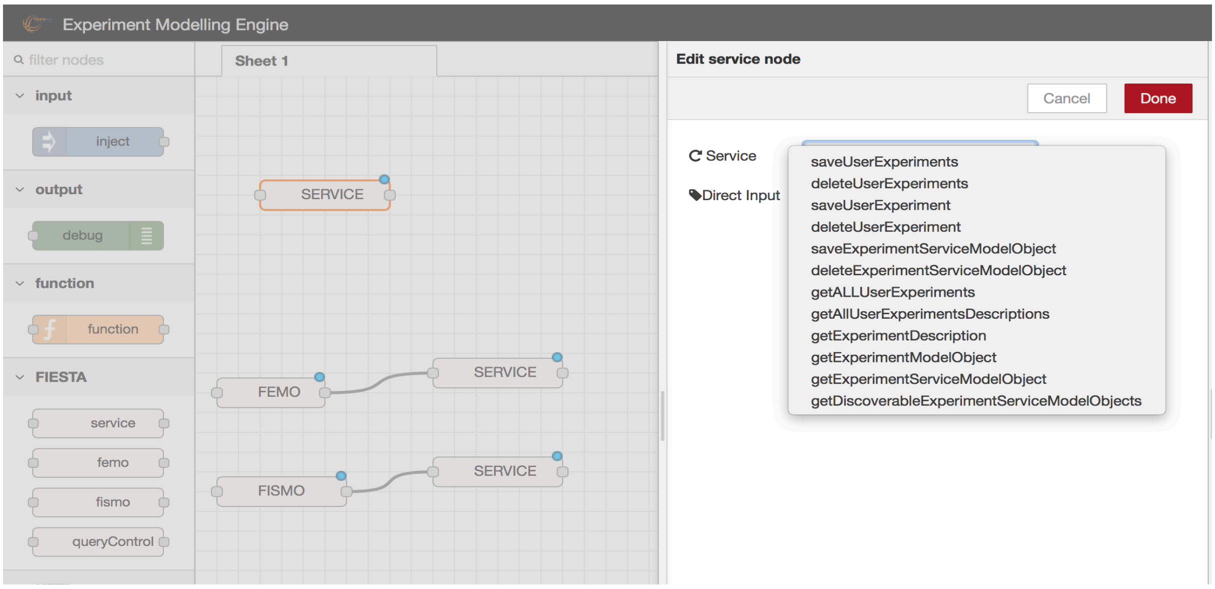
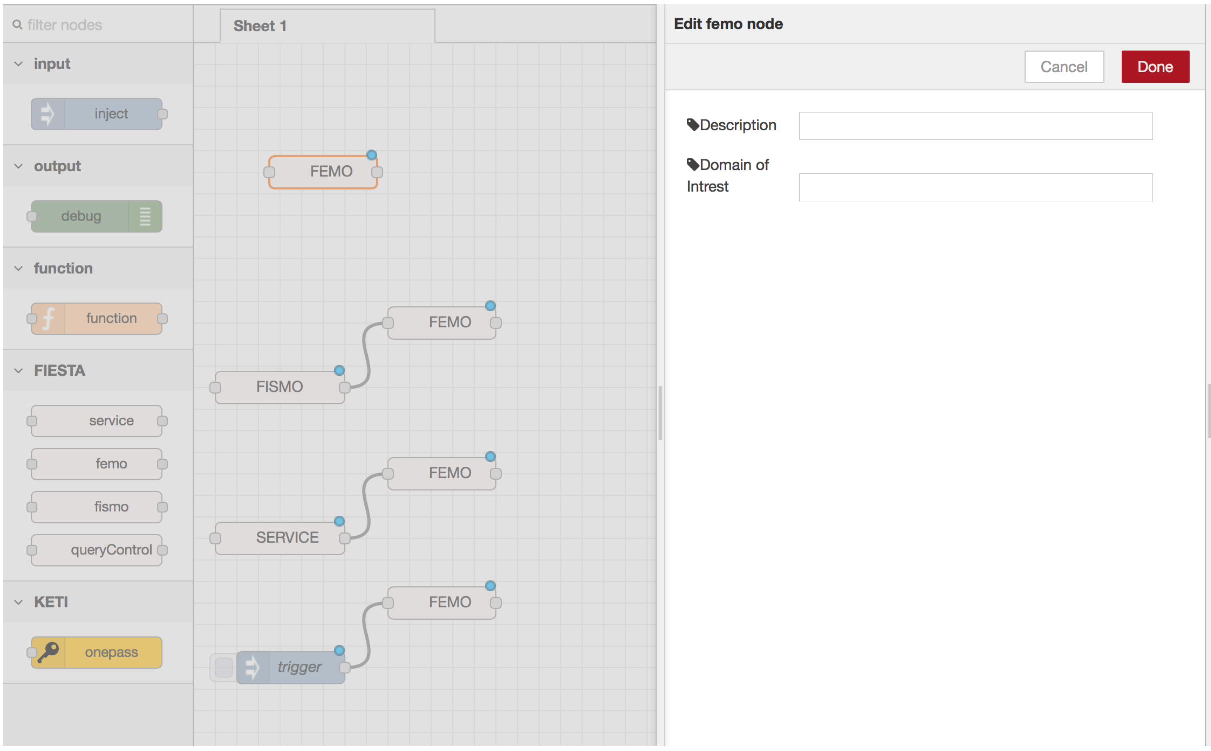


Figure 18. Service Node

The **FEMO Node** is a template node which generates FEMO template as output. The FEMO Node can take input from three kinds of nodes; Service Node, FISMO Node and any other kind of node. A FEMO Node receiving input from the Service Node will assign a FEMO ID to the FEMO template that is going to be generated (note: any FEMO that is generated without ID will be considered as a New FEMO Object). A FEMO Node receiving input from the FISMO Node will generate a FEMO template which has a FISMO template embedded in it. A FEMO Node receiving input from any other kind of node will simply trigger the template generation (note: a FEMO node that is not connected to FISMO will always generate an Empty FEMO Object). The FEMO Node should be configured by the user to provide information such as description, domain of interest (refer to section 4.2 - FESTA-IoT Experiment Model Object). The following Figure 19 below depicts the possible input connections of FEMO Node.

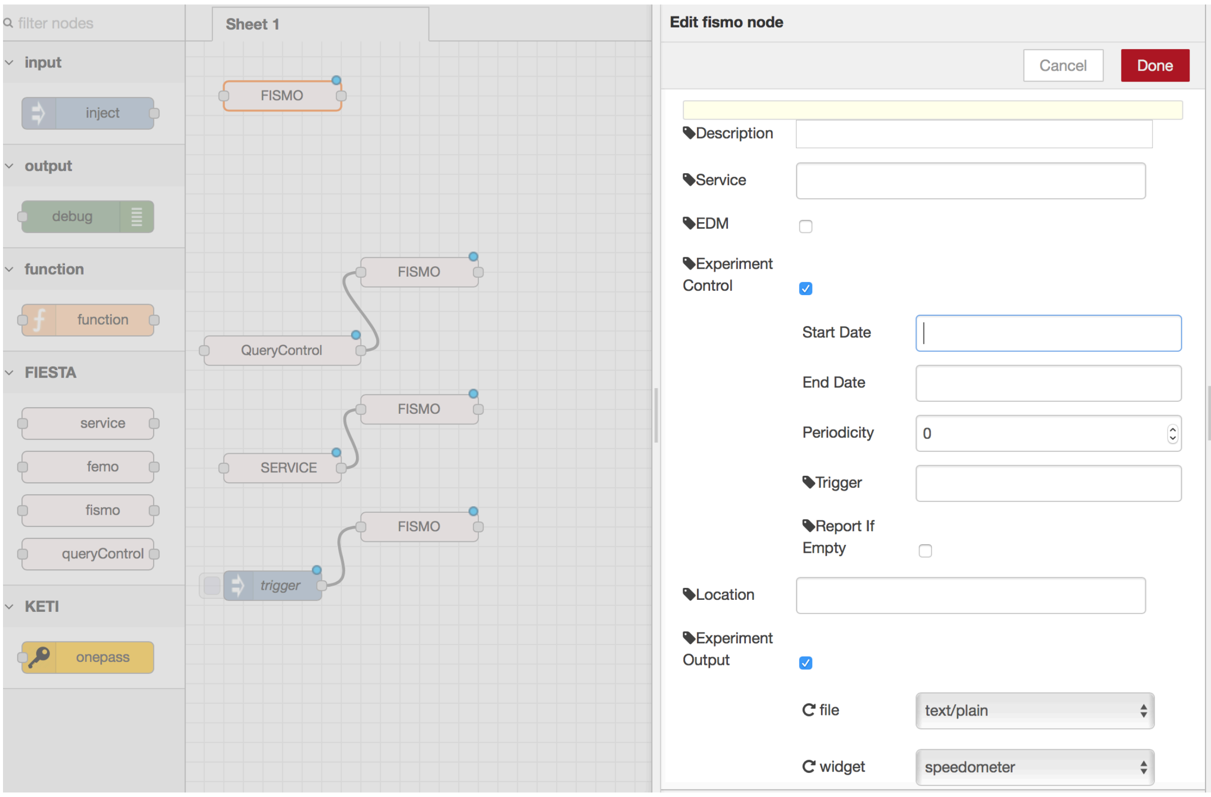


Description

Domain of Intrest

Figure 19. FEMO Node

The **FISMO Node** is a templating node which generates FISMO template as output. The FISMO Node can take input from three kinds of nodes; Service Node, Query Control Node and any other kind of node. A FISMO Node receiving input from the Service Node will assign a FISMO ID to the FISMO template that is going to be generated (note: any FISMO that is generated without ID will be considered as a New FISMO Object). A FISMO Node receiving input from the Query Control Node will generate a FISMO template which has a Query Control template embedded in it. A FISMO Node receiving input from any other kind of node will simple trigger the templet generation (note: a FISMO node that is not connected to Query Control Node will always generate a default ‘fed:queryControl’ for the FISMO Object). The FISMO Node should be configured by the user to provide information such as Description, Service, EDM, Experiment Control (optional), Location and Experiment Output (optional) (refer to section 4.2 - FESTA-IoT Experiment Model Object). Figure 20 below depicts the possible input connections of FISMO Node.



Edit fismo node

Description

Service

EDM □

Experiment Control ☑

Start Date

End Date

Periodicity 0

Trigger

Report If Empty □

Location

Experiment Output ☑

file text/plain

widget speedometer

Figure 20. FISMO Node

The Query Control Node is a template node which generates 'fed:queryControl' template. The Query Control Node can be used with any kind of node which would simple trigger the generation of 'fed:queryControl' template. The Query Control Node should be configured by the user to provide information such as Quantity Kind, Static Location, Query Interval, and Query Request (refer to section 4.2 - FESTA-IoT Experiment Model Object).

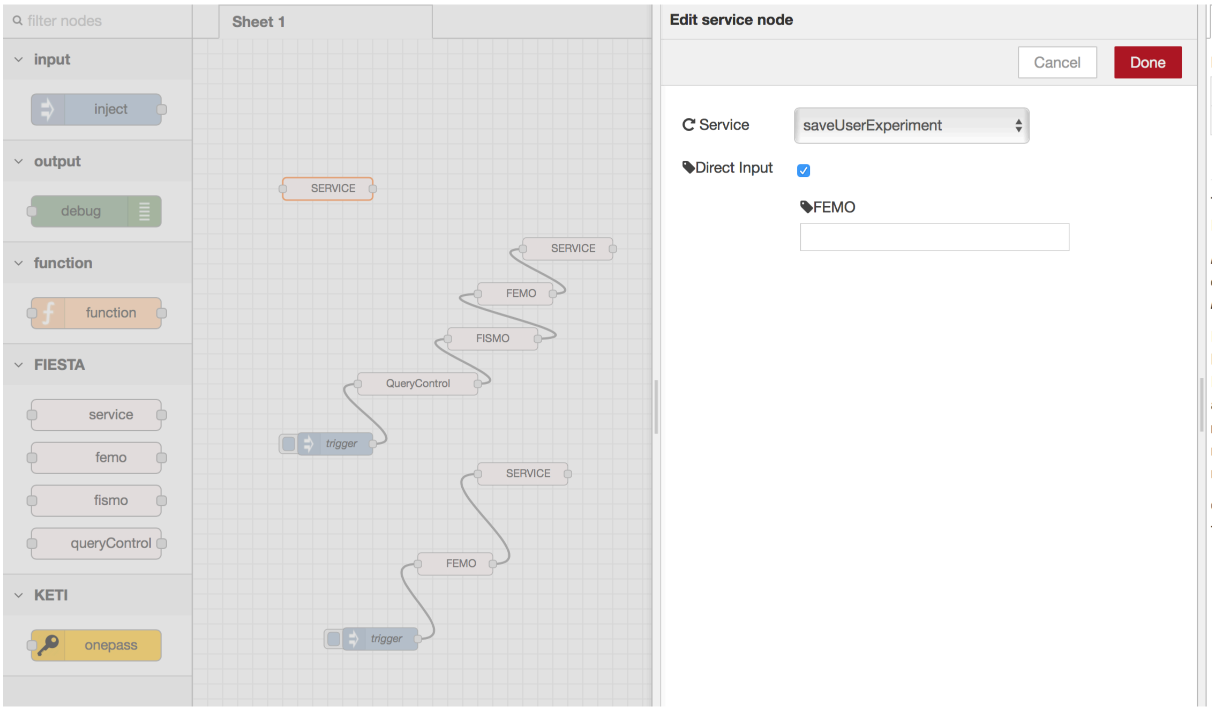
6.2 Experiment Operation Flows

Using Service Node or combinations of Service Node, FEMO Node, FISMO Node and Query Control Node we can perform twelve different operations on Experiment by manipulating the objects to comply with in the FEDSpec schema, some of these operation flows are described below.

'**Save User Experiments**' is one of the operation available by using Service Node. This operation can save a FEDSpec by taking FEDSpec as input at the configuration of the node. To perform 'Save User Experiment' just Service Node and any kind of trigger node is enough.

'**Delete User Experiments**' operation of Service Node deletes all user experiments and to create this flow, only Service Node and any kind of trigger node is enough.

'**Save User Experiment**' operation can be performed in multiple ways, one of them is by providing a FEMO Object at the Service Node we can perform this operation by using just the Service Node and any kind of trigger node. We can also perform this operation with the combinations of Service Node – FEMO Node – FISMO Node – Query Control Node – trigger node or Service Node – FEMO Node – FISMO Node or just Service Node – FEMO Node. These combinations should always end with the result of saving an experiment with the difference of having FEMO with embedded FISMO or an empty FEMO (i.e. FEMO without FISMO). Figure 21 shows the possible flows of the ‘Save User Experiment’.



Edit service node

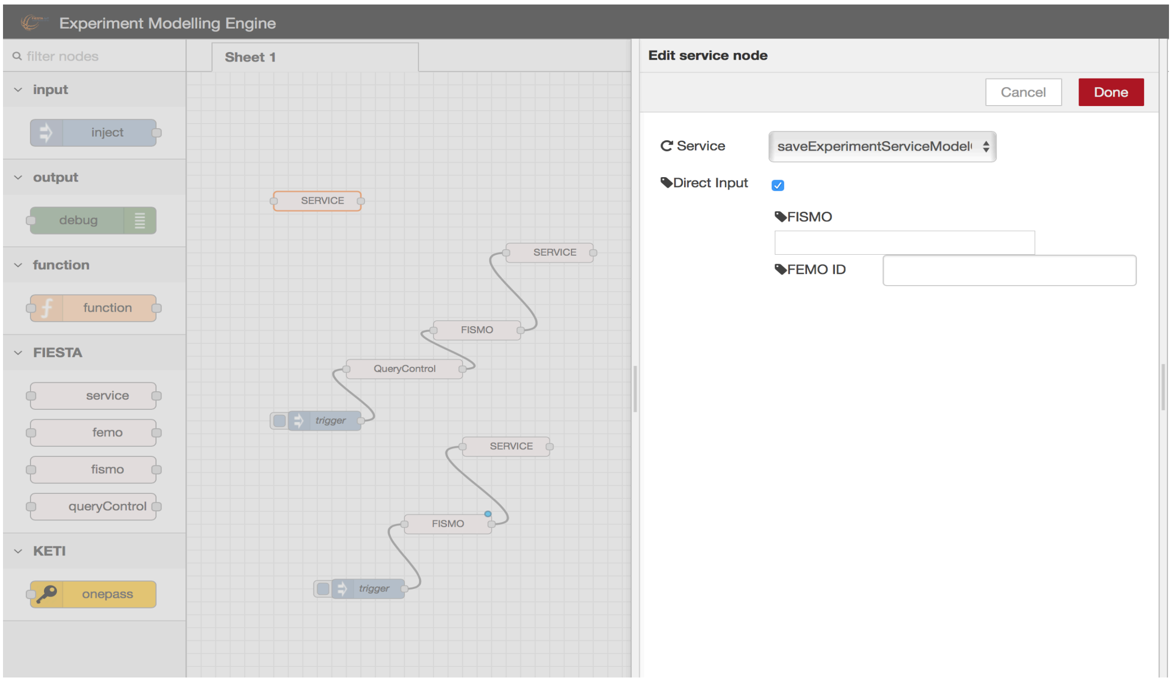
Service: saveUserExperiment

Direct Input ☑

FEMO

Figure 21. Save User Experiment Flows

‘**Save Experiment Service Model Object**’ operation can be performed in multiple ways, one of them is by providing a FISMO Object at the Service Node we can perform this operation by using just Service Node and any kind of trigger node. We can also perform this operation with the combination of Service Node – FISMO Node – Query Control Node –trigger node or Service Node – FISMO Node.



Edit service node

Service: saveExperimentServiceModel

Direct Input ☑

FISMO

FEMO ID

Figure 22. Save Experiment Service Model Object Flows

These combinations would always end with the result of saving an experiment service model object with the difference of having either the user defined or pre-determined 'fed:queryControl'. The Figure 22 above shows the possible flows of the 'Save Experiment Service Model Object'.

The Table 8 below provides the possible flows of the remaining operations. The nodes are connected in the order they are specified (the first specified nodes output is wired to the next specified node's input).

Table 8 Flows for Service Node's Services

Service Name	Input	Output	Flow
Delete User Experiment	FEMO-ID	Success message + FEMO-ID	Trigger Node – Service Node
Delete Experiment Service Model Object	FISMO-ID	Success message + FISMO-ID	Trigger Node – Service Node
Get All User Experiments	—	All Experiments	Trigger Node – Service Node
Get All User Experiments Descriptions	—	Experiment Descriptive Ids	Trigger Node – Service Node
Get Experiment Description	FEMO-ID	FEMO Descriptive ID	Trigger Node – Service Node
Get Experiment Model Object	FEMO-ID	FEMO Object	Trigger Node – Service Node
Get Experiment Service Model Object	FISMO-ID	FISMO Object	Trigger Node – Service Node
Get Discoverable Experiment Service Model Objects	—	List of FISMO	Trigger Node – Service Node

6.3 Installation

Node RED requires 'node.js' to be installed in the target machine. User can get the latest Long Term Support (LTS) version of Node 6.x from:

- Mac OS X Installer: Universal
- Windows Installer: 32-bit or 64-bit
- Linux Binaries: 32-bit or 64-bit

The easiest way to install Node-RED is to use node's package manager, npm. Installing it as a global module adds the command node-red to your system path:

```
$sudo npm install -g --unsafe-perm node-red
```

Once installed, you are ready to run Node-RED using the following command

```
$node-red
```

7 EXPERIMENT WORKFLOW

In this section, we first provide an updated version of the experiment workflow. Then we provide the integration details and the workflow of FIESTA-IoT Analytics Toolkit (FAT) component and EEE. Thereafter, we discuss the Living Lab Experiment which follows the experiment workflow to discover resources, define and execute experiment, and retrieve results from the FIESTA-IoT Meta-Cloud.

7.1 Experiment Workflow

Experiment Workflow is the process followed by an experimenter from the very moment he/she starts building his/her own experiment to the instant the first batch of results are gathered (after that, we can assume that the thread is an infinite loop). Figure 23 below depicts the different stages that are to be carried out in order to build a fully-fledged experiment/application/service. Below we proceed to delve into the details of each of the different phases that will shape an experiment, including a "pre-experiment" step that addresses the establishment of a secure channel between experimenters and FIESTA-IoT.

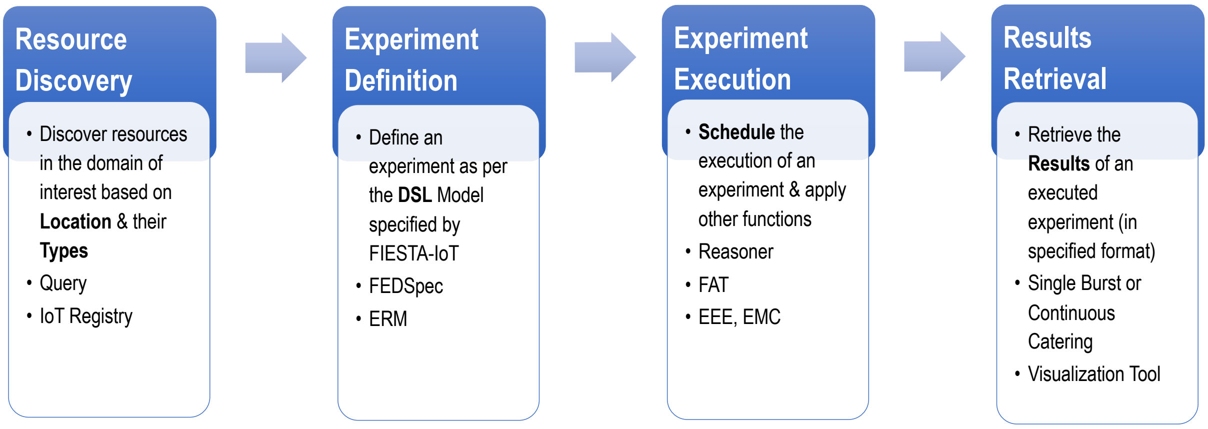

Resource Discovery
• Discover resources in the domain of interest based on **Location** & their **Types**
• Query
• IoT Registry
Experiment Definition
• Define an experiment as per the **DSL** Model specified by FIESTA-IoT
• FEDSpec
• ERM
Experiment Execution
• **Schedule** the execution of an experiment & apply other functions
• Reasoner
• FAT
• EEE, EMC
Results Retrieval
• Retrieve the **Results** of an executed experiment (in specified format)
• Single Burst or Continuous Catering
• Visualization Tool

Figure 23. Experiment Workflow

- **PHASE 0 "Experimenter registration"**. Aside the specification and definition of the experiment per se, external users must sign up with the FIESTA-IoT platform prior to extract data from the different underlying testbeds that compose the federation. Hence, the very first step to be done is to get registered as an external user, following a triple A (Authentication, Authorization and Accounting – AAA) service, whose outcome is gathered from the following steps:
  - Identify the user, differentiating between different roles: experimenter, testbed provider (raw-data producer), knowledge producer or value-added service provider. For a deeper classification of this taxonomy, the reader might refer to D2.4 [3], where we describe the *OpenAM*-based solution that will be in charge of protecting the communications between FIESTA-IoT and outsiders (in a bidirectional way).
  - Establish the ownership and bindings between experimenters and their respective experiments. Together with this linkage, other experimenters will be able to browse/discover among the off-the-shelf experimenters that have been previously registered into the platform. This way, as was mentioned in previous sections, experimenters might decide not to start

from scratch but from some baselines to rely on. As a note, some of the services to discover these experiments are described in Deliverable D4.7 [10].

  - Exchange the set of credentials (i.e. single sign-on token) to guarantee a protected and secure communication between experimenters (or testbeds) and the FIESTA-IoT platform (all the information regarding the security framework chosen for this project can be found in D4.3 [8]. Once the experiment has been authenticated and authorized, experimenters can proceed to carry out the following steps.

- **PHASE 1 "Domain and resource discovery"**. Despite the vast plethora of domains that are embraced by the FIESTA-IoT federation and its subjacent platforms (which will increase with new testbeds that are fostered via the open calls), experimenters usually focus on a single (or few) particular area or "domain of interest", thus filtering out all resources that are actually out of their scope. In this initial "discovery" stage, experimenters will retrieve a list (i.e. semantic annotated description) that holds all the resources/Virtual Entities/IoT Services that match the requirements specified for this stage. Typically, these searches will be based upon domain, location or physical phenomena-based queries.

- **PHASE 2 "Experiment definition"**. Once experimenters have selected their domain of interest and have framed the location and physical phenomena that they are interested in, it is time to go a step beyond and start defining the actual characteristics that will model the experiment *per se*, using the FEDSpec. This part defines how to deal with the information generated by these resources/Virtual Entities/IoT Services. To do this, there is a number of different offered possibilities to retrieve the data [3]. We slightly outline the mainstream ones below:

  - *Service invocation (≈ synchronous service)*. The first and most straightforward option to get data is through the invocation of the services that expose the resources (or the Virtual Entities' properties). These are stored as part of their respective annotated descriptions (see the FIESTA-IoT ontology defined in D3.1 [4]. Amongst the available services, experimenters will be able to e.g. retrieve the last value observed by a particular sensing device. It goes without saying that we plan to append more services in the future.

  - *Historical values.* Another possibility of getting information already captured in the FIESTA-IoT Meta-Cloud is the request of historical information; i.e. by manually specifying a time window that limits the range of data (e.g. for the sake of simplicity, and using a natural language equivalent query: "give me all what you have captured yesterday between noon and 7pm").

  - *Subscriptions (≈ asynchronous service).* The main drawback of the first two options is that they can only grasp information that has been already taken. Nonetheless, none of them do not directly contemplate the reception of future information. As a straightforward solution, one can schedule synchronous queries in order to periodically receive the requested data. However, it would be more sensible to support a "pub-sub" like mechanism that implements an efficient way of handling and dispatching all the asynchronously events received from the testbeds. So, the FIESTA-IoT platform will forward and hand the data only to the appropriate subscribers, thus the system performance would be notably enhanced.

Apart from the retrieval of the information, FIESTA-IoT will pay special attention to a number of additional features that operates above this raw data. Techniques like reasoning, complex events processing or composing of IoT services, to list a few, are envisaged to be part of the platform in the near future. A further description of this core part of the architecture can be found in Deliverable D3.6 [6].

Last, but not least, it is deemed necessary to elicit the execution schedule that will define the experiment's operation. Or said in other words, to specify when and how the experiment will run, including the way that the results will be presented to users (see Phase 4). Concerning this schedule, a clearer description is located in Deliverable D4.7 [10].

- **PHASE 3 "Experiment execution"**. Based on a pseudo-infinite loop behaviour (managed by the execution schedule defined in the previous phase), the application will be in charge of retrieving all the information (i.e. annotated data) that comes from the testbeds across the FIESTA-IoT Meta-Cloud. It is worth highlighting that this phase might contain the scheduled delivery of e.g. SPARQL queries to dynamically look for new resources deployed in the underlying platforms, alter the reasoning rules previously set during the experiment definition from the output of another parallel machine-learning engine, etc. Moreover, we must take into account that experiments' operation times are controlled by the execution schedule generated in the previous stage. To put an example on the table, there might be the case of experiments that are only executed once per week (e.g. on Monday noon), harvest all the data in a single burst, process it and yield the results, thus been switched off till the next week.
- **PHASE 4 "Results retrieval"**. After these definition, specification and further execution processes, the experiment will be ready to achieve results (either at a single burst or through the continuous catering of data). Depending on experimenters' needs or skills, they might be interesting in handling results in two different ways. On the one hand, they might have a raw-text-based shape (e.g. *CSV*, *XML*, *JSON* or whichever *RDF* serialization format if they focus on semantic descriptions), so that they can use their own analysis tools to process the results obtained (at experimentation level, that is, outside the FIESTA-IoT "influence area"). On the other hand, the FIESTA-IoT platform will support the usage of a number of visualization tools or widgets, thus facilitating the life to those experimenters who either do not have the technical expertise to deal with graphical interfaces or have enough with the off-the-shelf elements provided by the platform (or other added-value service providers who want to share their work with the project).

7.2 FIESTA-IoT Analytics toolkit Integration with EEE.

The FIESTA-IoT Analytics Toolkit (FAT) component is a data analysis tool that provides web service for experimenters to do basic analysis. In particular, the experimenter defines the input that has specific JSON format (see **Error! Reference source not found.** below). The complete JSON object contains a sequence of methods, the corresponding parameters and the data source where the data is available in our case the data source is a query. The result for the execution of a FAT is a CSV containing analyzed results. For complete description of the input, output, error format, we refer the readers to the deliverable 3.6 [6].

FAT has been integrated in the FIESTA-IoT Architecture. To invoke the services offered by FAT, experimenters have to follow specific guidelines set by FAT and EEE. These are provided next.

Currently, FAT has 3 different modes of invocation:

1. Invoking it as an independent service using UI
2. Invoking it as an independent service using API
3. Via EEE

```
{
 "Method": ["Method 1"," Method 2"," Method 3"],
 "Parameters": ["Parameters 1", " Parameters 2", " Parameters 3"],
 "SPARQLquery":"SPARQL request string"
 "SPARQLendpoint":"SPARQL endpoint"
}
```

Figure 24. Request Body JSON Object

Within the scope of this deliverable, we provide the workflow to be followed by the experimenters when FAT is invoked via EEE. In the FIESTA-IoT architecture, EEE interacts with IoT-Registry in two ways:

- By directly interacting with the IoT-Registry by sending the queries defined in the service object and
- By sending queries to FAT where FAT acts as a mediator between EEE and IoT-Registry.

In order to invoke FAT services, experimenters first need to set `"value"` parameter in the `"presentationAttr"` of the `"widget"` attribute in the service object of FEDSpec. This value should be a JSON object in a string format. A sample `"widget"` attribute is shown below:

```
<fed:widget widgetID="#FAT_ID_UNIQUE_ALWAYS_FIEXED#">
      <fed:presentationAttr name="requestBody" value="#VALUE#"/>
</fed:widget>
```

Here `#VALUE#` is as mentioned in **Error! Reference source not found.** above. However, the experimenters only provide “Method” and “Parameter” fields. Once this is set, the experimenter provides the query in the `"query"` attribute of the service object in the FEDSpec. This query should follow specifications set by FAT. A template sample query is provided below:

```
Prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
Prefix iotlite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#>
Prefix dul: <http://www.loa.istc.cnr.it/ontologies/DUL.owl#>
Prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
Prefix time: <http://www.w3.org/2006/time#>
Prefix m3-lite: <http://purl.org/iot/vocab/m3-lite#>
Prefix xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Select distinct ?sensingDevice ?dataValue ?dateTime
Where {
…
}
```

Note the Select statement. As per the guidelines set by FAT, the select statement should not be modified.

The EEE then creates the request body by appending the "DataProvider" field that has the specified query and sends it to the FAT. The FAT upon receiving the query forwards the query to the IoT-Registry and sets in the request header "accept=text/csv", "userID=#ExperimenterID#" and "fismoID=#FISMOID#". Note that the =#ExperimenterID# here is the userID of the experimenter. Upon validation of the query and other related parameters, FAT performs two tasks: (1) sends a response to the EEE saying the process has begun and (2) sends the query to the IoT-Registry. FAT then receive the result in CSV format from IoT-Registry. The FAT then performs analysis and stores the results in the FIESTA-IoT Experiment Result Storage (FERS) module. The Experimenter now has to constantly check for the availability of the results in the FERS as FAT operates in an asynchronous mode. The Experimenter is provided with a unique API from the FERS using which the Experimenter downloads all the related results

7.3 Living Lab Experiments

The living lab process, as outlined by Pallot [12], involves a cycle of co-creation, exploration, experimentation and evaluation of innovative ideas, scenarios, concepts and related technological artefacts in real-life use cases, as shown in Figure 25 below.

A living lab can be different from a testbed, depending on the role of the User. In a living lab, the User contributes to the co-creation and exploration of new ideas, and defining new scenarios for testing. This is then fed to experimenters who define and recursively refine a hypothesis that is to be tested, and then evaluate based on the results, and which is then fed back to the users for further refining, or divert using a different approach.

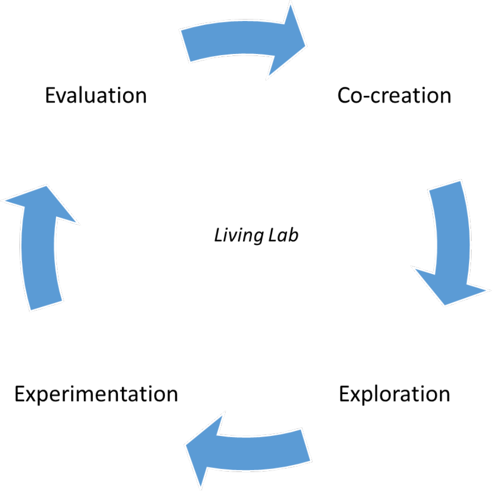


Figure 25. Living Lab Methodology Cycle

As illustrated by Nati et al [11], the typical execution of an experiment with an IoT testbed (e.g. SmartICS) involves several stakeholders. A substantial part of the overall work required to carry out an experiment is related to interactions between these stakeholders. A simple experiment stakeholder model is defined in Figure 26 below. The key assumption is that the experiment is external, i.e., carried out in the Experimentation-as-a-Service (EaaS) mode, as is the case within the FIESTA-IoT ecosystem.

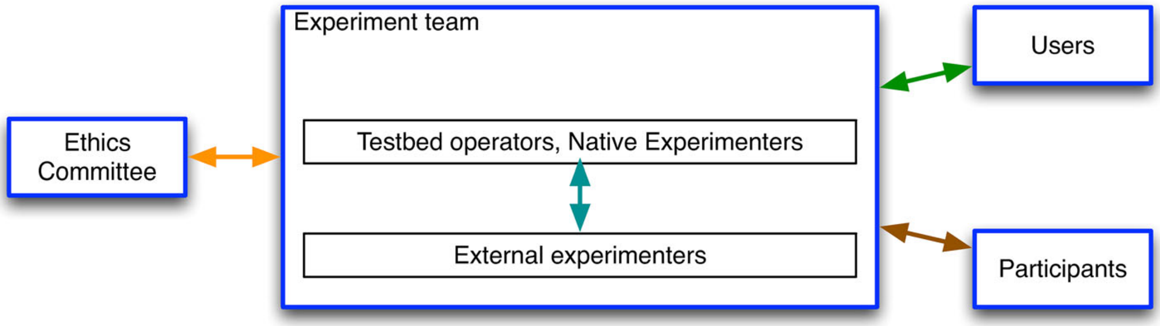


Figure 26. Experiment Stakeholder Model

The stakeholders in this model are:

- *Testbed Operators*: who are the people who have developed the IoT testbed and maintain its daily operation. They have the definitive technical expertise on how the testbed operates and how to interact with it.
- *External Experimenters*: who are the clients of the experimentation service. They have envisioned a specific experiment, seeking to attain results from, and eventually conduct its execution. Prior to the experiment, their knowledge of the capabilities of a testbed is limited.
- The *Users* are all people who can potentially be affected by experimental applications running on the testbed and/or whose behaviour can be observed by such applications. For example, in the case of SmartICS, a user is anyone whose office desk has been fixed with an IoT node. The experiment team

approach users to seek their approval with the installation, and participation in the experiment for feedback.

It is preferable that *External Experimenters* interact with testbed operators. Thus, some of the testbed operators (who are researchers themselves) may get involved in the research dimension of the experiment, by catering to or customizing certain requirements for the External Experimenter. In this case, they become known as *Native Experimenters*.

In the context of FIESTA-IoT, the role of the experimenter (as the *External Experimenter*) within the cycle will be to conduct experiments with the data originating from the living lab, and evaluating the datasets against different criteria of interest. For example, experimenters can define experiments for a domain, and test against another instance of the same or different domain, to evaluate the reusability of the experiment. They could also provide feedback to the testbed providers on several aspects, including data quality, additional data sources for the relevant domain, performance analysis of a testbed. For domain specialists, they can provide feedback on requirements for improving the living lab experience.

Examples of experiments for *External Experimenters* include (but not limited to):

- Domain-specific analysis (buildings, transportation, environmental, agricultural etc.)
- Multi-domain data analysis
- Dataset quality analysis
- system performance analysis

The type of workflows for experiments that can be conducted for living labs can be classified as below:

- Experimenter (Alert) – In the living lab environment an experimenter may wish to design an experiment that periodically sends alerts. Such an experimenter will seek to monitor the living lab environment, and by using a reasoning system (or similar data processing systems) can receive alerts to activities of interest. For example, such an experimenter may design an experiment monitoring the total power usage in the living lab environment, where if the total power exceeds a certain threshold, results in a notification to the User. An example of the workflow is shown in Figure 27 below.
- Experimenter (Hypothesis) – In the living lab environment an experimenter may wish to design an experiment that analyses historical data to test a hypothesis. Such an experimenter will design a question/hypothesis that would require historical data analysis tools and methods to obtain an answer. Examples of such questions/hypotheses include the following: an experimenter may ask the question is there relationship between the office temperature and activity levels (such levels will be captured by office power consumption). An example of the workflow is shown in Figure 28 below.

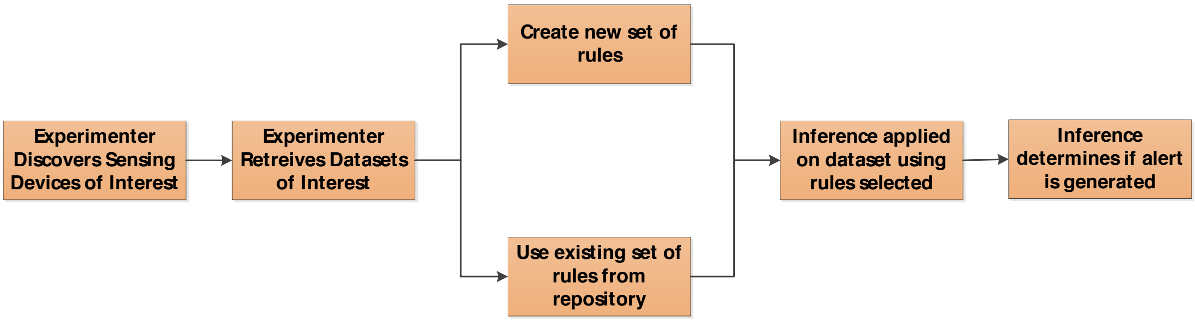

Experimenter Discovers Sensing Devices of Interest
Experimenter Retreives Datasets of Interest
Create new set of rules
Use existing set of rules from repository
Inference applied on dataset using rules selected
Inference determines if alert is generated

Figure 27. Experiment Workflow for Alerts

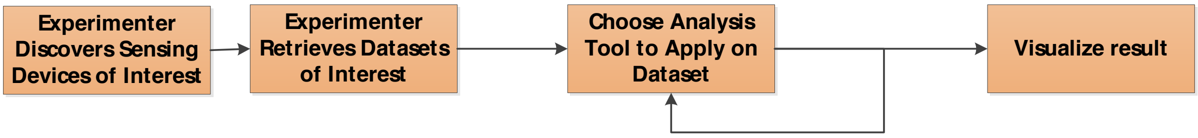

Experimenter Discovers Sensing Devices of Interest
Experimenter Retrieves Datasets of Interest
Choose Analysis Tool to Apply on Dataset
Visualize result

Figure 28. Experiment Workflow for Hypothesis

8 CONCLUSIONS

This deliverable presented and discussed the specification and implementation of an innovative concept Experiment-as-a-Service (EaaS). The EaaS model allows researchers and experimenters to define and execute data-intensive experiments on top of heterogeneous set of IoT testbeds. This is achieved by aggregating and ensuring the interoperability of data and services offered by different platforms or testbeds. The work discussed in this document fulfils the EaaS related requirements specified in [3].

We took inspiration from existing work having a complementary objective in order to design the EaaS concept. In conclusion, the Meta-Cloud architecture and its implementation have been carried out. The use of the Meta-Cloud is explained through the experiment and data workflows. The Meta-Cloud is employing the EaaS model, a cornerstone component which has been designed and implemented called FIESTA-IoT Experiment Description Specification (FEDSpec) which is capable of hosting experiments of a specific experimenter. The Experiment Registry Management (ERM) tool (UI & API) has been designed and implemented that allows experimenters to easily manage and interact with FEDSpec. The integration of Node-RED in order to generate experiment DSL automatically is discussed. The workflow of the FIESTA-IoT Analytics Toolkit (FAT) and Experiment Execution Engine (EEE) is specified. In addition, a concrete use case namely the LivingLab experiment is implemented and discussed.

Nonetheless, EaaS model is a core part of the FIESTA-IoT infrastructure that allows the consumption of aggregated data streams and services via developing applications (experiments). The work carried out in this regard is well documented and made available to the users (experimenters) of the FIESTA-IoT infrastructure. The three-in-house experiments have used this model to perform their experiments. Furthermore, fine tuning the specification and implementation of EaaS model will be carried out after receiving feedback by third parties who will perform experiments by joining the FIESTA-IoT consortium via open calls.

REFERENCES

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

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

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

[4] FIESTA-IoT, “Deliverable 3.1: Semantic Models for Testbeds, Interoperability and Mobility Support, and Best Practices V1”, 2016.

[5] FIESTA-IoT, “Deliverable 3.2: Semantic Models for Testbeds, Interoperability and Mobility Support, and Best Practices V2”, 2016.

[6] FIESTA-IoT, “Deliverable 3.6: Concept and Development for IoT Data Analytics and IoT Stream and Service Management”, 2017

[7] FIESTA-IoT, “Deliverable 4.1: EaaS Model Specification and Implementation”, 2016.

[8] FIESTA-IoT, “Deliverable 4.3: Authentication Authorization, Data Protection and Reservation of Resources”, 2016

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

[10] FIESTA-IoT, “Deliverable 4.7: Infrastructure for Submitting and Managing IoT Experiments V1”, 2016

[11] M. Nati, A. Gluhak, J. Domaszewicz, S. Lalis, and K. Moessner, “Lessons from SmartCampus: External Experimenting with User-Centric Internet-of-Things Testbed,” Wireless Personal Communications, pp. 1–15, 2014.

[12] Pallot M. (2009). Engaging Users into Research and Innovation: The Living Lab Approach as a User Centred Open Innovation Ecosystem. Webergence Blog. http://www.cwe-projects.eu/pub/bscw.cgi/1760838?id=715404_1760838

[13] R. Agarwal, D. Fernandez, T. Elsaleh, A.Gyrard, J. Lanza, L. Sanchez, N. Georgantas, V. Issarny, “*Unified IoT Ontology to Enable Interoperability and Federation of Testbeds*”, IEEE 3rd WF-IoT, Dec 2016, Reston Virginia, US 2016.

APPENDIX I- DEFINED SCHEMATA

Table 9. FEDSpec Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified" targetNamespace="http://www.fiesta-iot.eu/fedspec"
 xmlns:fed="http://www.fiesta-iot.eu/fedspec"
xmlns:prt="http://www.w3.org/2007/SPARQL/protocol-types#">

 <xs:import namespace="http://www.w3.org/2007/SPARQL/protocol-types#"
  schemaLocation="sparql/protocol-types.xsd" />

 <xs:element name="FEDSpec">
  <xs:annotation>
   <xs:documentation>FIESTA-IoT Experiment Description Specification
   </xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element maxOccurs="unbounded" ref="fed:FEMO" />
   </xs:sequence>
   <xs:attribute name="userID" use="required" type="xs:anyURI" />
  </xs:complexType>
 </xs:element>

 <xs:element name="FEMO">
  <xs:annotation>
   <xs:documentation>FIESTA-IoT Experiment Model Object
   </xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element minOccurs="0" maxOccurs="1" ref="fed:description" />
    <xs:element minOccurs="0" maxOccurs="1" ref="fed:EDM" />
    <xs:element ref="fed:domainOfInterest" />
    <xs:element maxOccurs="unbounded" ref="fed:FISMO" />
   </xs:sequence>
   <xs:attribute name="id" use="optional" type="xs:anyURI" />
   <xs:attribute name="name" type="xs:NCName" use="required" />
  </xs:complexType>
 </xs:element>

 <xs:element name="description" type="xs:string" />

 <xs:element name="EDM" type="xs:string">
  <xs:annotation>
   <xs:documentation>Experiment design metadata.</xs:documentation>
  </xs:annotation>
 </xs:element>

 <xs:element name="FISMO">
  <xs:annotation>
   <xs:documentation>FIESTA-IoT Service Model Object</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element minOccurs="0" maxOccurs="1" ref="fed:description" />
    <xs:element minOccurs="0" name="discoverable" type="xs:boolean"
    default="false" />
   <xs:element ref="fed:experimentControl" />
   <xs:element ref="fed:experimentOutput" />
   <xs:element ref="fed:queryControl" minOccurs="0" />
   <xs:element name="service" nillable="false" type="xs:anyURI"
    minOccurs="0" />
   <xs:element ref="fed:rule" minOccurs="0" />
  </xs:sequence>
  <xs:attribute name="id" use="optional" type="xs:anyURI" />
  <xs:attribute name="name" type="xs:NCName" use="required" />
 </xs:complexType>
</xs:element>

<xs:element name="queryControl">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="fed:quantityKind" minOccurs="0" />
   <xs:element ref="fed:staticLocation" minOccurs="0" />
   <xs:element ref="fed:queryInterval" minOccurs="0" />
   <xs:element ref="prt:query-request" maxOccurs="1"
    minOccurs="1" />
   <xs:element maxOccurs="1" minOccurs="0" ref="fed:dynamicAttrs" />
  </xs:sequence>
 </xs:complexType>
</xs:element>

<xs:element name="domainKnowledge" type="xs:anyURI" />

<xs:element name="staticLocation">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="latitude" type="xs:string" />
   <xs:element name="longitude" type="xs:string" />
  </xs:sequence>
 </xs:complexType>
</xs:element>

<xs:element name="queryInterval">
 <xs:complexType>
  <xs:sequence>
   <xs:element minOccurs="0" name="fromDateTime" type="xs:dateTime" />
   <xs:element minOccurs="0" name="toDateTime" type="xs:dateTime" />
   <xs:element minOccurs="0" name="intervalNowToPast" type="xs:int" />
  </xs:sequence>
 </xs:complexType>
</xs:element>

<xs:element name="experimentControl">
 <xs:complexType>
  <xs:sequence>
   <xs:element minOccurs="0" ref="fed:scheduling" maxOccurs="1" />
   <xs:element name="trigger" type="xs:anyURI" minOccurs="0" />
   <xs:element name="reportIfEmpty" type="xs:boolean"
    minOccurs="0" default="true" />
  </xs:sequence>
 </xs:complexType>
</xs:element>

<xs:element name="rule">
```

```
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="fed:ruleDefinition" />
   <xs:element ref="fed:domainKnowledge" />
   <xs:element ref="fed:quantityKind" />
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" />
 </xs:complexType>
</xs:element>

<xs:element name="ruleDefinition" type="xs:string">
 <xs:annotation>
  <xs:documentation>i.e. Jena rule or SPARQL construct
  </xs:documentation>
 </xs:annotation>
</xs:element>

<xs:element name="domainOfInterest">
 <xs:annotation>
  <xs:documentation>List of URLs linking with M3-lite taxonomy.
  </xs:documentation>
 </xs:annotation>
 <xs:simpleType>
  <xs:list itemType="xs:anyURI" />
 </xs:simpleType>
</xs:element>

<xs:element name="quantityKind">
 <xs:annotation>
  <xs:documentation>List of URLs linking with M3-lite taxonomy.
  </xs:documentation>
 </xs:annotation>
 <xs:simpleType>
  <xs:annotation>
   <xs:documentation>URL linking with M3-lite taxonomy.
   </xs:documentation>
  </xs:annotation>
  <xs:list itemType="xs:anyURI" />
 </xs:simpleType>
</xs:element>

<xs:element name="scheduling">
 <xs:complexType>
  <xs:all>
   <xs:element form="qualified" name="startTime" type="xs:dateTime"
    minOccurs="0" />
   <xs:element name="Periodicity" minOccurs="0" maxOccurs="1"
    type="xs:int" />
   <xs:element minOccurs="0" name="stopTime" type="xs:dateTime" />
  </xs:all>
 </xs:complexType>
</xs:element>

<xs:element name="experimentOutput">
 <xs:complexType>
  <xs:sequence maxOccurs="1" minOccurs="1">
   <xs:element minOccurs="0" ref="fed:file" maxOccurs="unbounded" />
   <xs:element maxOccurs="unbounded" ref="fed:widget"
    minOccurs="0" />
  </xs:sequence>
  <xs:attribute name="location" type="xs:anyURI" />
 </xs:complexType>
</xs:element>

<xs:element name="file">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="type" type="xs:string" />
  </xs:sequence>
 </xs:complexType>
</xs:element>

<xs:element name="widget">
 <xs:complexType>
  <xs:sequence>
   <xs:element maxOccurs="unbounded" ref="fed:presentationAttr" />
  </xs:sequence>
  <xs:attribute name="widgetID" use="required" type="xs:anyURI" />
 </xs:complexType>
</xs:element>

<xs:element name="presentationAttr">
 <xs:complexType>
  <xs:attribute name="name" use="required" type="xs:string" />
  <xs:attribute name="value" use="required" type="xs:string" />
 </xs:complexType>
</xs:element>

<xs:element name="dynamicAttrs">
 <xs:annotation>
  <xs:documentation>Definition of the query dynamic attributes and
   their default values
  </xs:documentation>
 </xs:annotation>
 <xs:complexType>
  <xs:sequence>
   <xs:element name="predefinedDynamicAttr" minOccurs="0">
    <xs:complexType>
     <xs:sequence>
      <xs:element ref="fed:dynamicQueryInterval" minOccurs="0" />
      <xs:element ref="fed:dynamicGeoLocation" minOccurs="0" />
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="dynamicAttr" maxOccurs="unbounded"
    minOccurs="0">
    <xs:complexType>
     <xs:attribute name="name" type="xs:string" />
     <xs:attribute name="value" type="xs:string" />
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>

<xs:element name="dynamicQueryInterval">
 <xs:complexType>
  <xs:sequence>
```

```
    <xs:element minOccurs="0" name="fromDateTime" type="xs:dateTime" />
    <xs:element minOccurs="0" name="toDateTime" type="xs:dateTime" />
    <xs:element minOccurs="0" name="intervalNowToPast" type="xs:int" />
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name="dynamicGeoLocation">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="latitude" type="xs:string" />
    <xs:element name="longitude" type="xs:string" />
   </xs:sequence>
  </xs:complexType>
 </xs:element>

</xs:schema>
```

Table 10: Descriptive IDs Schema

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified"
targetNamespace="urn:fiestaiot:experiment:descriptiveids:xsd:1"
 xmlns:edid="urn:fiestaiot:experiment:descriptiveids:xsd:1">

 <xs:element name="ExpDescriptiveIDs">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="edid:FemoDescriptiveID" maxOccurs="unbounded" />
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name="description" type="xs:string" />
 <xs:element name="name" type="xs:string" />

 <xs:element name="FismoDescriptiveID">
  <xs:complexType>
   <xs:sequence>
    <xs:element minOccurs="0" ref="edid:description" />
    <xs:element minOccurs="0" ref="edid:name" />
   </xs:sequence>
   <xs:attribute name="id" type="xs:anyURI" />
  </xs:complexType>
 </xs:element>

 <xs:element name="FemoDescriptiveID">
  <xs:complexType>
   <xs:sequence>
    <xs:element maxOccurs="1" minOccurs="0" ref="edid:description" />
    <xs:element minOccurs="0" ref="edid:name" />
    <xs:element ref="edid:FismoDescriptiveID" />
   </xs:sequence>
   <xs:attribute name="id" type="xs:anyURI" />
  </xs:complexType>
 </xs:element>
```

```
</xs:schema>
```

APPENDIX II- EXPERIMENT FEDSPEC

Table 11: Complete valid FEDSpec example

```
<?xml version="1.0" encoding="UTF-8"?>
<fed:FEDSpec
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:fed="http://www.fiesta-iot.eu/fedspec"
 xmlns:prt="http://www.w3.org/2007/SPARQL/protocol-types#"
 xmlns:vbr="http://www.w3.org/2007/SPARQL/results#"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.fiesta-iot.eu/fedspec file:/C:/FIESTA/FEDSpec.xsd"
                                                                     userID="testuser1">

 <fed:FEMO name="MySecondExperiment">
  <fed:description>LargeScale crowdsensing experiment</fed:description>
  <fed:domainOfInterest>http://purl.org/iot/vocab/m3-lite#Transportation
   http://purl.org/iot/vocab/m3-lite#Pollution
   http://purl.org/iot/vocab/m3-lite#City
   http://purl.org/iot/vocab/m3-lite#Health
  </fed:domainOfInterest>
  <fed:FISMO name="2ndUseCase">
   <fed:description>Over time all noise observations for a given location</fed:description>
   <fed:discoverable>true</fed:discoverable>
   <fed:experimentControl>
    <fed:scheduling>
     <fed:startTime>2016-11-08T18:50:00.0Z</fed:startTime>
     <fed:Periodicity>250</fed:Periodicity>
     <fed:stopTime>2017-11-08T18:49:59.0Z</fed:stopTime>
    </fed:scheduling>
   </fed:experimentControl>
   <fed:experimentOutput
    location="http://ExperimentServer.org/store/"
   ></fed:experimentOutput>
   <fed:queryControl>
    <prt:query-request>
     <query><![CDATA[
                        # [1 / 1] visualization type: 'Gauge' and sensors
                        Prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
                        Prefix iotlite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#>
                        Prefix dul: <http://www.loa.istc.cnr.it/ontologies/DUL.owl#>
                        Prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
                        Prefix time: <http://www.w3.org/2006/time#>
                        Prefix m3-lite: <http://purl.org/iot/vocab/m3-lite#>
                        Prefix xsd: <http://www.w3.org/2001/XMLSchema#>
                        select ?s ?tim ?val
                        where {
                            ?o a ssn:Observation.
                            ?o ssn:observedBy ?s.
                            ?o ssn:observedProperty ?qkr.
                            ?qkr a ?qk.
                            Values ?qk {m3-lite:Sound m3-lite:SoundPressureLevelAmbient}
                            ?o ssn:observationSamplingTime ?t.
                            ?o geo:location ?point.
                            ?point geo:lat "4.346104E1"^^xsd:double.
                            ?point geo:long "-3.80649E0"^^xsd:double.
                            ?t time:inXSDDateTime ?ti.
                            ?o ssn:observationResult ?or.
                            ?or  ssn:hasValue ?v.
```

```
                          ?v dul:hasDataValue ?val.
                      } group by (?s) ?tim ?val
                  ]]></query>
  </prt:query-request>
 </fed:queryControl>
</fed:FISMO>
<fed:FISMO name="3rdUseCase">
 <fed:description>Over time noise observations for a given bounding
                        box (time period in scheduling)</fed:description>
 <fed:discoverable>true</fed:discoverable>
 <fed:experimentControl>
  <fed:scheduling>
   <fed:startTime>2016-11-08T18:50:00.0Z</fed:startTime>
   <fed:Periodicity>250</fed:Periodicity>
   <fed:stopTime>2017-11-08T18:49:59.0Z</fed:stopTime>
  </fed:scheduling>
 </fed:experimentControl>
 <fed:experimentOutput
  location="http://ExperimentServer.org/store/"
 ></fed:experimentOutput>
 <fed:queryControl>
  <prt:query-request>
   <query><![CDATA[
                      # [1 / 1] visualization type: 'Gauge' and sensors
                      Prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
                      Prefix iotlite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#>
                      Prefix dul: <http://www.loa.istc.cnr.it/ontologies/DUL.owl#>
                      Prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
                      Prefix time: <http://www.w3.org/2006/time#>
                      Prefix m3-lite: <http://purl.org/iot/vocab/m3-lite#>
                      Prefix xsd: <http://www.w3.org/2001/XMLSchema#>
                      select ?s (max(?ti) as ?tim) ?val ?lat ?long
                      where {
                          ?o a ssn:Observation.
                          ?o ssn:observedBy ?s.
                          ?o ssn:observedProperty ?qkr.
                          ?qkr a ?qk.
                          Values ?qk {m3-lite:Sound m3-lite:SoundPressureLevelAmbient}
                          ?o ssn:observationSamplingTime ?t.
                          ?o geo:location ?point.
                          ?point geo:lat ?lat.
                          ?point geo:long ?long.
                          ?t time:inXSDDateTime ?ti.
                          ?o ssn:observationResult ?or.
                          ?or  ssn:hasValue ?v.
                          ?v dul:hasDataValue ?val.
                          {
                              select  (max(?dt)as ?ti) ?s
                              where {
                                  ?o a ssn:Observation.
                                  ?o ssn:observedBy ?s.
                                  ?o ssn:observedProperty ?qkr.
                                  ?qkr a ?qk.
                                  Values ?qk {m3-lite:Sound
                                           m3-lite:SoundPressureLevelAmbient}
                                  ?o ssn:observationSamplingTime ?t.
                                  ?t time:inXSDDateTime ?dt.
                              }group by (?s)
                          }
                          FILTER (
                             (xsd:double(?lat) >= "-90"^^xsd:double)
                          && (xsd:double(?lat) <= "90"^^xsd:double)
                          && ( xsd:double(?long) >= "-180"^^xsd:double)
                          && ( xsd:double(?long) <= "180"^^xsd:double)
                          )
                      } group by (?s) ?tim ?val ?lat ?long
                  ]]></query>
  </prt:query-request>
 </fed:queryControl>
</fed:FISMO>
<fed:FISMO name="4thUseCase">
 <fed:description>3rd usecase with noise more than x dB(A)</fed:description>
 <fed:discoverable>true</fed:discoverable>
 <fed:experimentControl>
  <fed:scheduling>
   <fed:startTime>2016-11-08T18:50:00.0Z</fed:startTime>
   <fed:Periodicity>250</fed:Periodicity>
   <fed:stopTime>2017-11-08T18:49:59.0Z</fed:stopTime>
  </fed:scheduling>
  <fed:reportIfEmpty>false</fed:reportIfEmpty>
 </fed:experimentControl>
 <fed:experimentOutput
  location="http://ExperimentServer.org/store/">
  <fed:file>
   <fed:type>application/xml</fed:type>
  </fed:file>
 </fed:experimentOutput>
 <fed:queryControl>
  <prt:query-request>
   <query><![CDATA[
                      # [1 / 1] visualization type: 'Gauge' and sensors
                      Prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
                      Prefix iotlite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#>
                      Prefix dul: <http://www.loa.istc.cnr.it/ontologies/DUL.owl#>
                      Prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
                      Prefix time: <http://www.w3.org/2006/time#>
                      Prefix m3-lite: <http://purl.org/iot/vocab/m3-lite#>
                      Prefix xsd: <http://www.w3.org/2001/XMLSchema#>
                      select ?s (max(?ti) as ?tim) ?val ?lat ?long
                      where {
                          ?o a ssn:Observation.
                          ?o ssn:observedBy ?sensorID.
                          ?o ssn:observedProperty ?qkr.
                          ?qkr a ?qk.
                          Values ?qk {m3-lite:Sound m3-lite:SoundPressureLevelAmbient}
                          ?o ssn:observationSamplingTime ?t.
                          ?o geo:location ?point.
                          ?point geo:lat ?latitude.
                          ?point geo:long ?longitude.
                          ?t time:inXSDDateTime ?ti.
                          ?o ssn:observationResult ?or.
                          ?or  ssn:hasValue ?v.
                          ?v dul:hasDataValue ?value.
                          FILTER (
                             (xsd:double(?latitude) >= "-90"^^xsd:double)
                          && (xsd:double(?latitude) <= "90"^^xsd:double)
                          && ( xsd:double(?longitude) >= "-180"^^xsd:double)
                          && ( xsd:double(?longitude) <= "180"^^xsd:double)
                          )
                          FILTER(?value>="50"^^xsd:double)
                          FILTER(?ti > "%%fromDateTime%%"^^xsd:dateTime && ?ti
                               < "%%toDateTime%%"^^xsd:dateTime)
                      } group by ?sensorID ?time ?value ?latitude ?longitude

                  ]]></query>
  </prt:query-request>
  <fed:dynamicAttrs>
    <fed:predefinedDynamicAttr>
      <fed:dynamicQueryInterval>
        <fed:intervalNowToPast>600000</fed:intervalNowToPast>
      </fed:dynamicQueryInterval>
    </fed:predefinedDynamicAttr>
   </fed:dynamicAttrs>
 </fed:queryControl>
</fed:FISMO>
<fed:FISMO name="5thUseCase">
 <fed:description>3rd usecase with noise less than x dB(A)</fed:description>
 <fed:discoverable>true</fed:discoverable>
 <fed:experimentControl>
  <fed:scheduling>
   <fed:startTime>2016-11-08T18:50:00.0Z</fed:startTime>
   <fed:Periodicity>250</fed:Periodicity>
   <fed:stopTime>2017-11-08T18:49:59.0Z</fed:stopTime>
  </fed:scheduling>
 </fed:experimentControl>
 <fed:experimentOutput location="http://ExperimentServer.org/store/">
 </fed:experimentOutput>
 <fed:queryControl>
  <prt:query-request>
   <query><![CDATA[
                      # [1 / 1] visualization type: 'Gauge' and sensors
                      Prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
                      Prefix iotlite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#>
                      Prefix dul: <http://www.loa.istc.cnr.it/ontologies/DUL.owl#>
                      Prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
                      Prefix time: <http://www.w3.org/2006/time#>
                      Prefix m3-lite: <http://purl.org/iot/vocab/m3-lite#>
                      Prefix xsd: <http://www.w3.org/2001/XMLSchema#>
                      select ?s (max(?ti) as ?tim) ?val ?lat ?long
                      where {
                          ?o a ssn:Observation.
                          ?o ssn:observedBy ?s.
                          ?o ssn:observedProperty ?qkr.
                          ?qkr a ?qk.
                          Values ?qk {m3-lite:Sound m3-lite:SoundPressureLevelAmbient}
                          ?o ssn:observationSamplingTime ?t.
                          ?o geo:location ?point.
                          ?point geo:lat ?lat.
                          ?point geo:long ?long.
                          ?t time:inXSDDateTime ?ti.
                          ?o ssn:observationResult ?or.
                          ?or  ssn:hasValue ?v.
                          ?v dul:hasDataValue ?val.
                          {
                              select  (max(?dt)as ?ti) ?s
                              where {
                                  ?o a ssn:Observation.
                                  ?o ssn:observedBy ?s.
                                  ?o ssn:observedProperty ?qkr.
                                  ?qkr a ?qk.
                                  Values ?qk {m3-lite:Sound
                                      m3-lite:SoundPressureLevelAmbient}
                                  ?o ssn:observationSamplingTime ?t.
                                  ?t time:inXSDDateTime ?dt.
                              }group by (?s)
                          }
                          FILTER (
                             (xsd:double(?lat) >= "-90"^^xsd:double)
                          && (xsd:double(?lat) <= "90"^^xsd:double)
                          && ( xsd:double(?long) >= "-180"^^xsd:double)
                          && ( xsd:double(?long) <= "180"^^xsd:double)
                          )
                          FILTER(?val<="45"^^xsd:double)
                      } group by (?s) ?tim ?val ?lat ?long
                  ]]></query>
    </prt:query-request>
   </fed:queryControl>
  </fed:FISMO>
 </fed:FEMO>
</fed:FEDSpec>
```