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

EaaS Model Specification and Implementation V1

Document Id:	FIESTA-IoTD41-060920-Draft
File Name:	FIESTA-IoTD41-060920-Draft.pdf
Document reference:	Deliverable 4.1
Version:	Draft
Editor:	Amelie Gyrard, Martin Serrano
Organisation:	NUIG-DERI, Insight Center for Data Analytics
Date:	20 / 09 / 2016
Document type:	Deliverable
Dissemination level:	PU, CO

DOCUMENT HISTORY

Rev.	Author(s)	Organisation(s)	Date	Comments
V01	Martin Serrano	NUIG-DEI	YYYY/MM/DD	Initial Draft Proposal
V01	Amelie Gyrard Cassio Prazeres	NUIG-DERI	2015/05/27 2015/06/17	Backgound, State of the art, first insights. LDoW-PaN model: Linked Data on the Web - Presentation and Navigation
V01	Amelie Gyrard	NUIG-DERI	2015/06/30 2015/07/07 2015/07/07 2015/09/07 2015/10/07 2015/14/07	Frequently updated with new papers read, new ideas following telcos WP4 Architecture with the different tasks WP4 FIESTA Meta-Cloud Architecture Experiments specifications section WP4 overview section with tasks & (deliverable table) Description of testbeds (model/ontology, dataset, architecture)
V02	Martin Serrano, Amelie Gyrard	NUIG-DERI	2015/16/07 2015/20/07 2015/21/07	Refactoring, some sections have been moved to Deliverable 4.3 brainstorming document (all sections related to testbed and linked data) Section Fed4Fire Global vision of FIESTA
V02	Maria Bermudez Luis Sanchez Amelie Gyrard	UNIS UNICAN NUIG	2015/22/07 2015/22/07 2015/30/07	Contribute to the functionality table All fruitful feedback for all telcos and exchanged emails which enables improving the document and pictures and go forward. To improve the picture of the fiesta ontology main modules
V02	Nikos Kefalakis	AIT	2015/05/08	Initial proposal of an FIESTA DSL along with provided services for managing experiments.
V03	Minwoo Ryu Ramnath teja chekka	KETI	2015/24/08	Survey on existing DSLs and DSL related IDE tools
V04	Amelie Gyrard	NUIG-DERI	2015/09/15 2015/09/15	Accept changes from AIT and KETI Memento Patel et al. section, add memento/thought/ tools mentioned/discussed during telco with Rachit and Nikos, SuperStreamCollider section + Update Fed4Fire section

			2016/03/14	Ontologies for describing services OWL-S section
V05	Amelie Gyrard	NUIG-DERI ALL	2016/04/14 2016/04/27 2016/05/10 2016/05/11	Restructuration of the document – Table of Contents State of the art from T4.4 moved here Table of content restructuration
V06	Amelie Gyrard	NUIG-DERI	2016/06/08	Update ToC OneDrive collaborative version Experiment ontology section Remove draft idea section experiment/fiesta ontology
V07	Amelie Gyrard	NUIG-DERI	2016/06/17	OneDrive latency issue Update on Word Composition of service section: example picture + description
V08	Amelie Gyrard Martin Serrano	NUIG-DERI	2016/06/20	Restructuration Refactoring experiment ontology section description Methodology section
V09	Amelie Gyrard	NUIG-DERI	2016/06/22 2016/06/24	Fiesta-IoT platform, technologies and tools pictures (Deliverable 4.4) Register experiment sequence diagram + description Executing reasoning and query engines sequence diagram + description
V10	Amelie Gyrard	NUIG-DERI	2016/06/25 2016/06/28	Update section 1 positioning, fiesta-IoT scope, wp4 overview, audience to follow WP3 deliverables, terminology definition Update methodology section
V11	Rachit Agarwal	INRIA	2016/07/04 2016/07/05	Update section 3.3.3
V12	Amelie Gyrard	NUIG-DERI	2016/08/05 2016/12/07	Data workflow to build an experiment Move LDoW-PaN model section to D4.3 Refactor limitation of existing work section Refactoring + WP4 Architecture section
V12	Alexander Willner, Ronald Steinke	Fraunhofer FOKUS	2016/08/29	Added information about Fed4FIRE CLI's and GUI's, added OMN and related ontologies
V13	David Gómez Fernández, Jorge Lanza Amelie Gyrard	UNICAN NUIG	2016/18/07 2016/05/08	Section Sample workflow UNICAN Update and unify references +

			2016/10/08	executive summary + Conclusion section
V14	Minwoo Ryu Ramnath teja chekka Amelie Gyrard	KETI NUIG	2016/18/08	Section AUTOMATICALLY generating the DSL Section Survey workflow editors: Google Blockly, NodeRed + updates: typos, figures, refs, etc.
V15	Nikos Kefalakis Amelie Gyrard	AIT NUIG	2016/01/09 2016/01/09	Added OpenIoT DSL, FIESTA-IoT DSL (FEDSpec) specification, Experiment Registry Management API specification and ERM prototype Implementation + updates: typos, figures, refs, etc.
V16	Mengxuan Zhao Tiago Teixeira Alireza Ahrabian Amelie Gyrard	EGM Unparallel UNIS NUIG	2016/19/09 2016/16/09 2016/16/09 2016/19/09	Quality review Technical Review Technical Review Address TRs and QR, generate version for submission
V17	Martin Serrano	NUIG	2016/09/15	Format Final Check-ups / updates
V18	Martin Serrano	NUIG	2016/09/15	Circulated for Approval
Draft	Martin Serrano	NUIG	2016/09/20	EC Submitted

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TERMS AND ACRONYMS

DSL	Domain Specific Language
EaaS	Experiment as a Service
EEE	Experiment Execution Engine
ERM	Experiment Registry Management
FEDSpec	FIESTA-IoT Experiment Model Object
FEMO	FIESTA-IoT Experiment Model Objects
FIRE	Future Internet Research & Experimentation
FISMO	FIESTA Service Model Object
ICO	Internet Connected Object
IERC	European Research Cluster on the Internet of Things
IoT	Internet of Things (IoT)
IoT-A	Internet of Things Architecture
LSM	Linked Stream Middleware
LOD	Linked Open Data
LOV	Linked Open Vocabularies
LOV4IoT	Linked Open Vocabularies for Internet of Things (LOV4IoT)
OAMOs	OpenIoT Application Model objects
OpenIoT	Open Source cloud solution for the Internet of Things
OSDSpec	OpenIoT Service Description specification
OSMO	OpenIoT Sensor Model Object
OWL	Ontology Web Language
SaaS	Sensing as a Service
SSC	Super Stream Collider
VoID	Vocabulary of Interlinked Datasets
WoT	Web of Things

EXECUTIVE SUMMARY

Experiment-as-a-Service (EaaS) is designed to aggregate and ensure the interoperability of data streams stemming from different platforms or testbeds, as well as the need to provide tools and techniques or building applications that horizontally integrate diverse IoT solutions, but no concrete solutions have been designed yet [Serrano et al. EaaS 2015]. Our own interpretation of EaaS is assisting experimenters and developers in designing interoperable IoT applications, more precisely in semantically annotating data, deducing meaningful knowledge from sensor data, combining applicative domains to build smarter IoT applications/experiments.

In this document, we start by investigating work having a complementary objective in order to design this EaaS innovative concept. Based on this literature study, the FIESTA-IoT Meta-Cloud methodology and architecture have been designed. 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). An API has been designed to deal and easily interact with the EaaS Model. A concrete use case has been implemented through the NodeRed workflow editor to design experiments and generating the experiment workflow compliant with the EaaS Model/DSL.

1 INTRODUCTION[1]

1.1 FIESTA-IoT Scope

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

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

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

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

[1] Sections 1.1 to 1.4 are included in WP4 deliverables in order to make each deliverable self-contained.

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

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

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

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

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

1.2 WP4 overview

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

The objectives of WP4 are:

- To specify and implement the testbed agnostic operations (e.g., data access, service execution, data/service management) that comprise the EaaS model of the project, along with tools and techniques for combining them into experiments/workflows.
- To specify and implement tools and techniques for accessing resources from the various testbeds in a secure way.
- To specify and implement tools and techniques for testbed agnostic access to data sets stemming from multiple heterogeneous IoT platforms.
- To specify and implement a portal infrastructure enabling the submission of experiments over semantically interoperable testbeds.
- To design and implement tools and techniques for managing experiments, including management of information the sensors and data streams that the experiments use, the volume of data that they consume/use, the timing of the execution of the various experiments and more.

The WP4 Tasks cross all aspects of the FIEST-IoT Infrastructure. They are:

- **T4.1 FIESTA Meta-Cloud and EaaS Model Implementation**: This task will specify and implement the EaaS model of the project in terms of the workflows that will be associated with the specification of an experiment. To this end, a modeling language (e.g., a Domain Specific Language (DSL)) for modeling experiments associated with IoT data and resources will be specified, along with tools for specifying, parsing and enacting this language. The decoding of this language will end-up executing data access and actuating functions over the underlying testbeds, based on the interfaces specified/implemented in WP3. The EaaS infastructure will be deployed within a cloud computing infrastructure (FIESTA cloud), which will interface to the individual semantically interoperable testbed (via the cloud interfaces specified and implemented in WP4). A directory service (meta-directory) will be also specified and implemented on the basis of the semantic models of IoT resources specified in WP3.
- **T4.2 Techniques for Secure Access and Reservation of Resources**: This task will ensure that experimenters will have secure access to the resources of the federated testbeds. It will deal with the provision of a solution for

authentication and authorization of resources (e.g., devices, data streams), which will be operational independently of the number and type of underlying IoT platforms and testbeds. This solution will enable experimenters to gain access to all the FIESTA testbeds through a single set of credentials. Furthermore, the task will research and provide techniques for reserving resources through keeping track of the various experiments, the data streams and IoT resources that they use, the lifetime of the experiments, logging information about the experimenters and the experiments that they execute and more. By keeping track of the resources reserved/used (at the level of experiments and experimenters) this task will provide a foundation for managing experiments and the IoT/cloud resources that they use. In terms of the implementation of the low-level security and resource reservation functionalities, results for related projects (e.g., Fed4FIRE) will be studied and appropriately reused.

- **T4.3 Testbed Agnostic Access to Datasets**: This task will implement tools and techniques for accessing data sets over the FIESTA meta-cloud infrastructure and in a way totally independent and transparent to the underlying testbeds. As a starting point the task will provide the means for querying, accessing and processing data sources/streams at the level of the FIESTA meta-cloud, based on query languages and technologies (e.g., SPARQL) that are suitable for the semantic models specified in WP3. Accordingly, add-ons will be implemented in order to provide additional functionalities required for the experimentation (e.g., access to metadata of the data streams / datasets). To this end, the cloud interfaces over the IoT/cloud testbeds (developed also in WP3) will be exploited.
- **T4.4 Experiments Submission Portal**: This task will develop a portal infrastructure, which will serve as a single entry point for accessing data, services and resources of all the federated IoT testbeds. The portal will enable experiments to browse resources and datasets, regardless of their testbed-origin. Furthermore, it will integrate the authentication and authorization functionalities developed in T4.2, as part of the users'/experimenters' management module of the portal.
- **T4.5 Experiments Development, Deployment and Management Tools**: This task will provide a range of tools for designing, deploying and managing experiments. These tools will be made available through the portal infrastructure of the previous task. In terms of experiments design, the task will provide simple visual tools that will enable the specification of data processing functions and IoT workflow over the interoperable testbeds. For example, they will enable the selection of data resources (in a testbed agnostic way), and the visual definition of processing functions over them. Moreover, they will enable the composition of IoT-oriented workflows/processes over multiple testbeds. The deployment tools to be provided will enable the enactment of the experimental workflows over the FIESTA middleware infrastructure. Finally, the management tools will enable experimenters to access information about the available testbeds, the available data sets (and their status), the running experiments, the experimenters, experimental results and research data produced during the experiments and more.

Table 1. WP4 Deliverables

No.	Deliverable	Related Task	Responsible Partner	Contributors
D4.1	EaaS Model Specification and Implementation	T4.1	NUIG-DERI	AIT, UNICAN, KETI, Inria/UNIS
D4.2	Authentication, Authorization, Data Protection and Reservation of Resources	T4.2	ITINNOV	UNICAN
D4.3	Tools and Techniques for Managing Interoperable Data sets	T4.3	NUIG-DERI	UNINNOVA
D4.4	Infrastructure for Submitting and Managing IoT Experiments	T4.4 T4.5	NUIG-DERI Inria	AIT, UNIS, UNINNOVA, UNICAN, KETI, Com4Innov

1.3 Audience

This deliverable addresses following audiences:

- **Researchers and engineers within the FIESTA-IoT consortium** will take into account various requirements in order to research, design and implement the APIs needed to support Testbeds associated to FIESTA-IoT Platform.
- **Testbed owners who wish to join FIESTA-IoT** will be able to use the tools to annotate the data their Testbed is producing. These annotation tools should comply with the semantic model proposed within FIESTA-IoT. By doing so, the Testbed can either become Class I, Class II, or Class III Testbed (see Deliverable D2.4 [FIESTA-IoT D2.4, 2015]) for the definitions of various classes of Testbeds).
- **Experiment owners who wish to join FIESTA-IoT** will be able to understand how and what IoT data is stored within the FIESTA-IoT Meta Cloud and thus would be able to align their experiments that could utilize such data.
- **Researchers on Future Internet Research and Experimentation (FIRE) focusing on semantically storing data produced by their experiments** will find guidelines to store data produced by their experiments in a semantic manner either in their own repository or utilizing the FIESTA-IoT platform. The researchers will be able use the ontology and the tools as the reference. Further, if they wish to extend/modify the ontology and tools for their own research, they will be able to do so.
- **Members of other Internet of Things (IoT) communities and projects (such as projects of the IERC cluster)** can take this document as an initial reference or inspiration to design and implement their own Testbed that also stores data that is semantically annotated.
- **Open call** participants will be able to understand better the technical details needed for them to join the FIESTA-IoT consortium.
- **Standardization bodies** will have access to this deliverable as it will be a public document and therefore the ontology developed can be standardized following the involvement and reach a wider adoption.

1.4 Terminology and Definition

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

Table 2 Terminology and Definitions table

Term	Definition
Device	Technical physical component (hardware) with communication capabilities to other Information Technology (IT) systems. A device can be attached to, or embedded inside a physical entity, or monitor a physical entity in its vicinity. The device could be: • **Sensor**: A sensor is a special device that perceives certain characteristics of the real world and transfers them into a digital representation. • **Actuator**: An actuator is a mechanical device for moving or controlling a mechanism or system. It takes energy, usually transported by air, electric current, or liquid, and converts that into some kind of motion.
Discovery	Discovery is a service to find unknown resources/entities/services based on a rough specification of the desired result. It may be utilized by a human or another service. Credentials for authorization are considered when executing the discovery.
Domain	Refers to an application area where the meaning of data corresponds to the same semantic context. For instance, pressure in Water Management Domain may refer to water pressure on pipes while in Air Quality Domain it refers to atmospheric pressure.
Information	Content of communication; data and metadata describing data. The material basis is raw data, which is processed into relevant information, including source information (e.g., analogue and state information) and derived information (e.g., statistical and historical information).
Measurement	The important data for the experimenter. It represents the minimum piece of information sent by a specific resource, which the experimenter needs in order to fulfil the objective of the experiment.
Metadata	The metadata is the additional information associated with the measurement, facilitating its understanding.
Physical Entity	Any physical object that is relevant from a user or application perspective. Physical Entities are the objects from the real world that can be sensed and measured and they are virtualized in cyber-space using Virtual Entities.
Requirement	A quantitative statement of business-need that must be met by a particular architecture or work package.
Resource	Computational element that gives access to information about or actuation capabilities on a Physical Entity.

Testbed	A Testbed is an environment that allows experimentation and testing for research and development products. A Testbed provides a rigorous, transparent and replicable environment for experimentation and testing.
Federated Testbeds	A Testbed federation or federated Testbeds is the interconnection of two or more independent Testbeds for the creation of a richer environment for experimentation and testing, and for the increased multilateral benefit of the users of the individual independent Testbeds.
Interoperability	The ability of two or more systems or components to exchange information and use the information that has been exchanged.
Experiment	Experiment is a test under controlled conditions that is made to demonstrate a known truth, examine the validity of a hypothesis, or determine the efficacy of something previously untried.
Experiment-as-a-Service	Concept designed for interoperability of experiments and assist developers in developing or designing experiments.
Semantic Interoperability	Semantic interoperability is the ability of computer systems to exchange data with unambiguous, shared meaning. Semantic interoperability is a requirement to enable machine computable logic, inference, knowledge discovery, and data federation between information systems.
Service	Services (Technology) are services designed to facilitate the use of technology by end users. These services provide specialized technology-oriented solutions by combining the processes/functions of software, hardware, networks, telecommunications and electronics.
Virtual Entity	Computational or data element representing a Physical Entity. Virtual Entities can be either Active or Passive Digital Entities.

2 BACKGROUND & RELATED WORK

In this section, we investigate work having a complementary objective in order to design this EaaS innovative concept: interoperability of services, Domain Specific Languages (DSL) and data workflows.

2.1 Interoperability of Services

This section explains the literature providing solutions towards interoperability of services such as Web of Things (WoT), Sensing-as-a-Service (SaaS), Linked Open Services, Linked Data Services and Semantic Web Services.

2.1.1 Web of Things & Sensing-as-a-Service (SaaS)

The concept of '**Web of Things (WoT)**' has been designed to connect sensors to the Web and easily get access to sensor data to publish and share it on the Web [Guinard et al. WoT 2010].

Perera et al define the concept of '**Sensing-as-a-Service (SaaS)**' within the OpenIoT project [Perera et al. SaaS 2015] [Perera et al. SaaS 2014] to easily retrieve data produced by heterogeneous hardware by sending data on the Web, following the idea of Web of Things. They design a set of tools to implement the SaaS concept as explained in Table 3. Each tool introduced in Table 3 is explained below.

Table 3. Set of tools developed by Perera et al. to implement SaaS

Name of the tool	Component type	Functionalities
CADDOT	Model	Sensor discovery ICO configuration
DAM4GSN	Model	Resource
MOSDEN	Data processing engine	Data processing engine
CASCOM	Model/Middleware	Context-aware configuration of filtering, fusion, and reasoning
CASSARAM	Architecture	Selecting a subset of relevant sensors out of a large set of sensors with similar functionality and capabilities

They overcome the following research challenges:

- Dynamic configuration of internet connected objects and hubs
  - How to make the discovery and configuration of ICOs efficient and effective? (faster? Less time consuming? Less labour intensive?)

  - How to handle mobility, and dynamicity of the ICOs?
  - How to handle the heterogeneity of ICOs?
  - How to support context-aware sensor configuration at run time?
  - What parameters of ICOs may need to be configured at run time?
  - How to utilize existing open source technologies to address these issues in a cost effective and scalable manner?
  - How to enable efficient local data processing, filtering and fusing?
- Dynamic configuration of internet of things cloud platform
  - How to facilitate the dynamic discovery and configuration of IoT services in an efficient and effective (faster? Less time consuming? Less labour intensive?)
  - How to support context-aware service provisioning at run time?
  - Why is context important in resource discovery in IoT?
  - What context parameter would be important to data consumers?
  - How to allow the data consumers to achieve their objectives without knowing underlying technical details?
  - How to utilize existing open source technologies to address these issues in a cost effective and scalable manner?
  - How to enhance existing software platforms in a way that both technical experts and non-technical personnel would be benefit and able to sensing as a service platforms successfully.

To overcome such research challenges, Perera et al. design five tools as depicted in Figure 2: CADDOT, DAM4GSN, MOSDEN, CASCOM and CASSARAM.

CADDOT (Context Aware Dynamic Discovery of Things) is a model and a tool to ease the tasks of sensor discovery and configuration of Internet Connected Objects (ICOs) as depicted in Figure 1 [Perera et al. 2014]. The major objective is to support non-technical users by allowing them to configure smart environments, through the SmartLink tool implementing the CADDOT model, in the following tasks: (1) the number of sensors, (2) deal with heterogeneous hardware and software, (3) scheduling, sampling rate, communication frequency, (4) data acquisition, (5) dynamicity, and (6) context. This approach interlinks sensor hardware to cloud-based IoT middleware platforms. The CADDOT model comprises 8 phases: (1) Detect, (2) Extract, (3) Identify, (4) Find, (5) Retrieve, (6) Register, (7) Reason, and (8) Configure.

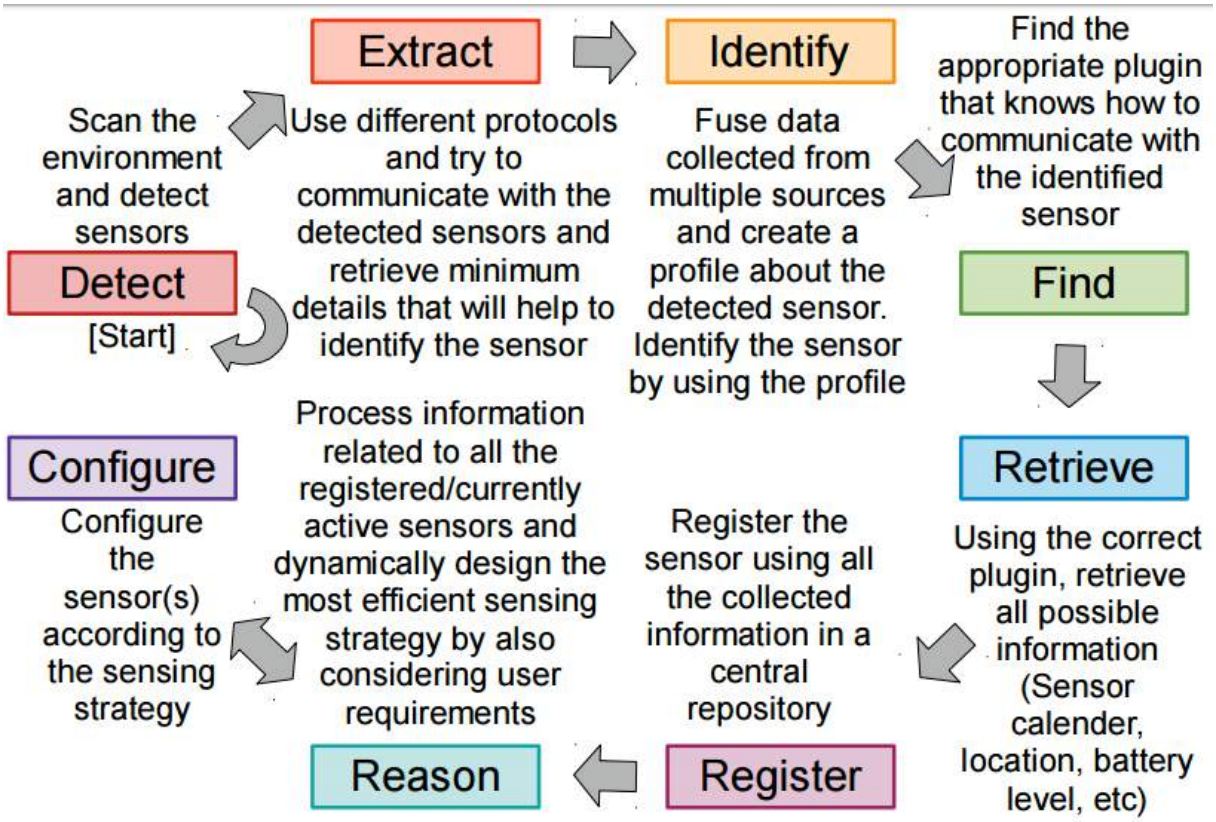

Figure 1.CADDOT model [Perera et al. 2014].

DAM4GSN (Data Acquisition Model for GSN) is designed for resources constrained devices to collect sensor data from sensors embedded in smart phones and maintain connectivity. DAM4GSN captures sensor data using sensors embedded within the mobile phones and pushes them to the IoT cloud platforms for further processing. This experience leads them to develop the more comprehensive platform called **MOSDEN**.

Mobile devices comprise several sensors:

- *Motion sensors* such as accelerometer, gravity, gyroscope, linear accelerometer, and rotation vector.
- *Position sensors* such as orientation, geomagnetic field, and proximity
- *Environment sensors* such as light, pressure, humidity and temperature.
- *Audio sensor* such as microphone.
- *Video sensor* such as camera.
- *Others sensors* such as digital compass and Global Positioning System (GPS).

MOSDEN (Mobile Sensor Data Processing Engine) is a plug-in based IoT middleware for mobile devices that allows collecting, processing, fusing and filtering sensor data without programming efforts. The main added value is to process the data locally in case of enough computational resources and domain knowledge; otherwise the data can be sent to the cloud for further processing and will reduce the network communication. MOSDEN is based on GSN [Aberer et al. GSN 2007] and W3C SSN ontology [Compton et al. SSN 2012].

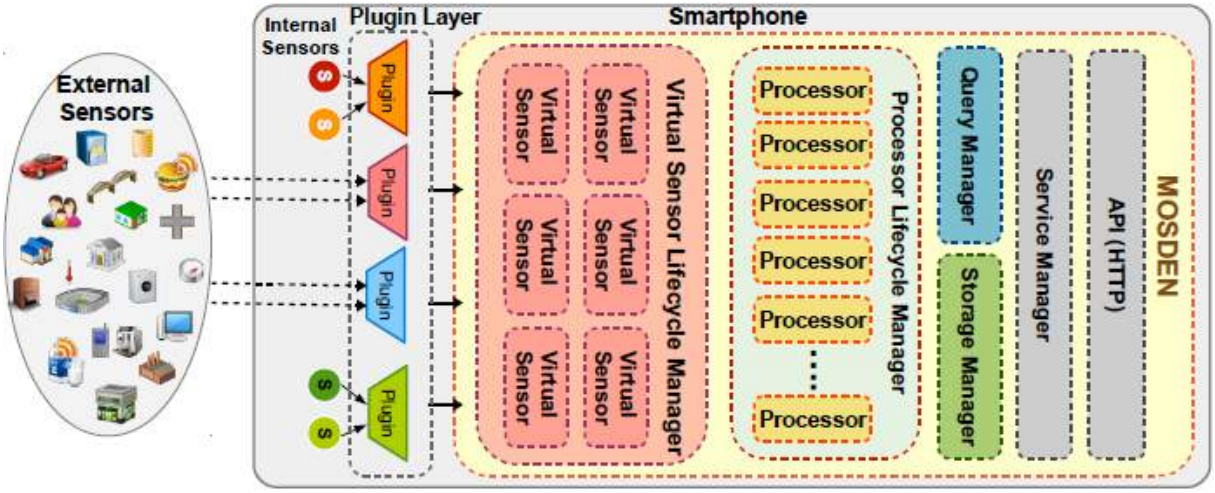

Figure 2. MOSDEN Architecture designed by Perera et al. [Perera et al. SaaS 2015]

CASCOM (Context-aware sensor configuration model) addresses the challenge of automated context-aware configuration of filtering, fusion, and reasoning mechanisms in IoT middleware. It enables non-technical users collecting relevant sensor data through cloud platforms [Perera et al. 2013]. The cloud platform will automatically recognize which sensors are needed to achieve the user's request. CASCOM is based on semantic web technologies to overcome this challenge.

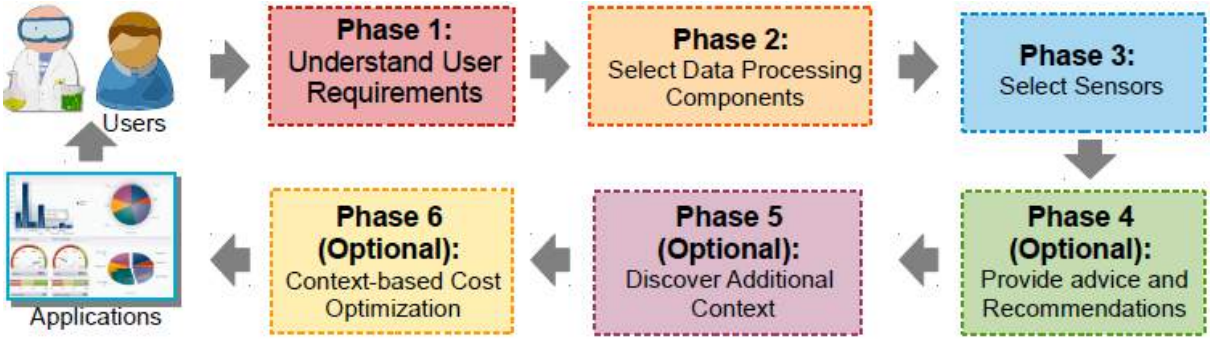

Figure 3. CASCOM execution flow designed by Perera et al. [Perera et al. SaaS 2015]

Perera designs the Question and Answer Oriented Task Description Ontology (QA-TDO) aligned with the W3C Semantic Sensor Networks (SSN) ontology [Compton et al. SSN 2012] and the Software Component Ontology (SCO).

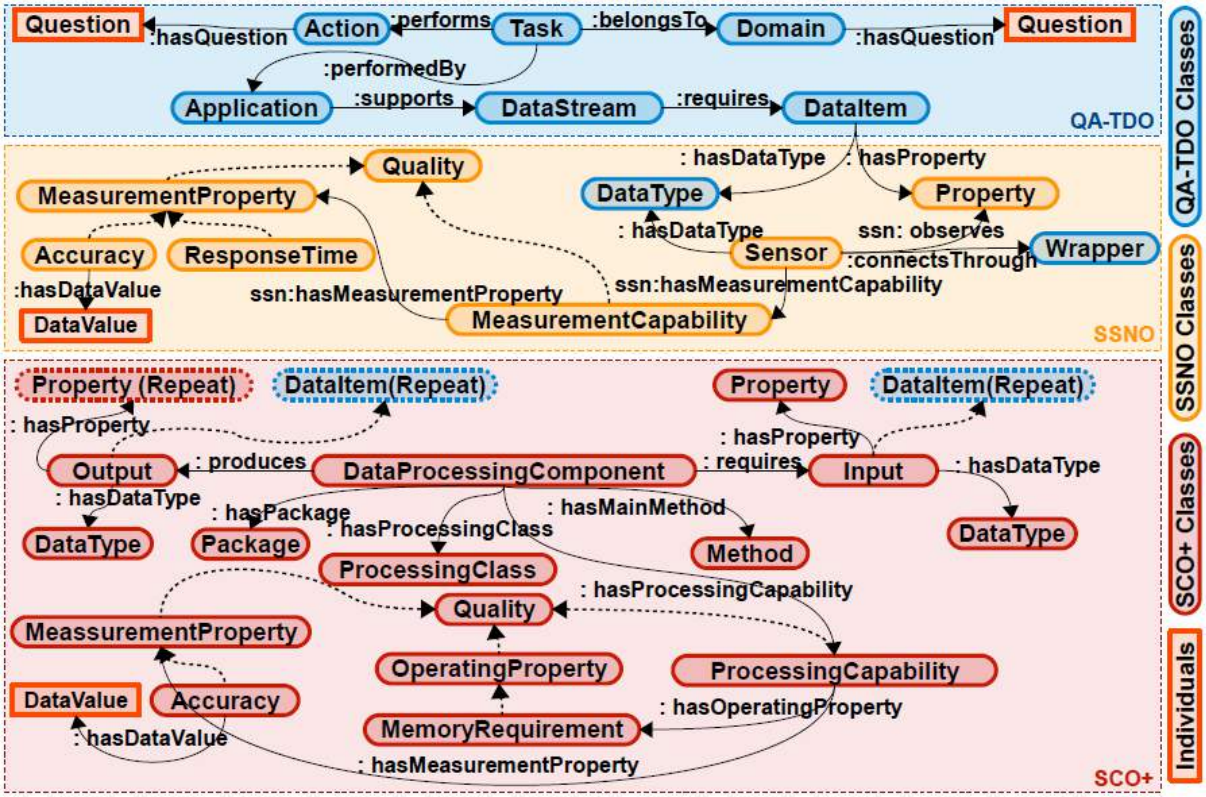

Figure 4. QA-TDO ontology aligned with W3C SSN and SCO ontologies [Perera et al. SaaS 2015]

CASSARAM (Context aware sensor search, selection and ranking model) addresses the challenge of efficiently selecting a subset of relevant sensors out of a large set of sensors with similar functionality and capabilities. CASSARAM takes user preferences into account and considers a broad range of sensor characteristics such as reliability, accuracy, location and battery life.

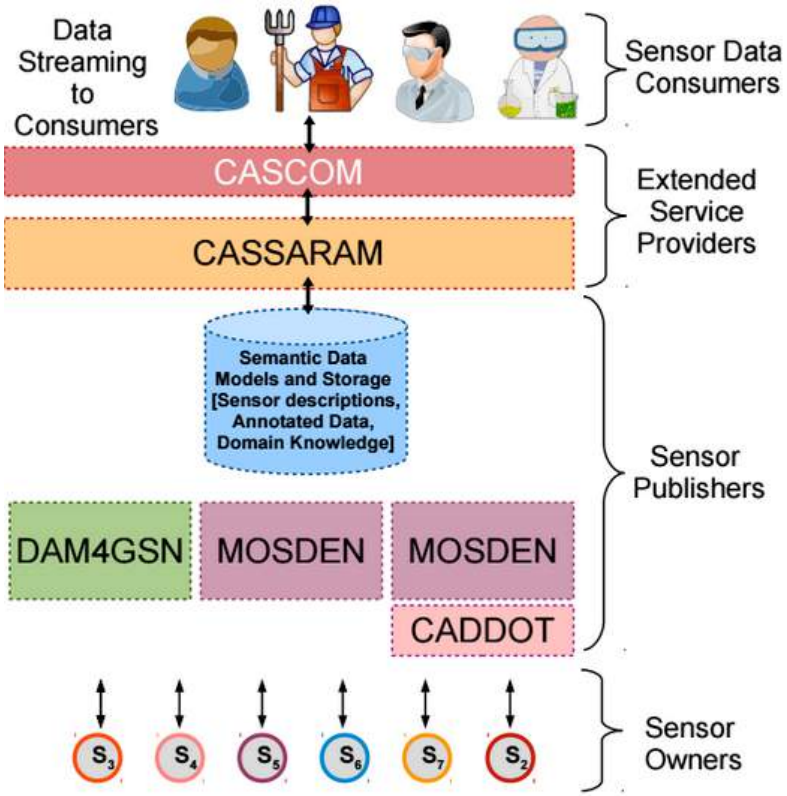

Figure 5. Architecture designed by Perera et al. [Perera et al. SaaS 2015]

Moreover, their work seems really reliable since they publish a survey regarding context-awareness applied to IoT where they analyse and compare about 50 projects [Perera et al. Survey 2014].

Perera et al. explain as a future work to address security issues, reasoning aspect, and real-time. His work has been applied to two uses cases: Phenonet[2], an agriculture scenario and OpenIoT.

2.1.2 Linked Open Services & Linked Data Services

The section references work having the common idea of interoperability of services that can be reused to build the concept of Experiment-as-a-Service (EaaS).

The concept of '**Linked Data Services (LIDS)**' for integrating data-providing services with Linked Data enables to automatically create links between services and existing data sets [Speiser et al. 2010] [Speiser et al. LIDS 2010].

The concept of '**Linked Open Services (LOS)**' has been designed by [Norton et al. 2010].

iServe[3] is a platform for the publication and discovery of services designed in the context of the EU project SOA4All [Pedrinaci et al. 2010]. Such services have been

[2] http://www.csiro.au/en/Research/D61/Areas/Robotics-and-autonomous-systems/Internet-of-Things/Phenonet

[3] http://iserve.kmi.open.ac.uk/

semantically annotated with the Minimal Service Model (MSM)[4], a common vocabulary to deal with the heterogeneity of diverse formalisms (e.g., OWL-S, WSMO, SAWSDL, WSMO-Lite for WSDL services, MicroSWMO, SA-REST for Web APIs). iServe uses the SAWSDL, WSMO_lite and hRESTs vocabularies. iServe uses Watson[5], an ontology search engine, to reduce integration issues through ontology reuse.

The SWEET[6] tool has also been developed in the context of SOA4All [Maleshkova et al. 2009]. SWEET stands for Semantic Web sErvices Editing Tool and supports users in creating semantic descriptions of RESTFul services based on the use of hRESTS (HTML for RESTful services) and MicroWSMO microformats which enable the creation of machine-readable service descriptions to encourage the reuse of services.

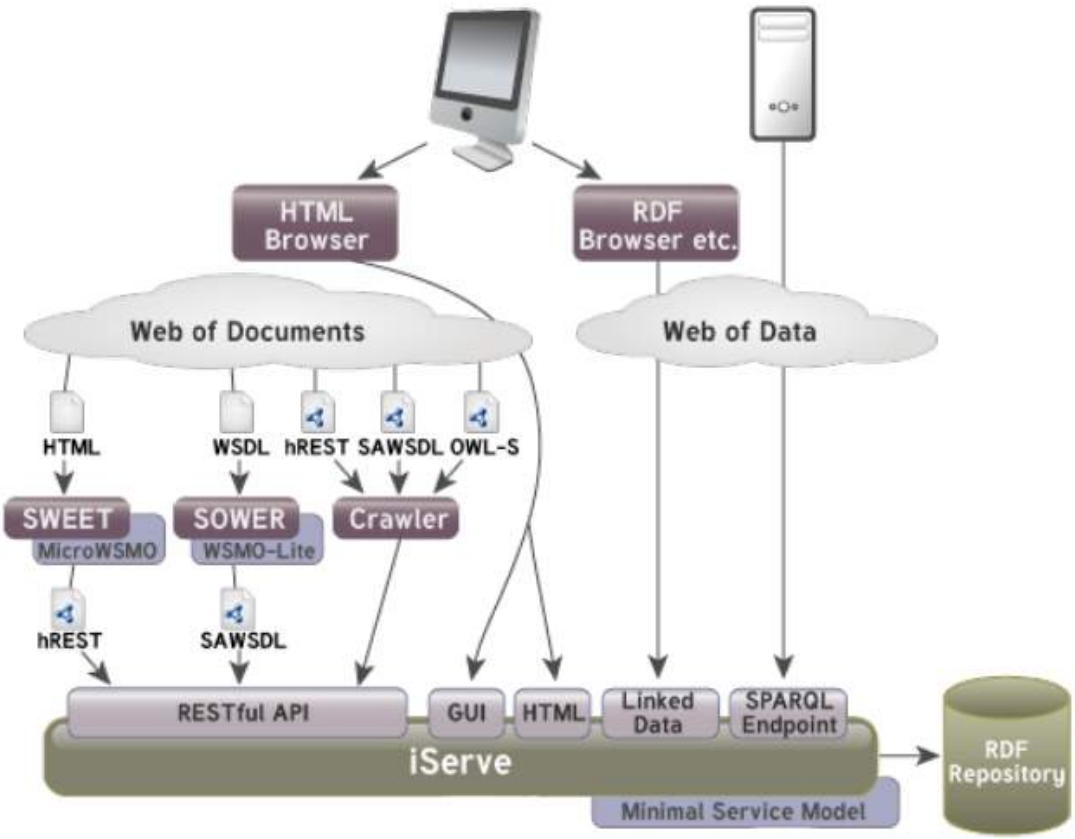

Figure 6. iServe's architecture [Pedrinaci et al. 2010]

[4] http://iserve.kmi.open.ac.uk/ns/msm/msm-2014-09-03.html

[5] http://watson.kmi.open.ac.uk/WatsonWUI/

[6] http://sweet.kmi.open.ac.uk/

2.1.3 Semantic Web Services

Semantic Web services [McIlraith et al. 2001] are the combination of two research fields: Semantic Web and Web Services. The main benefit of semantic web services is automating web services discovery, invocation, composition and execution monitoring.

Ontology Web Language for Services (OWL-S) is the most complete effort for describing semantic web services, the successor of DAML-S. OWL-S defines web services capabilities as follows [Martin et al. OWL-S 2004]:

- The *Service Profile* gives a high level description of a service and its provider.
- The *Service Model* describes the service's behaviour as a process. A process can be either composite or atomic.
- The *Service Grounding* defines mapping rules to link OWL-S.

OWL-S enables automatic discovery, invocation and composition of web services.

ODE SWS is a framework for designing and composing semantic web services [Gómez-Pérez et al., 2004]. The framework supports four ontologies:

- The Problem-Solving-Method (PSM) ontology to represent semantic web service's workflow.
- The Semantic Web Service ontology
- The Knowledge representation ontology
- The data type ontology

Web Service Description Language (WSDL) [Booth et al. WSDL 2007] is widely used to describe services or composite services and can be defined with service composition languages.

Web-of-Objects (WoO) provides a smart distributed application environment aggregating data from different domains that are currently isolated from each other [Ara et al., 2014]. WoO mainly focuses on the composition of services and registration of devices in IoT. It provides a semantic description and ontology that are used to search services. However, a semantic linking and querying engine are missing in their architecture to easily get enriched IoT data and high-level abstractions.

The **OWL-Ctx ontology** is an extension of the popular OWL-S ontology for describing services, provide composition of services and adapt it to context awareness [Furno et al. 2014]. On top of this ontology, they provide adaptation rules implemented according to the SWRL language.

Fujiti et al. design a semantics-based context-aware dynamic service composition framework which comprises [Fujii et al. 2009] :

- **Component Service Model with Semantics (CoSMoS)** is based on popular ontologies such as SOUPA [Chen et al. 2005] and GAS ontology [Christopoulou et al. 2005] to model contextual information. Both ontologies enable describing locations, device capabilities and user profiles. CoSMoS deals with rules (e.g., if I am in a meeting, do not use a cell phone or I am in a meeting on Monday between 9 a.m. and 10 a.m.)
- **Component Runtime Environment (CoRE)** is the component to infer new context (e.g., the user is in a meeting)
- **Semantic Graph based Service Composition (SeGSeC)** supports two kind of context-aware service composition: rule-based and learning-based.

Their framework is implemented with Java 1.6, WSDL4j to describe web service, Jena to build semantic web application, Weka for building decision trees (C4.5 algorithm). They provide two scenarios: (1) talk to someone using microphone, speaker and cell phone devices, and (2) watch movie depending on the context if he is alone or not.

An extension of OWL-S for modelling context-aware services and tasks has been designed, an essential step to provide context-aware service discovery and composition [Mokhtar et al. 2006]. The authors define their own context-aware service description to specify Quality of Service (QoS) and contextual condition aspects. The main novelty of their approach is to provide complex behaviors (i.e., workflows) for both services and tasks. Their web service composition approach is addressed from both a user-centric and service-centric point of views.

2.2 Domain Specific Language (DSL)

2.2.1 Definition of DSL

A Domain Specific Language (hereafter DSL) is a programming language designed specifically to express solutions to resolve problems in a specific domain. A DSL may be developed to meet needs of a particular platform, system, toolset, software problem, industry, or business challenge that cannot be effectively addressed by using mainstream languages.

A DSL can be contrasted with a general purpose language, such as C# and Java, which is designed to address a broad range of needs across the software development landscape. But in many cases, a subset of general purpose languages can be developed and implemented as a domain specific language to address a particular problem. For instance, Scala might be used to create a DSL for highly complex domain such as trading exchanges in the energy industry. Most software projects will incorporate a general language and several peripheral DSLs to add required functionality for various domains within the system.

DSL can be subdivided into three languages, i.e. Domain-Specific Modelling language (more generally, specification languages), Domain-Specific Markup language and Domain-Specific Programming language. Domain-Specific Modelling raises the level of abstraction beyond programming via specifying the solution directly using domain concepts. The final products will be generated from these high-level specifications.

Domain-Specific Modeling is a software engineering methodology for designing and developing system, for instance computer software. It involves systematic use of a DSL to represent various facets of a system.

Domain-Specific Modeling languages intend to support higher-level abstractions than general-purpose modelling language, so they require less effort and few low-level details to specify a given system.

2.2.2 OpenIoT DSL

OpenIoT service (experiments) management is facilitated by the usage of a descriptive language which is capable of hosting all the defined services from a specific User. This language is called **OpenIoT Service Description specification (OSDSpec)** and is depicted in Figure 7 below.

The OSDSpec instance is capable of describing in detail the Users request. The OSDSpec is used in the following methods of the OpenIoT Scheduler:

- “discoverService”, where it is used as input to describe the type of sensors the user wants to be involved in his/her sensor discovery service.
- “registerService”, where it is used as input to describe in detail the experiment which mainly includes:
    - The query execution schedule,
    - The sensor types which are involved in his/her query,
    - The mash-up libraries that are going to be used, and

- Directions on how to use the presentation libraries.

- "updateService", where it is used as input to alter a prior registered experiment.
- "getService", where it is used as an output and describes in detail a prior registered experiment.

As shown in Figure 7 below the OpenIoT Service Description specification is consisted of many **OpenIoT Application Model objects (OAMOs)** which hosts the Users Application description (complex services) which in its turn consists of multiple OpenIoT Sensor Model Objects (OSMO).

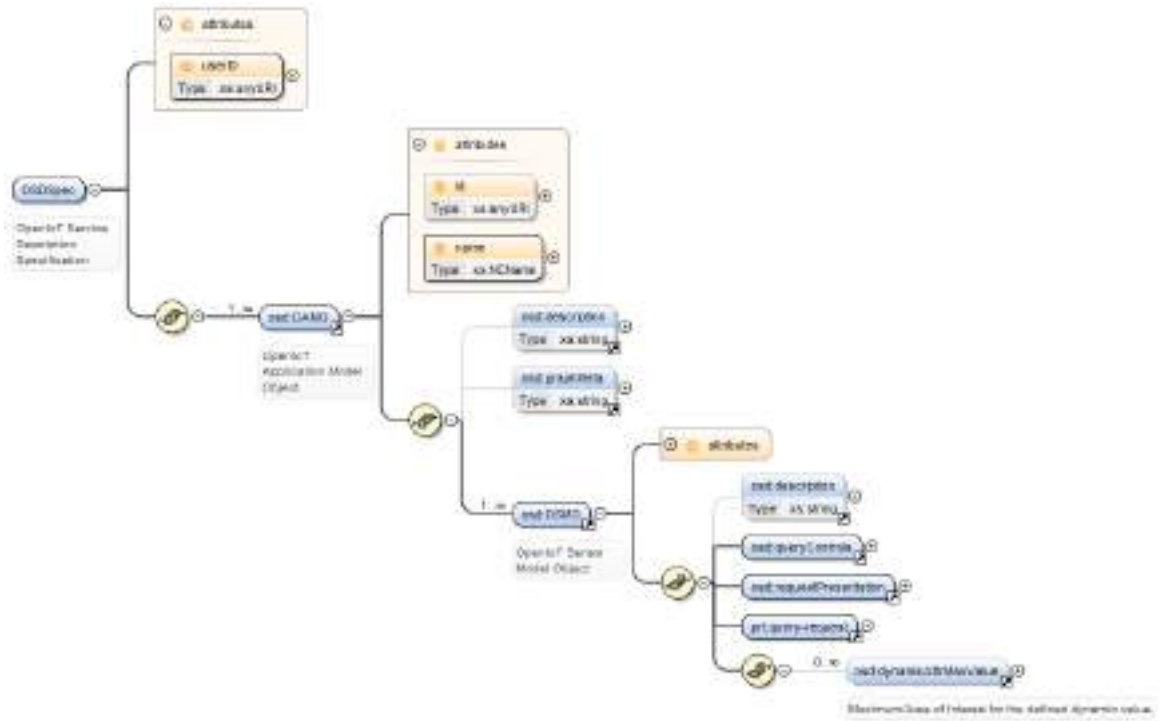

Figure 7 OSDSpec Schema graph

The **OpenIoT Sensor Model Object** (**OSMO**) is capable of hosting the required information to describe an experiment. Notably it includes a list of Semantic Sensor Network objects for describing the required sensors and a list of attributes to describe how these objects are going to be combined at the presentation tier. Figure 8 below depicts in detail all the contents of the OSMO object.

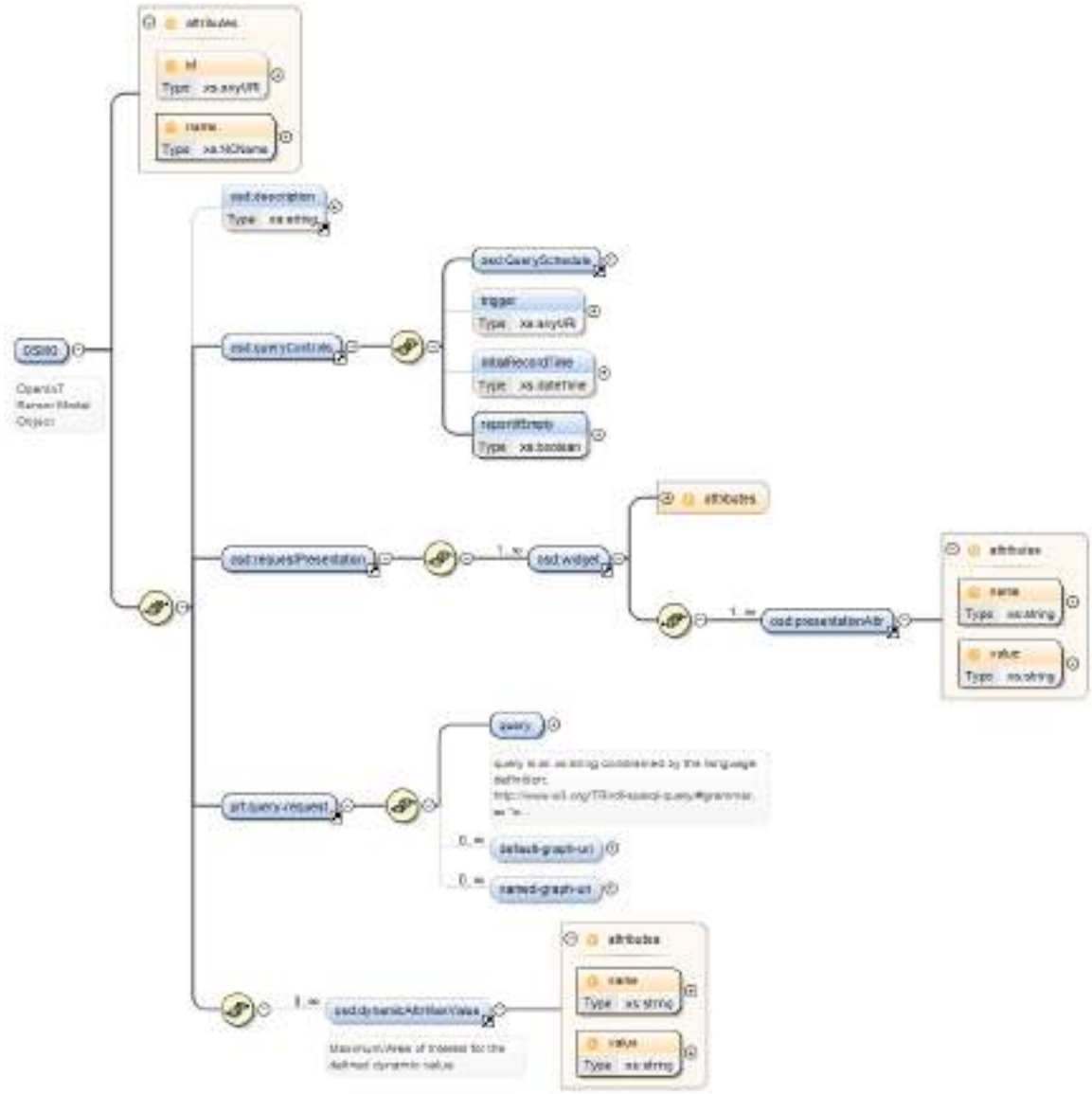

Figure 8 OSMO Schema graph

The main OSMO components include:

- The **QueryControls** which controls how the Service registration should be processed; in particular, it specifies the conditions under which the service should be invoked from the GUI (e.g., specifying a periodic schedule, defining specific visualizations etc).
- A **QuerySchedule** which may be defined to specify a periodic schedule for query execution for a specific subscription. Each field of QuerySchedule is a string that specifies a pattern for matching some part of the current time. The query will be executed each time the current date and time matches the specification in the QuerySchedule.
- The **PresentationAttr** which defines the preferred presentation GUI defined by the initial user at the service registration 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.

- The **RequestPresentationAttr** object which describes how the SSN objects are combined at the presentation tier.

More details for the OpenIoT DSL can be found at the OpenIoT deliverable D4.1 [Kefalakis et al. 2012]

2.2.3 SuperStreamCollider from OpenIoT

The SuperStreamCollider[7](SSC) is a stream-processing platform running in cloud environments and enabling the users to build complex mashups using semantically annotated Linked Streams and Linked Data [Quoc et al., 2012]. It exploits sensor data produced by the SmartSantander project, and semantically annotates it in real-time using the Linked Stream Middleware (LSM), a SPARQL extension to deal with real-time data. The cloud-based processing is performed in BonFire [Rahman et al., 2015]. SuperStreamCollider is based on the Virtual Wall API provided by the Fed4Fire project. It also reuses technologies provided by the Planet Lab project[8]. In the SuperStreamCollider-Fed4Fire experiment, a Hadoop cluster is used for easily retrieving IoT data among 2,000 SmartSantander sensor data sources such as humidity, air quality, noise pollution and parking status.

SuperStreamCollider provides a visual programming environment called SSC visual editor [Quoc et al., 2012], a web-based workflow editor for composing mashup data though drag & drop as depicted in Figure 9.

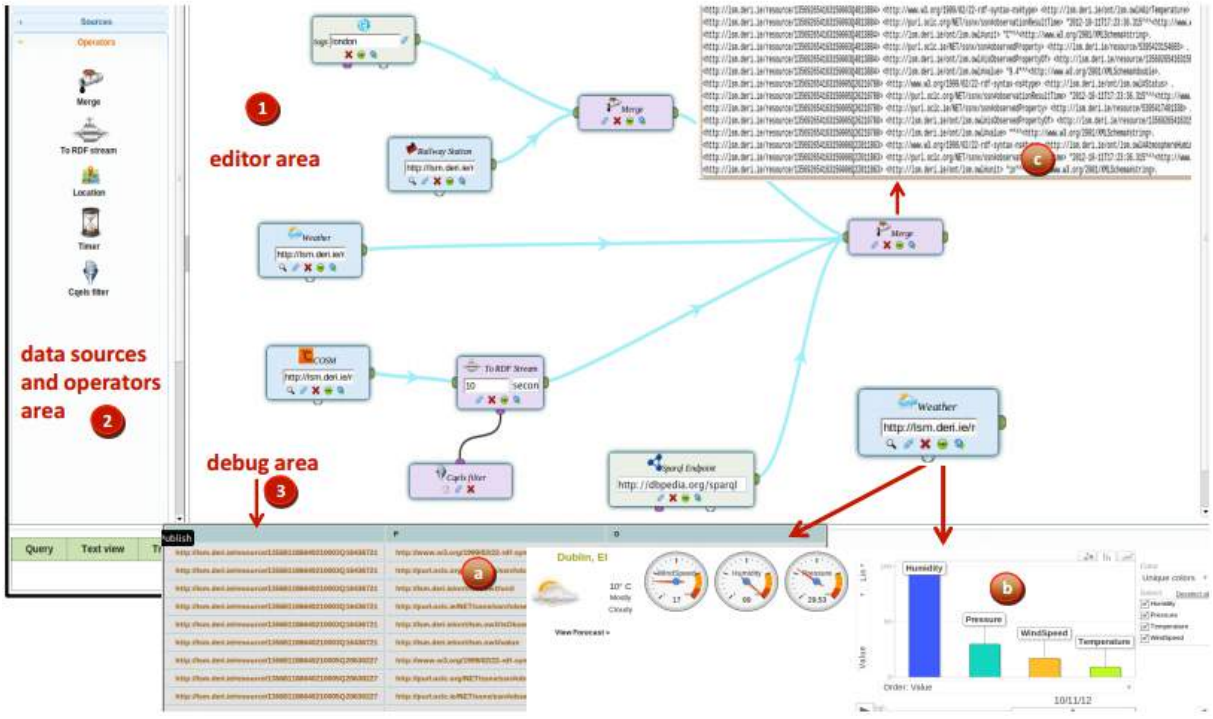

Figure 9. SuperStreamCollider visual editor [Quoc et al., 2012]

[7] http://superstreamcollider.org/

[8] https://www.planet-lab.org/

SuperStreamCollider also provides a CQELS/SPARQL query visual editor which supports users in creating, modifying and reusing CQELS/SPARQL queries using drag & drop technologies as depicted in Figure 10. The main benefit of this tool is reducing the effort of learning SPARQL and CQELS and speeding up the learning curve.

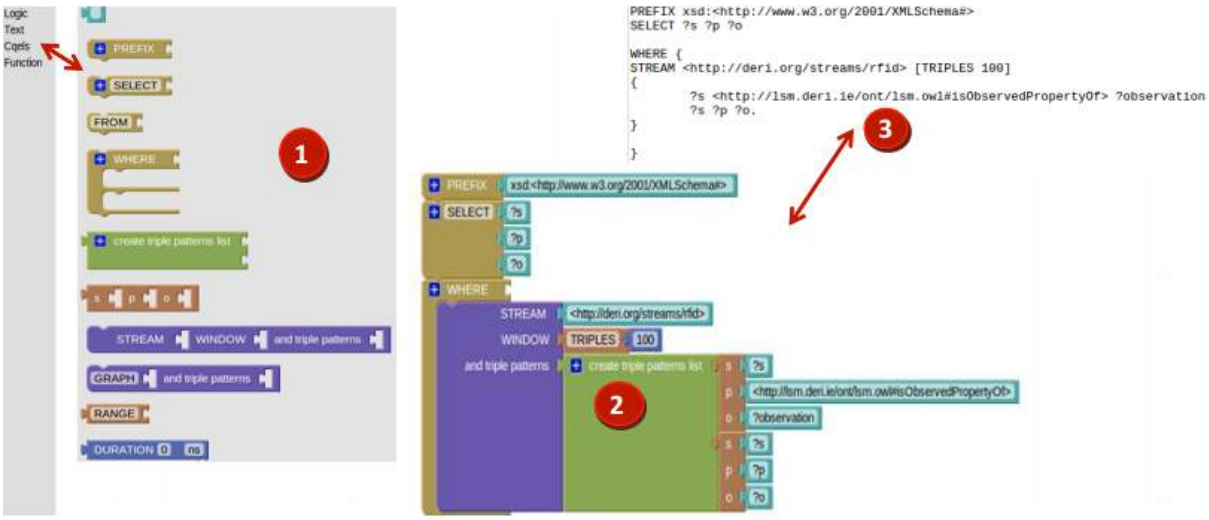

Figure 10. SuperStreamCollider, CQELS/SPARQL query visual editor [Quoc et al., 2012],

2.2.4 Fed4Fire

Fed4FIRE is an EU project, "A federation of testbed facilities is a collection of multiple independent testbeds that can be coordinated in different ways for the creation of rich, multi-functional environments for testing and experimentation; and has clear benefits for its main stakeholders – experimenters, and facility providers" [Ooteghem et al., 2014]. Fed4FIRE provides a common federation framework for future Internet Research and experimentation facilities. It enables experimenters to easily access heterogeneous testbeds (e.g., BonFIRE, PlanetLab and Virtual Wall) by using federated tools supporting experiment lifecycle experiment. Fed4FIRE comprises a set of technologies:

- **For resource description:** Resources are described using RSpec's and are grouped within Slices, i.e. requested virtual topologies. These RSpec's are arbitrary XML-based documents that define the offered (Advertisement RSpec), requested (Request RSpec), and allocated (Manifest RSpec) resources. A common denominator is the GENI RSpec v3, which includes definitions of Nodes and Links and can be extended by further XSD's. The formalization of resource descriptions are still the subject of current research. The latest approach include the OWL-based replacement developed within Fed4FIRE and published within the W3C Federated Infrastructures Community Group[9].

[9] https://www.w3.org/community/omn/

- **For resource discovery, selection, reservation, provisioning and release:** the XML-RPC-based Slice-based Federation Architecture (SFA) allows federation across facilities using TLS-based AuthN. SFA-aligned testbeds can be controlled by various SFA-compliant user tools. These tools include:
    - **Flack[10] (client GUI)**, developed within ProtoGENI. A Flash-based GUI used for user-friendly selection and provisioning of resources.
    - **Omn[11](client CLI)**, develop as a collaborative GENI effeort. A python-based reference implementation which also part of the GENI Control Framework (GCF) software package.
    - **jFed**[12] **(client GUI and CLI)**, developed by iMinds. A Java-based framework which contains automated testing tools, a probe for manual testing and an end-user experimentation-toolkit. It can be used to perform automated scenario tests.
    - **SFA Command-Line Interface (client CLI)**, called SFI and developed within PlanetLab and in its core used within MySlice (see below).
    - **MySlice**[13] **(user GUI),** developed by UPMC. An HTML-based portal deployed within Fed4FIRE[14] to graphically create slices, manage users, link to testbed documentation and join experiments. Its functionality can partly be compared with the jFed tool and both tools are tested to work with each other.
    - **SFAWrap**[15] **(provider AM)** developed by UPMC. An python-based AM to facilitate the task of testbed owners to provide an interface to their testbed if they are building a testbed from scratch, have an existing testbed but it is not yet federated via SFA or have a testbed that already offers an SFA interface.
    - **FITeagle[16] (provider AM)** developed by TUB. A Java-based AM to provide an SFA interface to new testbeds with a single command and functional prototype for semantic-based resource description (see W3C Federated Infrastructures Community Group).
- **For resource control (experimentation):** While SFA mainly addresses issues regarding the description, discovery and provisioning of resources, mechanisms are also needed to describe experiments and to control and monitor resources accordingly.

[10] http://protogeni.net/wiki/Flack

[11] http://trac.gpolab.bbn.com/gcf/wiki/Omni

[12] http://jfed.iminds.be/

[13] http://myslice.info/

[14] https://portal.fed4fire.eu

[15] http://sfawrap.info/

[16] http://fiteagle.org

- As a result, the **cOntrol and Management Framework (OMF)**[17] was developed and has been widely adopted to execute reproducible experiments. Workflows are described using the **OMF Experiment Description Language (OEDL)**, a Domain Specific Language (DSL) based on Ruby. An experimenter using an Experiment Controller (EC) to send the related messages and commands to a Resource Controller (RC). The communication is based on the Federated Resource Control Protocol (FRCP).
- Another EC, called **Network Experimentation Programming Interface (NEPI)**[18], allows complex experiments to be described and executed, has been enhanced to be compatible with FRCP. NEPI uses its own DSL to define workflows and supports combining resources from three different types of experimentation platforms: simulators, emulators, and real testbeds.

2.2.5 Ontologies for describing testbeds and services

As we will explain in section 2.2.8.4, a possible implementation of DSL is the Ontology Web Language (OWL) language, mainly used to design ontologies. For this reason, in this section, we review ontologies which could be relevant to design the EaaS model.

TaaSOR (Testbed-as-a-Service Ontology Repository) is an ontology for describing testbeds really promising for the FIESTA project [Tosic et al., 2012].

Ontology Web Language for Services (OWL-S) is the most complete effort for describing semantic web services, the successor of DAML-S [Martin et al., 2004] . OWL-S defines web services capabilities as follows (see Figure):

- What a service does? The *Service Profile* gives a high level description of a service and its provider.
- How the service works? The *Service Model* describes the service's behaviour as a process. A process can be either composite or atomic.
- How the service is implemented? The *Service Grounding* defines mapping rules to link OWL-S.
- The Service ontology links together the previous concepts: Service Profile, Service Model and Service Grounding.

OWL-S enables automatic discovery, invocation and composition of web services.

[17] https://omf.mytestbed.net

[18] http://nepi.inria.fr

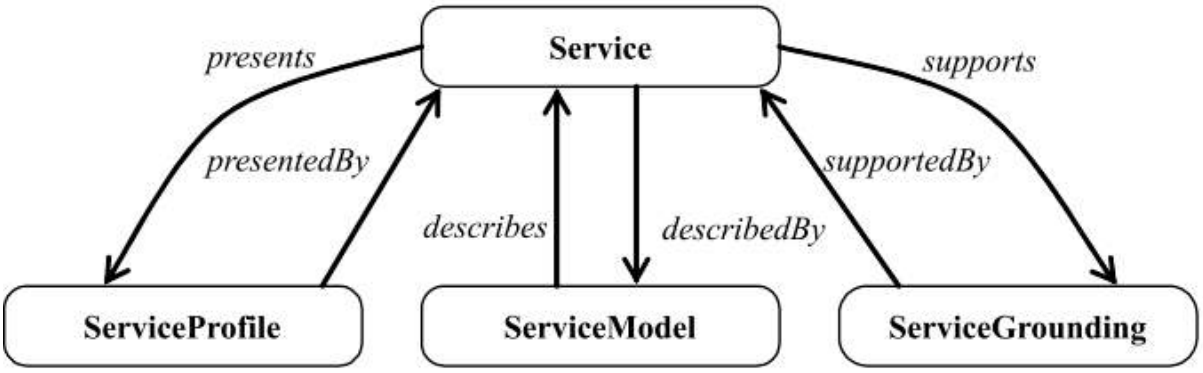

Figure 11. The OWL-S Service ontology [Sabou et al., 2006]

Minimal Service Model (MSM) ontology gives a generic service description based on WSMO-lite.

hREST specifically describes RESTful web services, it is based on WSMO-lite.

Lifecycle[19] is a simple vocabulary. That aims at describing a state machine, with states and transitions

Open-Multinet[20] **(OMN)** is a set of OWL-based ontologies to semantically describe federations of infrastructures, testbeds, and their resources including their status. They can be used within each phase of the experiment life cycle (describe, publish, discover, select, reserve, provision, monitor, control, and release). The ontologies incorporate existing work such as the OGF Network Mark-Up Language (NML), the Infrastructure and Network Description Language (INDL) or the Network Description Language based on the Web Ontology Language (NDL-OWL) and reuse existing ontologies such as Dublin Core (DC), Friend of a Friend (FOAF), Good Relations (GR), W3C Geo and W3C Time. The ontologies have mainly been developed and evaluated within the Fed4FIRE project and used within the FITeagle Aggregate Manager (AM). A demonstrator can be found at http://lod.fed4fire.eu. It can answer questions such as “Show me all 802.11 compatible nodes close to the Acropolis”, based on testbed information operated within FIRE and GENI.

2.2.6 Developing IoT applications

Patel et al. describe the challenge to ease application development dedicated to smart office and fire management IoT applications [Patel et al. 2011] [Patel et al. 2013]. They propose a tool to easily develop IoT applications, but the application developers still need to program the application logic layer and they do not explain the way to interpret sensor data. They discuss the need of common domain vocabularies.

However, their approach is not based on semantic web technologies and ontologies. They take into consideration actuators too. Their evaluation is based on two Eclipse

[19] vocabulary http://vocab.org/lifecycle/schema

[20] https://www.w3.org/community/omn/

plug-in: Metrics 1.3.6 and EclEmma to show that their tool reduce the development time. No demonstration is available and they do not provide end-user interactions.

2.2.7 Requirement of DSL for Modelling Experiments

2.2.7.1 Experiment Process Flow

In this section, we briefly introduce the process flow of experiments and capabilities implemented in each function block as follows according to Figure 12 and Figure 13 respectively.

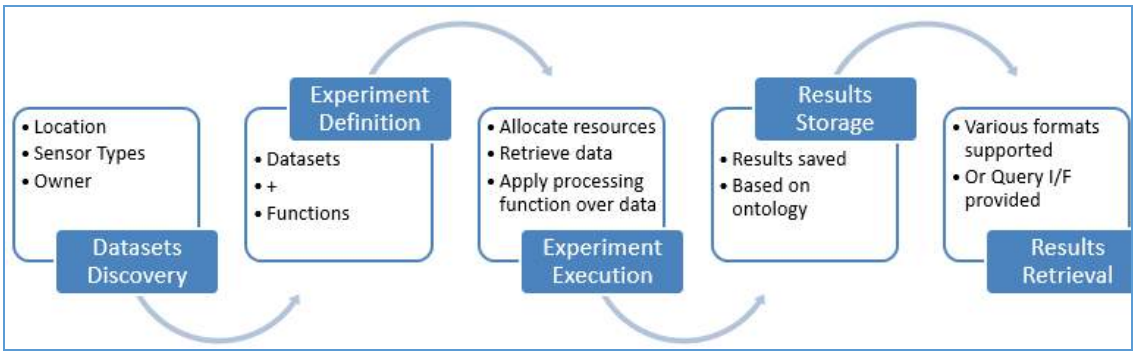

Figure 12. Process flow of experiments

Discovery available datasets	Experiment definition	Execute (save results to local database)	Monitor	Get final results

• Location(s)
• Sensor type(s)
	• Select datasets
	  • Location(s)
	  • Sensor type(s)
	  • Time interval (t_{start} - t_{finish})
	• Select processing function
	  • Predefined provided list of functions
	  • E.g. avg, stdev, regretion, k-mean
	• Define processing logic via a scripting language
		• Return experiment ID	• Fetch experiment metadata by ID	• CSV
				• XLS
				• ...

Figure 13. Capabilities associated with each process in an experiment

- Dataset Discovery retrieves with sensor types dataset and owner dataset, etc.
- Experiment Definition uses datasets specified in Dataset Discovery, It defines processing functions. For instance, average, STDEV etc. mathematical functions. Definition of processing logics via a scripting language
- Experiment Execution allocates requested resources, retrieve the data and apply processing functions over data and return experiment identifier.

- Result Storage stores the result of the function processing. The storage based on ontology.

- Result Retrieval supports various format for returned data.

2.2.7.2 Requirements Analysis for FIESTA-DSL

A generic reference model is required to enable the generation of different references architectures depending on domain-specific requirements.

According to Task 4.1 of FIESTA-IoT, a modelling language needs to be specified for modelling experiments associated with IoT data and resources, along with tools for specifying, parsing and enacting this language.

We summarize a few requirements for designing a DSL that suites requirements in FIESTA-IoT through analysing three proposed experiments from INRIA, NEC and UNICAN as follows:

- Shall support to modelling devices, including modelling monitoring and actuating devices, corresponding to the type and functionalities of each device;

- Shall support to modelling the statistical representation of the IoT data and resources;

- Shall enable to specify an unified access format for requested resources;

- Shall enable to specify a data resource and IoT device annotation model with the semantic information;

- Shall support to modelling parameters or datasets abstracted from functionalities of all participated testbeds, for instance, temperature and co2, so that experimenters are able to select one or several monitoring parameter(s). As a result, all data streams corresponding to the selected parameter(s) could be collected from all relevant testbeds and processed. The final result of execution of experiment will be responded to experimenters in a specific manner;

- Shall support data analysis techniques against additional data information for special experiment cases, for instance, for detection systems of industry area, *location* and *time* information becomes much more important than other application areas. In these cases, pattern mining approaches might be needed for big data analysis in order to prevent the occurrence of accidents.

More requirements might be derived according to process flow of experiment shown as Figure 12 and Figure 13.

2.2.8 Existing Domain Modelling languages

2.2.8.1 OCL

The Object Constraint Language (OCL) started as a complement of the UML notation with the goal to overcome the limitations of UML (and in general, any graphical notation) in terms of precisely specifying detailed aspects of a system design.

OCL is a rich language that offers predefined mechanisms for:

- Retrieving the values of an object
- Navigating through a set of related objects
- Iterating over collections of objects (e.g., for All, exists, select)

OCL includes a predefined standard library: set of types + operations on them

- Primitive types: Integer, Real, Boolean and String
- Collection types: Set, Bag, Ordered Set and Sequence
- Examples of operations: and, or, not (Boolean), +, -, >, < (Real and Integer), Union, size, includes, count and sum (Set).

Using OCL's tree-like description, feature model allows to describe mandatory and optional features of a subject, and they allow to specify alternative features as well conjunctive features.

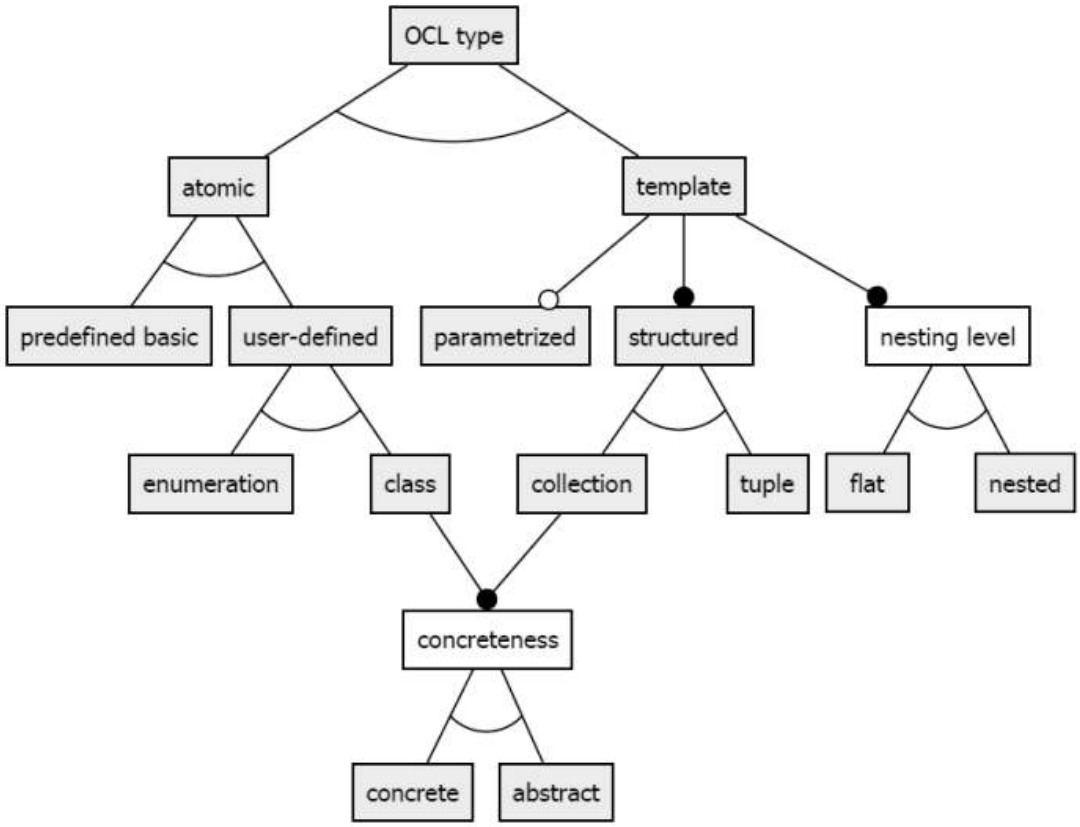

Figure 14. OCL Types as a Feature Model

2.2.8.2 ATL Transformation Language (ATL)

ATL is a model transformation language specified as both a meta-model and a textual concrete syntax. In the field of Model-Driven Engineering (MDE), ATL provides developers with a means to specify the way to produce a number of target models from a set of source models.

The ATL language is a hybrid of declarative and imperative programming. The preferred style of transformation writing is the declarative one: it enables to simply express mappings between the source and target model elements. However, ATL also provides imperative constructs in order to ease the specification of mappings that can hardly be expressed declaratively.

An ATL transformation program is composed of rules that define how source model elements are matched and navigated to create and initialize the elements of the target models. Besides basic model transformations, ATL defines an additional model querying facility that enables to specify requests onto models. ATL also allows code factorization through the definition of ATL libraries. Developed over the Eclipse platform, the ATL Integrated Development Environment (IDE) provides a number of standard development tools (syntax highlighting, debugger, etc.) that aim to ease the design of ATL transformations. The ATL development environment also offers a number of additional facilities dedicated to models and meta-models handling. These

features include a simple textual notation dedicated to the specification of meta-models, but also a number of standard bridges between common textual syntaxes and their corresponding model representations.

2.2.8.3 MiniUML

MiniUML is a subset of Universal Modelling Language, which can be conveniently specified using GraphViz graph description language.

GraphViz is somewhat simpler than full UML, yet it still allows for rich expression of many concepts used in software intensive systems.

Several notational conventions are used for denoting particular semantic meaning. Where possible, direct UML notation is used. It is of particular interest to object oriented database modelling. It is used by the WDBI project as a modelling language.

2.2.8.4 Web Ontology Language (OWL)

The Web Ontology Language (OWL) is designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics. OWL has three increasingly expressive sublanguages: OWL Lite, OWL DL and OWL Full.

OWL has been designed to meet this need for a Web Ontology Language. OWL is part of the growing stack of W3C recommendations related to the Semantic Web.

- XML provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents.

- XML Schema is a language for restricting the structure of XML documents and also extends XML with datatypes.

- RDF is a data model for objects ("resources") and relations between them, provides a simple semantics for this data model, and these data models can be represented in an XML syntax.

- RDF Schema is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalization-hierarchies of such properties and classes.

- OWL adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, and characteristics of properties (e.g. symmetry), and enumerated classes. Figure 15 shows the hierarchy of the OWL 2 RDF-Based Semantics.

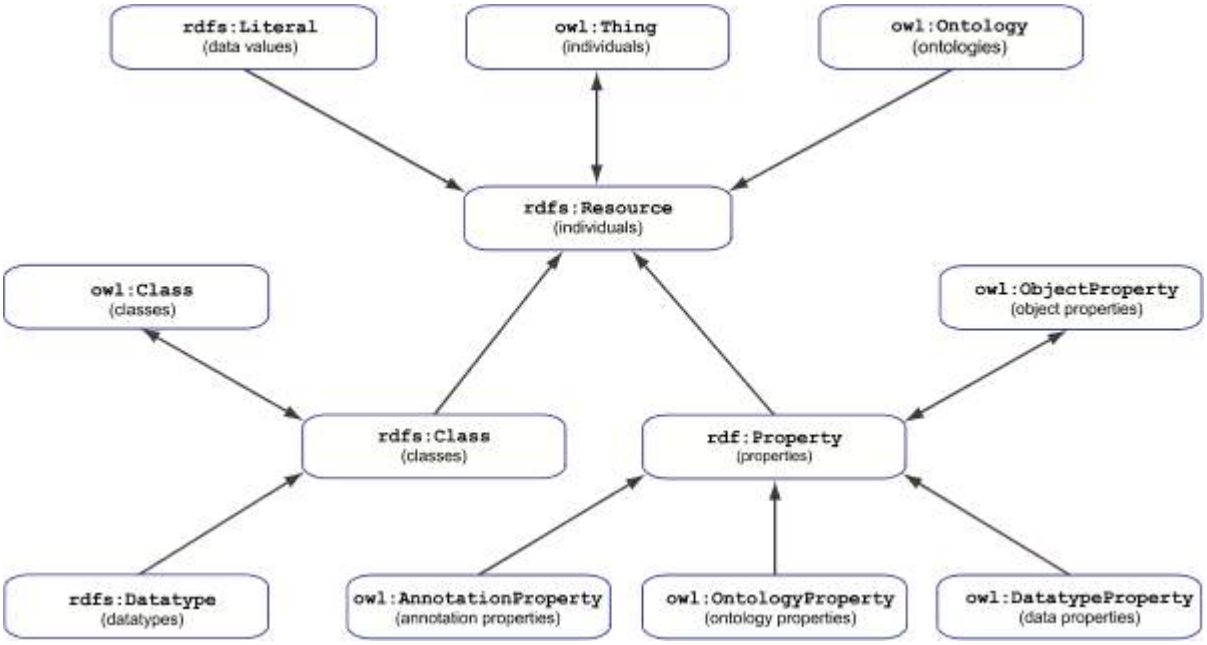

Figure 15. Parts Hierarchy of the OWL 2 RDF-Based Semantics

2.2.9 IDE Tools

2.2.9.1 Eclipse modelling project (XTEXT)

Xtext provides users with a set of domain-specific languages and modern APIs to describe the different aspects of users' programming language shown as Figure 16. The compiler components of your language are independent of Eclipse or OSGi and can be used in any Java environment. They include such things as the parser, the type-safe abstract syntax tree (AST), the serializer and code formatter, the scoping framework and the linking, compiler checks and static analysis aka validation and last but not least a code generator or interpreter. These runtime components integrate with and are based on the Eclipse Modelling Framework (EMF), which effectively allows users to use Xtext together with other EMF frameworks like for instance the Graphical Modelling Project GMF.

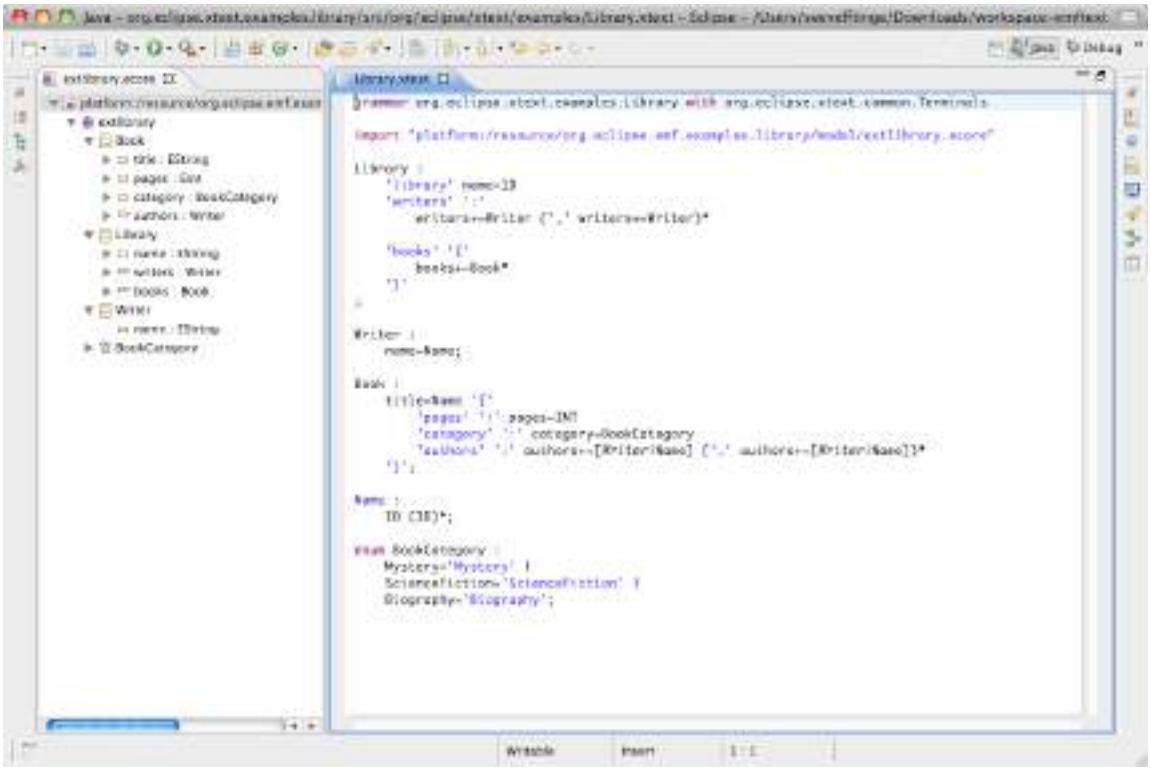

Figure 16. XTEXT Eclipse IDE Capture

2.2.9.2 Modelling SDK for Visual Studio-Domain-Specific Languages (MSDK)

Modelling SDK for Visual Studio-Domain–Specific Languages (MSDK) lets users develop a model quickly in the form of a domain-specific language (DSL). Users begin by using a specialized editor to define a schema or abstract syntax together with a graphical notation as shown in Figure 17. From this definition, MSDK generates:

- A model implementation with a strongly-typed API that runs in a transaction-based store,
- A tree-based explorer,
- A graphical editor in which users can view the model or parts of it that you define,
- Serialization methods that save your models in readable XML, and
- Facilities for generating program code and other artefacts using text templates.

Users can also customize and extend all of these features. The extensions are integrated in such a way that users can still update the DSL definition and re-generate the features without losing their extensions.

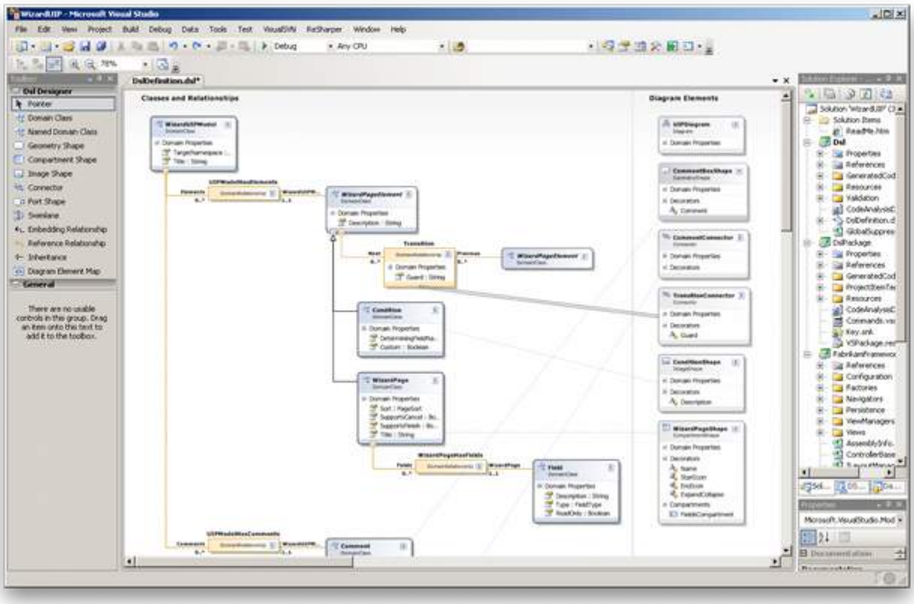

Figure 17. MSDK Editor Capture

Table 4 compares the different implementations of DSLs. In the context of FIESTA-IoT, we would prefer OWL to implement the DSL.

Table 4. Comparison among existing DSLs

	OWL	OCL	ATL
Notions	Axioms, Entities, Expressions	Self, UML Context, Invariants, Pre and Post-conditions	Header section (module), Import section (uses), Helpers section (helper), Rules (rules)
Data Types	Real, Decimal, Integers, Floating-Point numbers, Strings, Boolean Values, Binary Data, IRIs, Time Instants, XML Literals	Basic, UML Model, Enumeration, Let Expressions, Definition Expressions	Primitive data types, Operations on collection, Sequence, set, Ordered, Bag, Iterating over Collections, Enumeration, Tuple, Map.
Object Properties	topObjectProperty and bottomObjectProperty	Attribute, Association End, Operation with isQuery being true, Method With isQuery being true.	Matched Rules and Called Rules
Abstract Syntax	**axiom** ::= 'Class(' **classID** ['Deprecated'] **modality** { **annotation** | { **super** } ')' **modality** ::= 'complete' | 'partial' **super** ::= **classID** | **restriction**	context BagType inv: BagType.allInstances()->forAll(b | self.elementType.conformsTo(b.elementType) implies self.conformsTo(b))	module module_name; create output_models [from|refining] input_models; uses extensionless_library_file_name; helper context integer def : max(x : Integer) : Integer = ...; rule Author { from a : MMAuthor!Author to p : MMPerson!Person (name <- a.name, surname <- a.surname) }

2.3 Data worflow

2.3.1 Survey workflow editors: Google Blockly, NodeRed

Google Blockly is a library for building visual programming editors, Google Blockly is designed to easily install into one web application. Using Google Blockly the user will drag and drop blocks which are code minutes and by arranging these blocks the user generates the code which reflects the order of block arrangement. From the application's point of view Google Blockly is just a test area where the user types syntactically preferred languages JavaScript, Python, PHP, Dart or any other language. There are some well build visual programming editors which are using Google Blockly as their core like Wyliodrin. The following figure illustrates the overview of Google Blockly.

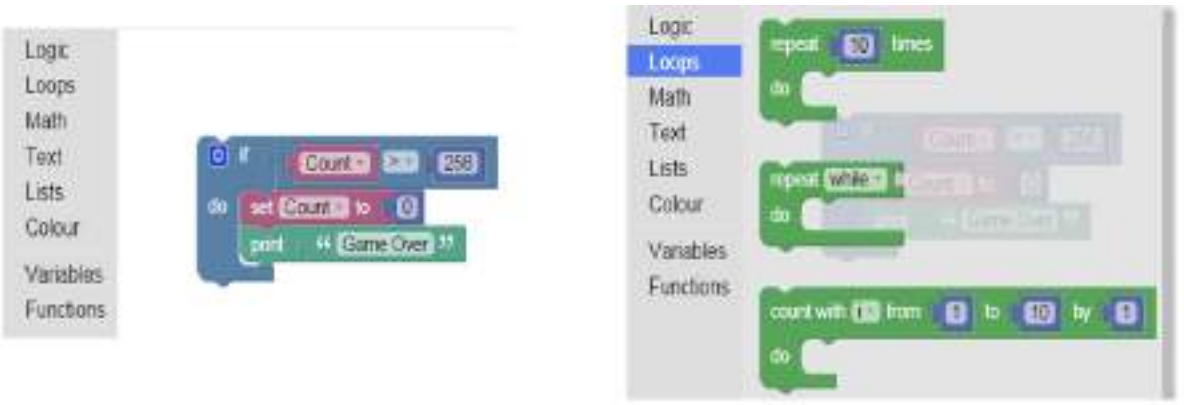

Figure 18. Google Blockly Overview

Node Red is a visual programming editor which provides a browser-based flow editor. In Node Red every node is a code minute and these nodes are wired together to generate a complete flow, this flow represent a whole experiment or program. The following figure illustrates the overview of Node Red.

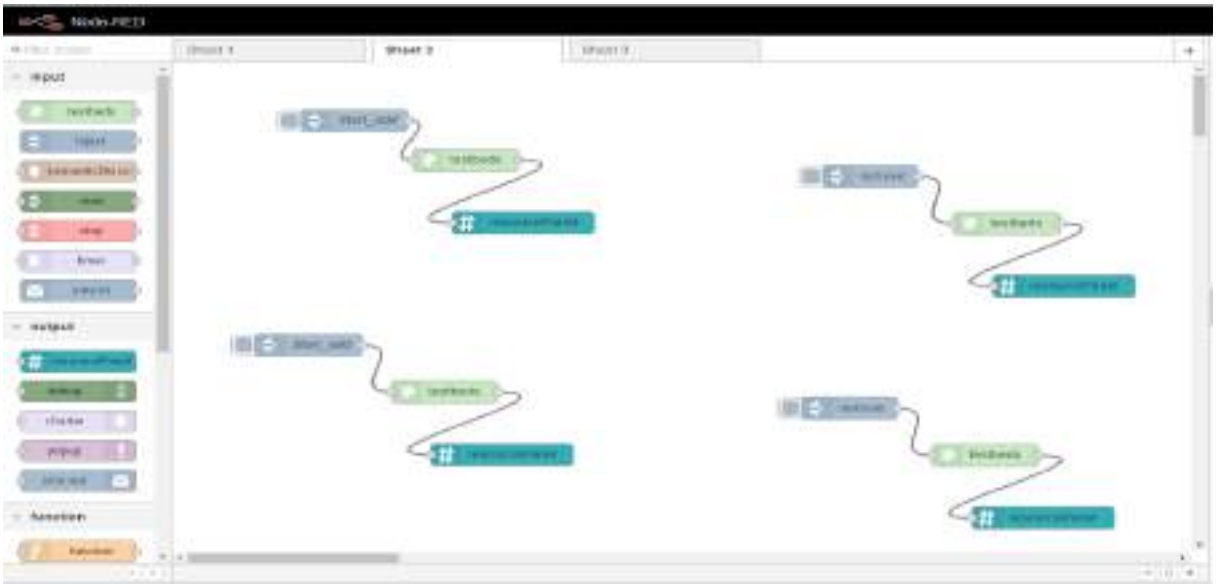

Figure 19. Node Red Overview

Both Google Blockly and Node Red are efficient in the area of visual programming and Node Red is used as the target editor because of the huge library, ease of use and complementing FIESTA-IoT scenario more than Google Blockly. The following table illustrates the comparison between Google Blockly and Node Red.

Table 5. Comparison between Google Blockly and Node Red

Features	Google Blockly	Node Red
Building blocks or nodes	Takes more time.	**Relatively less time.**
Library	Less amount of open sourced built libraries are available.	**Huge amount of libraries are available through node package manager.**
Programming complexity	Can build simple to complex program depending on the available blocks.	**Can build simple to complex program depending on the available nodes.**
Tools	The tools available to build a custom block are limited and primitive.	**Any node.js editor can be used.**
Program Flow	Depends on the arrangement of the blocks.	**Flows in a straight wire based execution order.**
Language	Java, Python, PhP	**Node JS and HTML**
Output	Java, Python, PhP	**Json**

2.3.2 Linked Data Architecture

A **Linked Data Application Architecture (LDAA)** is designed to ease data integration for the life sciences [Groth et al. 2014]. The platform called Open PHACTS Discovery Platform implements this architecture to combine Linked Data and APIs to enable the quick development of applications exploiting multiple datasets. The concept of “pay-as-you-go” is introduced

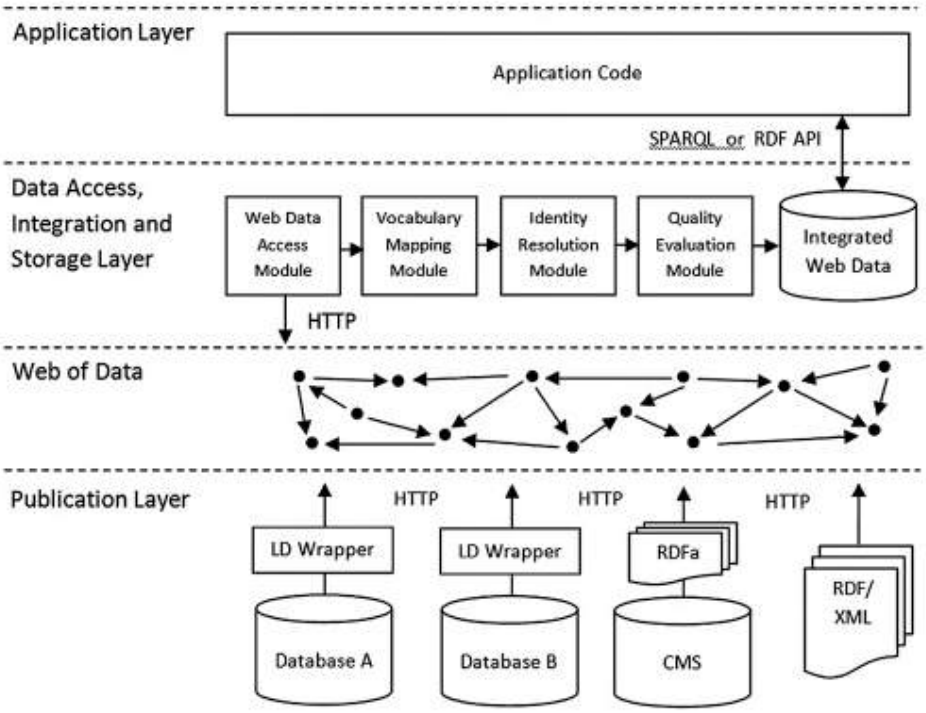

Figure 20. Linked Data Application Architecture (LDAA) [Groth et al. 2014]

Linked Sensor Middleware (LSM) is an architecture focused on dealing with real-time data produced by devices such as sensors [Le-Phuoc et al. 2014].

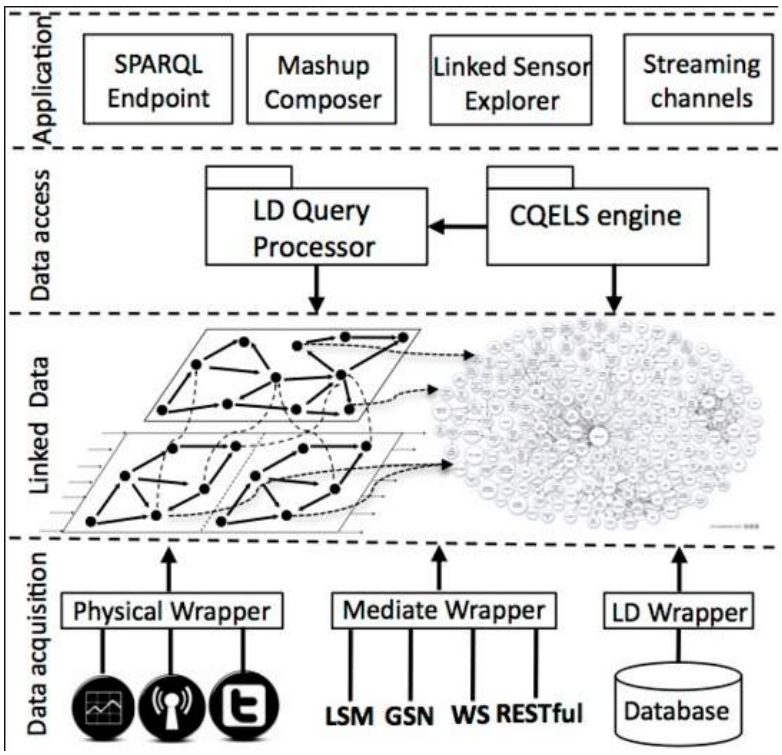

Figure 21. LSN architecture [Le-Phuoc et al. 2014]

Both architectures demonstrate the need to have a unified architecture to process the data, whatever the domain life sciences or Internet of Things.

2.4 Limitations of existing work

We analyse the following limitations in the literature that we studied:

- The lack of definition of experiment as a service with a concrete implementation.
- Perera et al. explain as a future work to address security issues, reasoning and real-time aspects. Such limitations are addressed within the FIESTA-IoT project. The security aspect is addressed in Task 4.2, the real-time aspect in Work Package 3, and reasoning in Task 3.4.
- Guinard et al. do not integrate semantic web technologies to their work.
- Fed4Fire is not applied to IoT. The federation of testbed could be reused in FIESTA-IoT.
- Fed4Fire does not provide a way to unify measurement/IoT data coming from heterogeneous testbeds.
- Fed4Fire does not provide API to easily get access to measurement data.
- Fed4Fire does not add value to measurement data (e.g., deduce new information).

Based on the literature study of concepts such as Sensing-as-a-Service, Linked Open Services and Linked Data Services, in the context of FIESTA-IoT, an architecture, called Meta-Cloud Architecture, is designed to deal with interoperability of data and services which is explained in the Section 3.

The idea of DSLs and ontologies in order to design the Experiment as a Service (EaaS) model is reused in Section 4. Section 5 and Section 6 explain the API specification and the prototype of the Eaas Model. Finally, to design the experiment workflow within FIESTA-IoT, the NodeRed tool has been chosen to automatically generate the DSL which is explained in Section 7.

3 FIESTA-IOT WP4 META-CLOUD

This section introduces the meta-cloud framework, more precisely, the methodology and the framework architecture to handle data produced by testbeds from a data management point of view. The main purpose of the methodology and architecture is to deduce meaningful information from data in order to design smart experiments. Finally, experiments workflows are explained.

3.1 Methodology

We design the SEG 3.0 methodology, SEG stands for SEGmentation and 3.0 for the Web 3.0 or Web of data. SEG 3.0 methodology has been mainly designed to analyze data to produce meaningful information and to specialize the methodology for IoT experimentation/interoperability. SEG 3.0 has been build based on our previous experience when designing a semantic engine for IoT [Gyrard et al. DataWorkflow 2014] [Gyrard et al. SemanticEngine 2015]. SEG 3.0 is not described in detail but SEG 3.0 is available in [Gyrard et al. SEG 3.0 Methodology 2016] [Gyrard et al. WebKnowledge 2016]. This methodology has been constantly improved and updated.

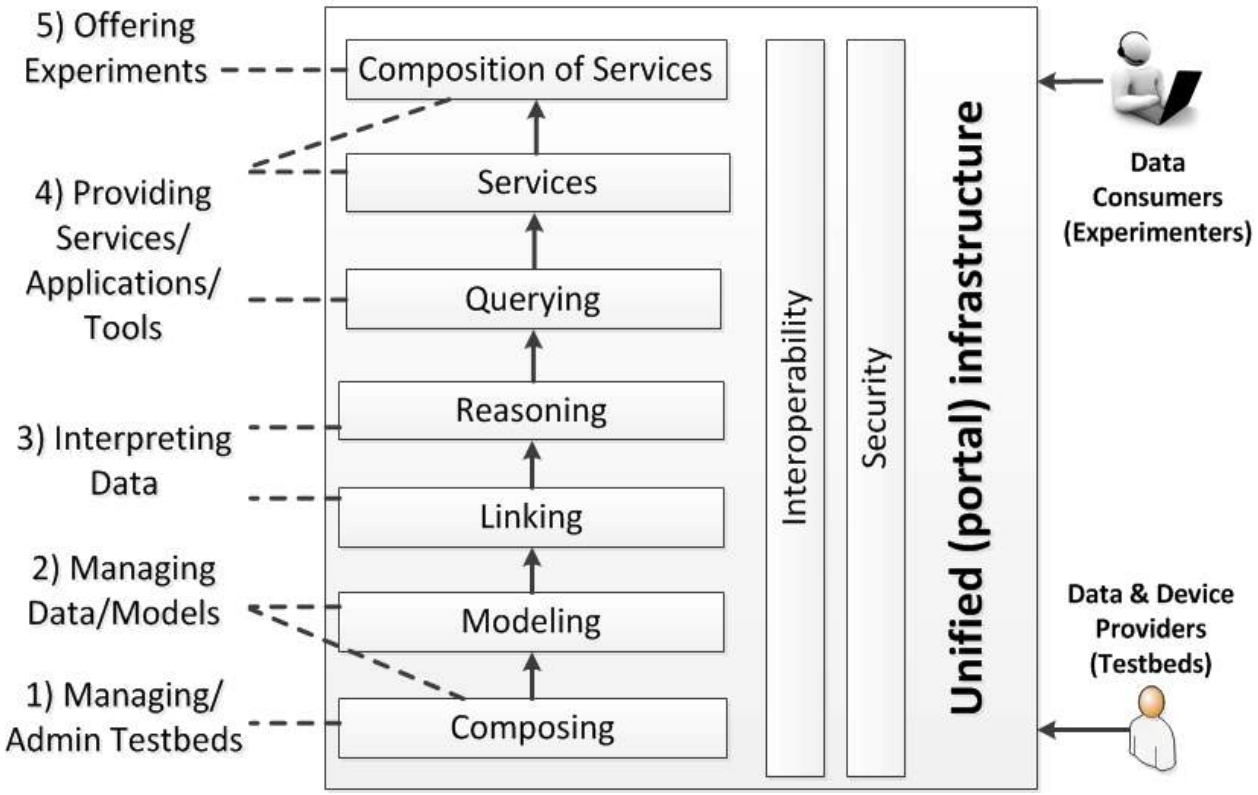

Figure 22. Methodology used to design the meta-cloud

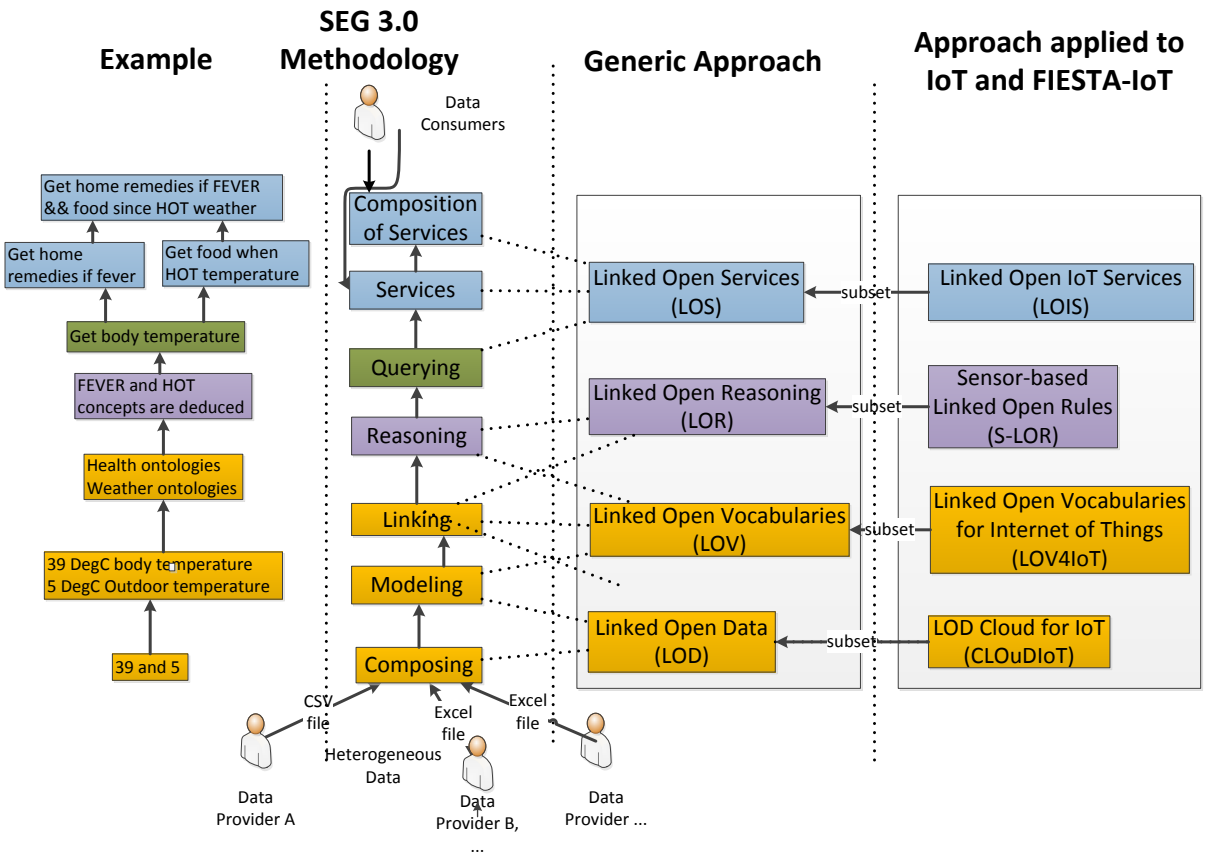

Figure 23. The SEG 3.0 methodology for building experiments ensuring Semantic Interoperability from data providers to data consumers

Different processes and steps are required to combine data from heterogeneous sources to build innovative and interoperable applications as depicted in Figure 23 Extensions of SEG 3.0 for IoT Experimentation comprises the following steps:

- **Composing Experiments** enables unifying heterogeneous data coming from different testbeds and using different data formats (e.g., CSV, Excel) or different terms (e.g, temp or temperature). It requires a common dictionary to unify terms employed to describe data. For instance, the M3 language[21] provides a nomenclature to unify sensor data.
- **Modeling Experiments** enables semantically annotating data with semantic web technologies (e.g., RDF, RDFS and OWL). This step employs models/vocabularies/ontologies to unify data, an intermediary step for the following experiment processes. The M3 ontology is the implementation of the M3 language, both used to unify semantic sensor data [Gyrard et al. oneM2MLanguage 2015].
- **Linking Experimentation** enables enriching data with other RDF datasets to get additional information. It exploits the idea of Linked Data and Linked Vocabularies.

[21] http://sensormeasurement.appspot.com/documentation/NomenclatureSensorData.pdf

- **Reasoning over experiments** enables updating the database/triple store with additional triples for instance by using reasoning engine (e.g., Jena rule-based inference engine) to deduce meaningful information from sensor data. It exploits the idea of Linked Rules.
- **Querying for Experimentation** enables querying RDF datasets through the SPARQL language based on ontologies used in the previous steps. It is an essential step to get data and build end-users services/applications.
- **Experimental Services** enables providing access to smarter data to end-users. The data is available through interoperable APIs or web services (e.g., RESTful web services). Such web services returns the result provided by the SPARQL query engine.
- **Composition of services** enables building complex applications by composing several services together. It can be achieved through the use of RESTful web service principles or semantic web services (e.g. OWL-S).

The SEG 3.0 methodology encourages the vision to enhance semantic interoperability from data to end-users applications which is inspired from the ``sharing and reusing''-based approach as depicted in Figure 23 the extension towards IoT experimentation comprises:

- **Linked Open Data (LOD)**, an approach to share and reuse data. It is used within the *Linking* step provided by SEG 3.0.
- **Linked Open Vocabularies (LOV)**, an approach to share and reuse models/vocabularies/ontologies. We extended this approach for IoT, called *Linked Open Vocabularies for Internet of Things (LOV4IoT)* [Gyrard et al. LOV4IoT V2 2016]. LOV and LOV4IoT are used within the *Modeling* step provided by SEG 3.0.
- **Linked Open Reasoning (LOR)**, an approach to share and reuse the way to interpret data to deduce meaningful information. LOR is an ongoing extension of Sensor-based Linked Open Rules (S-LOR) [Gyrard et al. S-LOR 2014], a dataset of interoperable logical rules to interpret IoT data. They are used within the *Reasoning* step provided by SEG 3.0.
- **Linked Open Services (LOS)**, an approach to share and reuse interoperable services/applications. It is used within the *Querying*, *Services* and *Composition of Services* steps provided by SEG 3.0. Semantic Web of Things generator is a first implemented approach towards this interoperability of IoT applications [Gyrard et al. SWoTGenerator 2015].

The main novelty of SEG 3.0 and its application on IoT Experimentation is not only sharing and reusing data, but sharing and reusing the entire chain from Linked Open Data (LOD) to Linked Open Services (LOS) to enrich data, more precisely, models, reasoning mechanisms and services associated to the data. SEG 3.0 is also envisioning, enriching and extending those approaches to apply it to IoT, detailed description is available in [Gyrard et al. WebKnowledge 2016] [Gyrard et al. SEG 3.0 Methodology 2016].

3.2 Architecture

The current architecture of the designed Meta-Cloud framework, an implementation of the SEG 3.0 methodology, is depicted in Figure 24. Data consumers (e.g., experimenters) can access the Meta-Cloud through a Graphical User Interface (GUI) or directly through web services (e.g., RESTful) or Application Programming Interfaces (APIs). The GUI eases the interaction between the data consumers and the Meta-Cloud framework by querying the web services and parsing the results.

In FIESTA-IoT, we integrated the proposed SEG 3.0 methodology. FIESTA-IoT Meta Cloud comprises the following components:

- **Semantic Annotator** enables unifying IoT data to deal with heterogeneous devices provided by testbeds (e.g., smart cities). It corresponds to the *Composing* step from SEG 3.0. The main benefit of using the M3-lite taxonomy is generating unified data. Two options are possible to be compliant with the M3-lite taxonomy: 1) either data descriptions already follow the M3-lite taxonomy, it just needs to semantically annotate the data according to the right namespaces, 2) the terms provided by the M3-lite taxonomy are not followed, a simple algorithm is required (string comparison and synonyms detection).
- **Reasoning engine** enables deducing meaningful information from sensor data. For instance the Jena inference engine can be executed to updates the data repository with more triples. It corresponds to the *Reasoning* step from SEG 3.0.
- **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.
- **Meta-cloud data endpoint** will provide access to semantically annotated data (e.g., SPARQL endpoint, web services).
- **Registry** enables registering a new resource, testbed, service or experiment. It is included within the *Service* step from SEG 3.0.
- **Discovery** enables discovering already registered resources, testbeds, services or experiments. It is included within the *Service* step from SEG 3.0.
- **Repositories**.
- Application Programming Interfaces (APIs) query semantic repositories through queries (e.g., SPARQL). There are five semantic repositories:
    - **Testbed and Resource Repository** stores the resources e.g., Thermometer device) employed in testbeds (e.g., smart city, smart office).
    - **Data Repository** stores semantic sensor data semantically annotated with ontologies explained below. .
    - **Ontology Repository** stores the ontologies and taxonomies, more precisely: experiment, IoT-lite, QU-rec20, W3C SSN and M3-lite taxonomy.

- **Rule Repository** stores logical rules (e.g., Jena rules) to deduce meaningful information from IoT/WoT data (e.g., IF Precipitation > 50 mm/h THEN Heavy Rain).
- **Experiment and Service Repository** stores semantic descriptions of experiments compliant with the experiment ontology explained below. It also stores various services descriptions to get access to data (e.g., SPARQL endpoints).

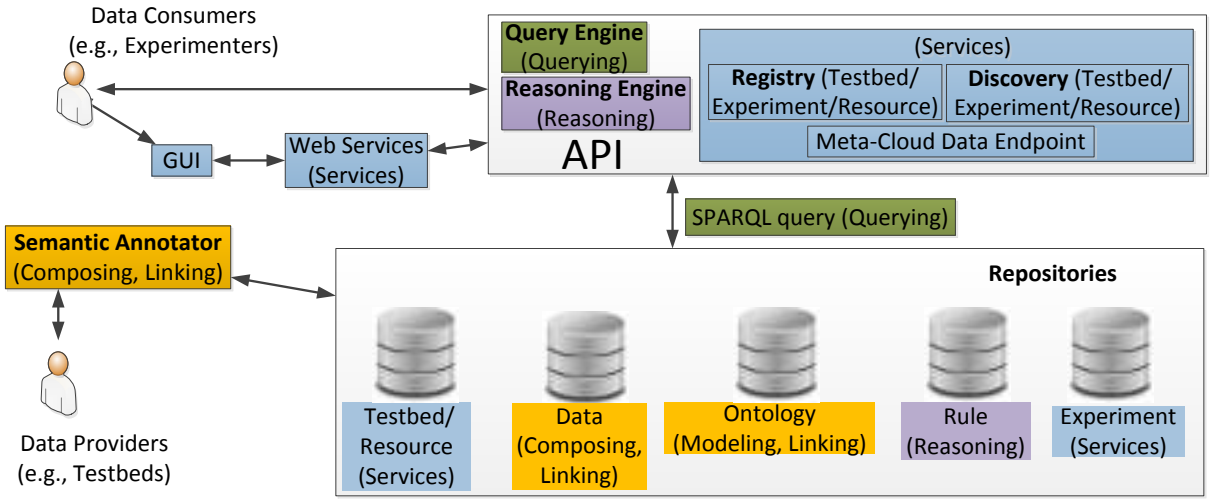

Figure 24. FIESTA WP4 Meta-Cloud architecture

3.3 Experiment Workflow

We describe in this section experiment workflows.

3.3.1 Sample Workflow

In this section we present an illustrative workflow that describes 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 25 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.

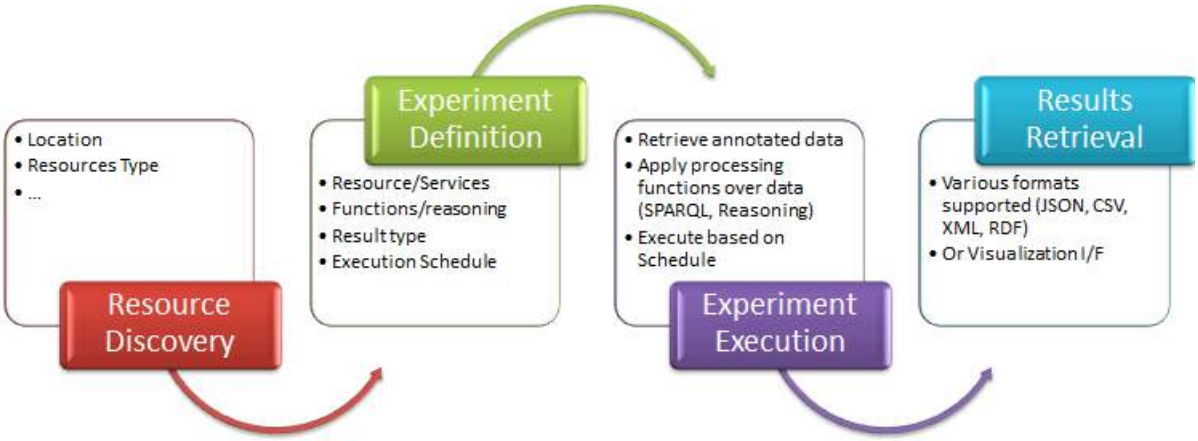

Figure 25. Experiment step-by-step procedure

- **PHASE 0 "Experimenter & Experimenter registration"**. Aside the specification and definition of the experiment per se, external users must sign up into 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 [FIESTA-IoT D2.4, 2015], 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 D4.4.1.

- 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 [FIESTA-IoT D4.2, 2016]). 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 even increase as long as new testbeds are fostered throughout the next months), 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*, through the specification of the DSL. 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[22] to retrieve the data. 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 [FIESTA-IoT D3.1, 2016]). 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

[22] The reader might refer to [FIESTA-IoT D2.4, 2016] in order to receive further indications about the system use cases that allow these features to be doable within the architecture.

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 D3.3.1 [FIESTA-IoT D3.3.1, 2016].

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 D4.4.1 [FIESTA-IoT D4.4.1, 2016].

- **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[23], 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-

[23] For more information about these features, please refer to [FIESTA-IoT D4.3, 2016]

shelf elements provided by the platform (or other added-value service providers who want to share their stuff with the project).

3.3.2 Crowdsensing Workflow

Asking crowds is a way to get information about the environment. Crowdsensing or participatory sensing is one of the techniques whereby users can provide information about the environment by participating in the experiment via their mobile phones.

The FIESTA-IoT experiment workflow mostly remains the same with respect to the DSL creation for the experiment. However, as users are participating they should be able to provide data using some mechanism to FIESTA-IoT. In FIESTA-IoT, such users would be associated to some testbed. As crowdsensing usually is accompanied by personal information, it is necessary for the testbeds to notify users if they want to make the user data public and then user should decide if they really want their data to be public.

One workflow, where users can be asked to participate in making their data public and available for experimentation via FIESTA-IoT (consider that a user installs a participatory sensing application provided by the testbed on their mobile devices and uses the application to provide their data to the testbed data store):

1. FIESTA-IoT platform administrator registers with the testbed FIESTA-IoT platform information via a specific registration page provided by the testbed. A successful registration returns an ID. This is used by the testbed to associate users that register to the testbed and want to get associated to the FIESTA-IoT Platform. This ID is a one time unique `ID`.
2. FIESTA-IoT then disseminates this `ID` to the user either already associated with the testbed or would be new users of the participatory sensing application of the testbed.
3. Testbed creates a form and specify in the form what data of the user is made public. Using this form the users can specify if they want to be associated to a specific `ID`.
4. After the user has specified `ID`, testbed makes only specified user data available only to the creator of the `ID` (FIESTA-IoT in our case, see Figure 22). Of course this step would also be using authentication of FIESTA-IoT on the testbed side.
5. Along with user's authorization, a user can anytime specify to not be associated to any specific `ID`, to this when a user would dissociate them from a `ID`, the testbed should not send the user data anymore to creator of the `ID` (FIESTA-IoT in our case).

This workflow is currently under implementation within one of the partner testbed (other than the 4 "in house" testbeds) that has crowdsensing information. Such workflow would enable crowdsensing and living labs approach.

Nevertheless this is one workflow, where users can join the testbed and their data can be made available to FIESTA-IoT. Some testbeds might follow different workflows. These approaches depend on what is the level of user involvement and what personal data is collected. If no personal data is collected then there is no need for such workflow. Testbeds can decide to make such information public. However, testbeds should just notify users.

Currently, SmartSantander is the only “in house” testbed that has participatory sensing Information [FIESTA-IoT D2.2, 2015]. The discussion is ongoing to decide what and how participatory data from the SmartSantander testbed would be made available to FIESTA-IoT and how such information would be stored in the FIESTA-IoT meta cloud data repository. Such information could be found in forthcoming deliverable 3.1.2 [FIESTA-IoT D3.1, 2016].

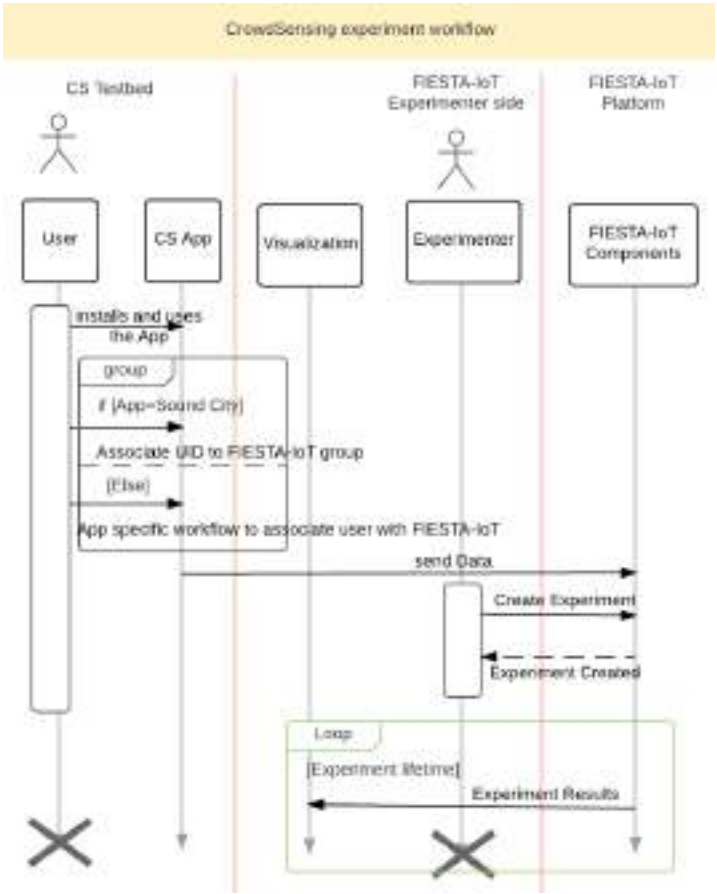

Figure 26: Sequence diagram for crowdsourcing experiment.

3.3.3 Data Workflow

We have designed a data workflow to assist users/experimenters in building experiments, as depicted in Figure 27. It comprises the following steps:

- **STEP 1 “Generating experiment template”** enables experimenters to download an experiment template designed according to the experiment ontology explained hereafter. The experimenters indicate the sensors that they are interested in to retrieve data available within the FIESTA-IoT platform. In

order to avoid any ambiguity, the applicative domain is also requested in order to define an explicit context. Both sensors and applicative domains follow the M3-lite taxonomy as explained in the D3.1.1 [Aggarwal et al. FIESTA-IoT D3.1.1 2016]. This step provides an experiment template with a set of files required in the following steps.

- **STEP 2 "Semantically annotating data with RDF"** enables data coming from testbeds to be semantically annotated following the FIESTA-IoT ontology [Aggarwal et al. FIESTA-IoT D3.1.1 2016]. This semantic annotation is a cornerstone component to unify data coming from different testbeds. This step is done by testbeds providers.
- **STEP 3 "Executing the reasoning engine"** enables to deduce meaningful information from data available within FIESTA-IoT. The semantic annotation previously done is also a required step to load and execute logical rules compliant with the FIESTA-IoT ontology. In FIESTA-IoT, the Jena inference engine [24] is employed and updates the Semantic Data repository with additional triples (e.g., the "Warm" concept can be deduced from a temperature). The execution of the reasoning engine can be either hidden within the FIESTA-IoT platform or employed by experimenters if they wish to.
- **STEP 4 "Executing the query engine".** In FIESTA-IoT, the Jena ARQ query engine is employed to execute SPARQL queries on semantic datasets. The execution of the query engine can be either hidden within the FIESTA-IoT platform or employed by experimenters if they wish to. In the later case, the experimenters can write their own SPARQL queries to get a subset of data that they are interested in.
- **STEP 5 "Display result in a user interface"** enables experimenters to get smart data returned by the Jena ARQ query engine. Experimenters can display in a user friendly interface data coming from the FIESTA-IoT platform and heterogeneous testbeds.

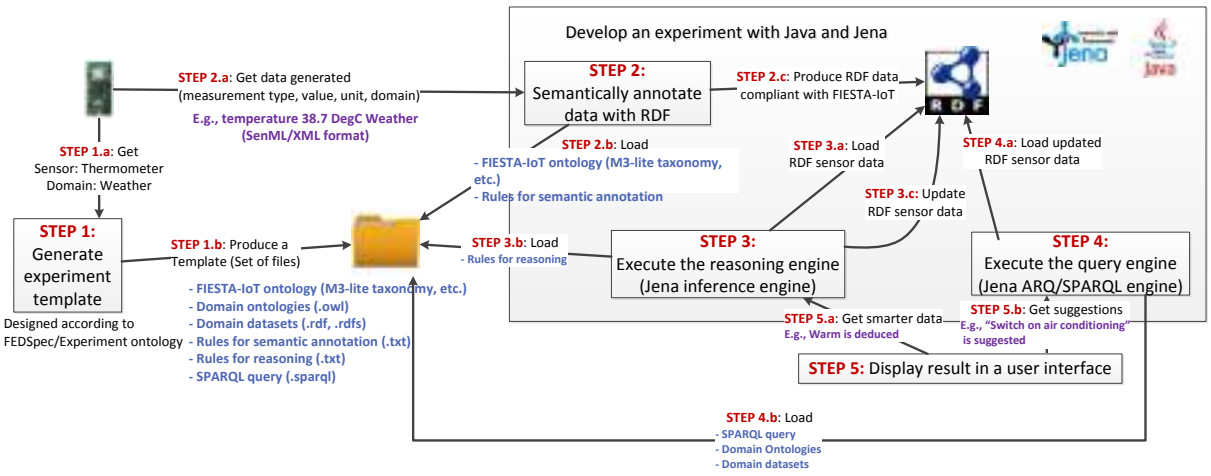

Figure 27. Example of data workflow to build an experiment

[24] https://jena.apache.org/documentation/inference/

4 EXPERIMENT-AS-A SERVICE (EAAS) MODEL

We assume two kinds of experimenters

- Expert with DSL
- Non-expert/beginners, they are not familiar with DSL technologies. Such experimenters could reuse pre-defined experiments already designed by the previous experimenters. Two main questions arise regarding the trust and quality of experiments.

4.1 Registering an experiment

The experimenter registers a new experiment through a user interface or directly through web services or APIs. He fills the parameters requested such as the name, the description, the list of devices and applicative domains necessary, and the endpoint URL to get access to data to build the experiments. Then, an experiment instance is automatically generated and designed according to the experiment DSL. The experiment instance is added in the experiment repository. If no error occurred, the experimenter is informed that the new experiment has been correctly added to the FIESTA-IoT platform.

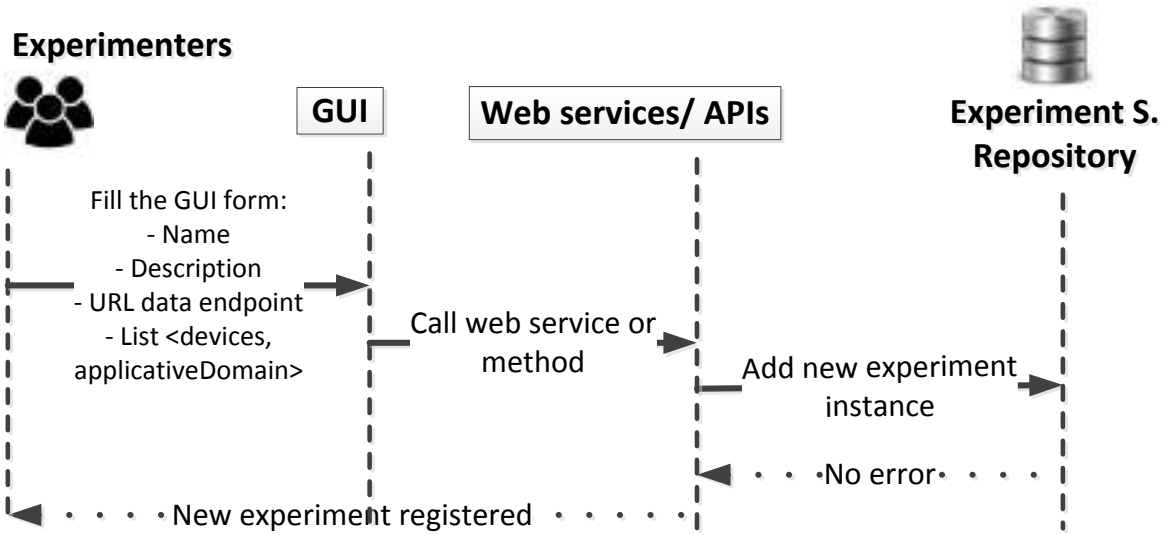

Figure 28. Registering an new experiment instance sequence diagram

4.2 Executing reasoning and query engines

Experiments would enable functionalities such as analytics on data and query data. We briefly introduce two experiment examples one for executing a reasoning engine on data and the second on for executing a query engine on data. Such functionalities are provided by the Experiment Execution Engine (EEE), more explanations can be found in Deliverable 4.4.1 [FIESTA-IoT D4.4.1, 2016].

The experimenters interact with the Experiment Execution Engine (EEE) through two actions:

- Executing the “Built-in reasoner” to deduce meaningful information from the semantic data repository. It is a logic-based rule engine; its implementation is the Jena inference engine[25]. The Built-in reasoner loads a subset of data from the semantic data repository and a subset of rules according to specific quantity Kind following the M3-lite taxonomy. Then it updates the semantic data repository with inferred data.

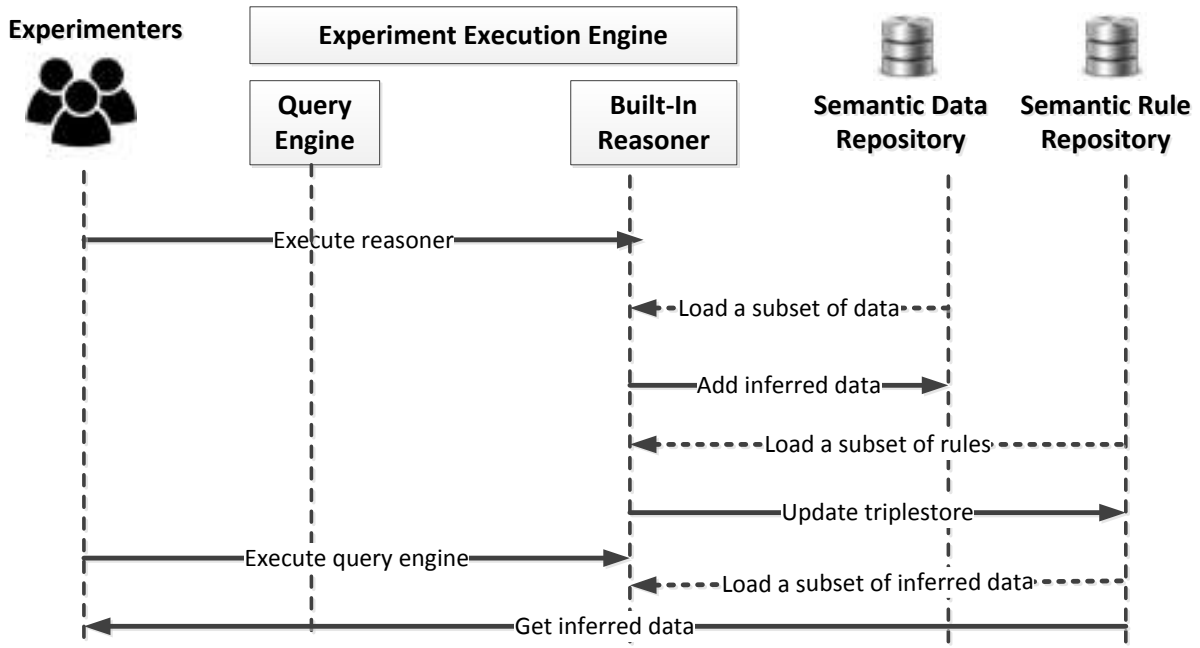

Figure 29. Interactions between the experiments and the experiment execution engine sequence diagram

4.3 FIESTA-IoT Experiment Model Object (FEDSpec)

FIESTA-IoT experiment management is facilitated by the usage of a descriptive language (DSL) which is capable of hosting all the experiments of a specific User. This language is called FIESTA-IoT Experiment Model Object (FEDSpec) and is depicted in Figure 30.

[25] https://jena.apache.org/documentation/inference/

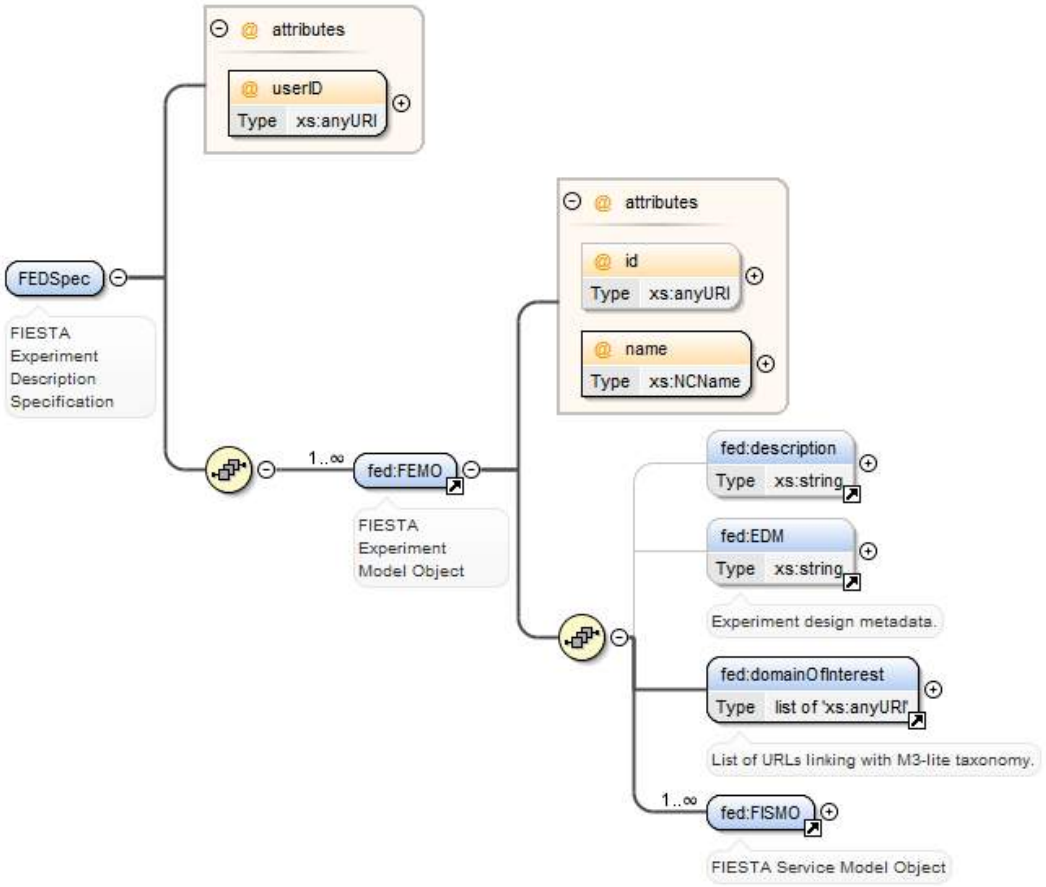

Figure 30. FEDSpec Schema graph

FEDSpec, as shown in Figure 30 above, can host all the defined experiments of an Experimenter by including multiple FIESTA-IoT Experiment Model Objects (FEMO). FEMO is responsible of holding the description of a single experiment and iconsists 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 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.

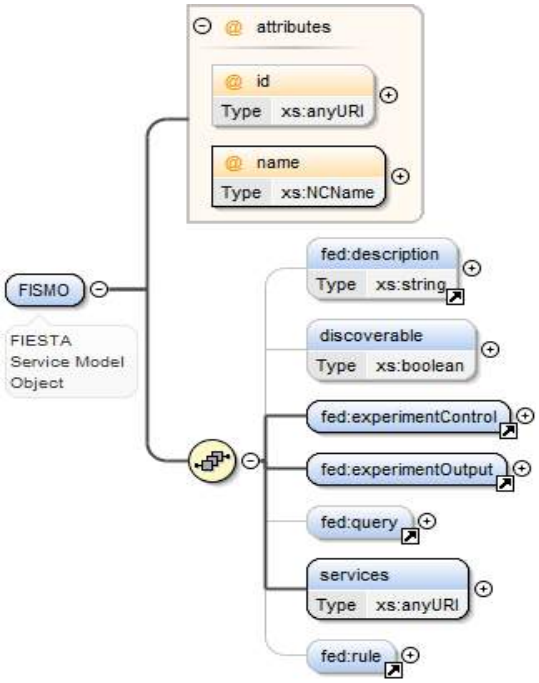

Figure 31 FISMO Schema graph

FISMO, as shown in Figure 31 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 32 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.

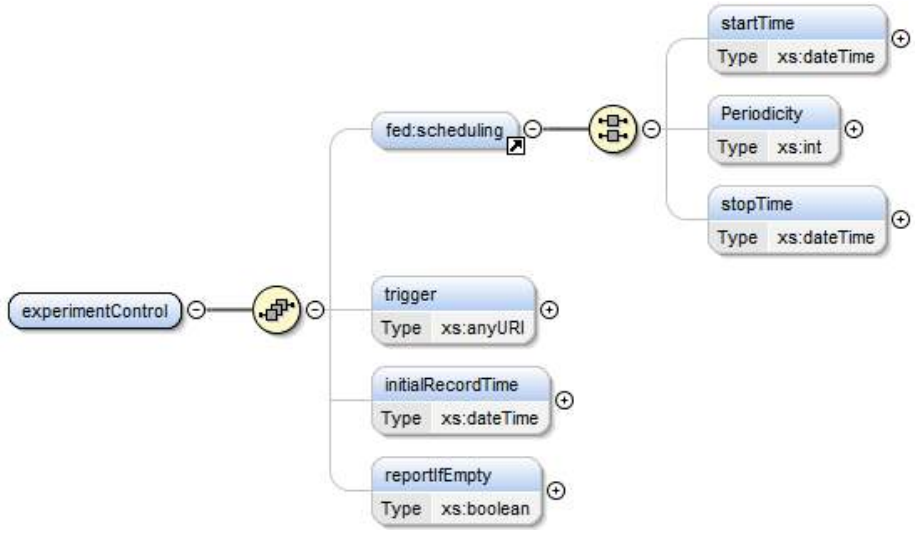

Figure 32 Experiment Control Schema graph

- The **experiment output**: (shown in Figure 33 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 (i.e. CSV, XLS, XML, etc.)
    - 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.

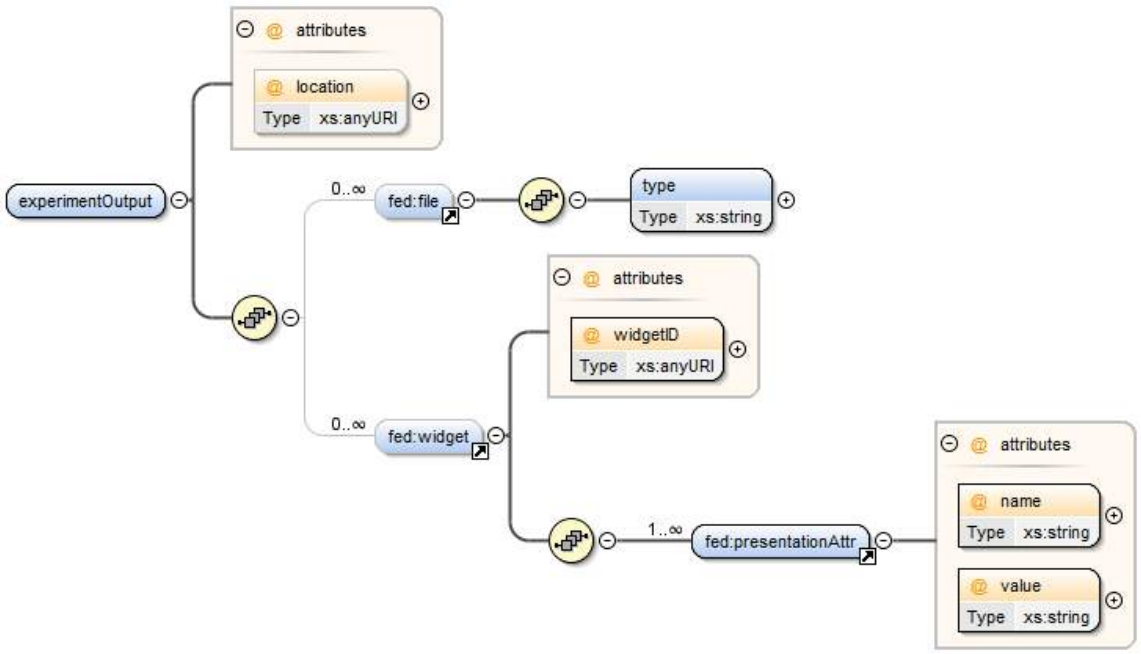

Figure 33 Experiment Output Schema graph

- 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 34 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[26]) that should be executed in order to retrieve the results of the experiment Service.
    - **Dynamic attributes**: (optional) this entity provides the ability to dynamically update attribues 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

[26] 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 and
  - dynamic geo-location
- **Dynamic attribute**: (optional) it provides a list of dynamic attributes with a user defined name and a user defined initial value.

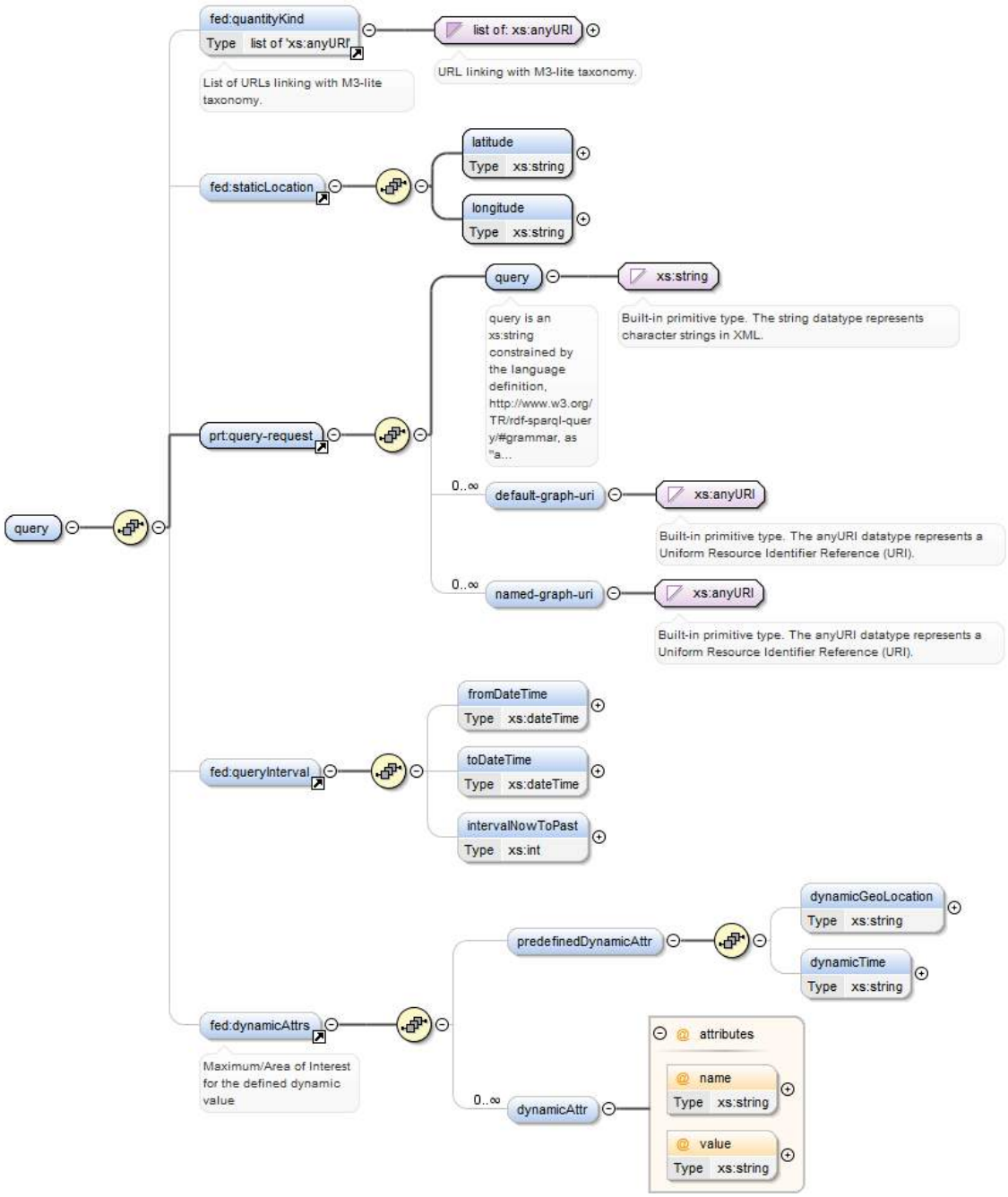

Figure 34 Query Control Schema graph

- 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 35 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.

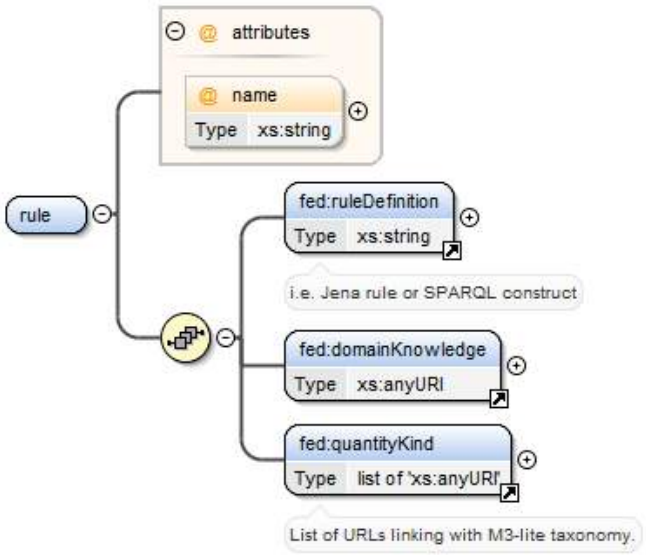

Figure 35 Rule Schema graph

4.4 Experiment Ontology

A complementary work has been done when designing the FEMO DSL, the experiment ontology has been designed to describe the notion of experiments, with a focus on the exploitation of data, by executing the reasoning engine and the query engine. The idea of the experiment ontology was also to define the idea of reusing experiments to build the concept of “Experiment-as-a-Service”.

The long-term vision within FIESTA-IoT is to align both FEMO and experiment ontologies to handle complementary functionalities. For instance, the experiment ontology provides the functionality of executing reasoning on data, and the FEMO provides the functionality to associate specific visualizations to data.

4.4.1 Overview

Software and Ontology engineering patterns are frequently used to design reusable and interoperable applications and services, called “***Experiment***". For this reason, an experiment ontology[27] has been designed to ensure interoperability among services and applications. The experiment ontology is inspired from the M3 ontology which

[27] http://fiesta-iot-tools.appspot.com/ONTOLOGIES/experiment.owl#

defined the idea of pre-defined IoT application templates [Gyrard et al. Generator 2015].

An experiment can be a simple service (e.g, get average temperature) or a composition of services (e.g., suggest home remedies when a fever is detected).

Our experiment ontology is using IoT-lite[28]and W3C SSN ontology[29] ontologies to explain that experiments get access to sensor data produced by devices, and M3-lite taxonomy[30] to classify device type, unit type, quantity kind type, etc.

An extension will be provided to be compliant with OWL-S ontology[31] since the ontology describes services as well and the Stream Annotation Ontology (SAO)[32] ontology to support real-time aspect.

Figure 36 introduces the description of an experiment which comprises of:

- The name described with *rdfs:label* (e.g., GetAverageTemperature).
- The description described with *rdfs:comment* (e.g., compute the average temperature in a city).
- An experiment has a type *iot-lite:Service*.
- The *m3-lite:domainOfInterest* property enables to classify experiments by IoT applicative domain (e.g., smart home, healthcare, etc.). An URI from the classification of IoT applicative domains from the M3-taxonomy is expected.
- *iot-lite:endpoint* to get access to data produced by devices.
- The *query* property has been designed to reuse existing queries (e.g., SPARQL queries) to easily retrieve data necessary to build the experiment. It is based on the idea of reusing SPARQL queries [Daga et al. 2015].
- *domainOntology* property has been designed to reuse domain ontologies within the experiment (e.g., healthcare ontology). Such ontologies are referenced within LOV and LOV4IoT.
- The *rule* property has been designed to reuse a subset of interoperable logical rules already implemented [Gyrard et al. S-LOR 2014].
- The *domainDataset* property has been designed to link both semantic sensor datasets and domain-specific datasets necessary to interconnect knowledge and build the experiment.
- The *widget* property enables providing the appropriate user interface according to the data type (e.g., a line chart for temperature data and a map for location data).

[28] http://purl.oclc.org/NET/UNIS/fiware/iot-lite#

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

[30] https://mimove-apps.paris.inria.fr/ontology/m3lite.html

[31] https://www.w3.org/Submission/OWL-S/

[32] http://purl.oclc.org/NET/UNIS/sao/sao#

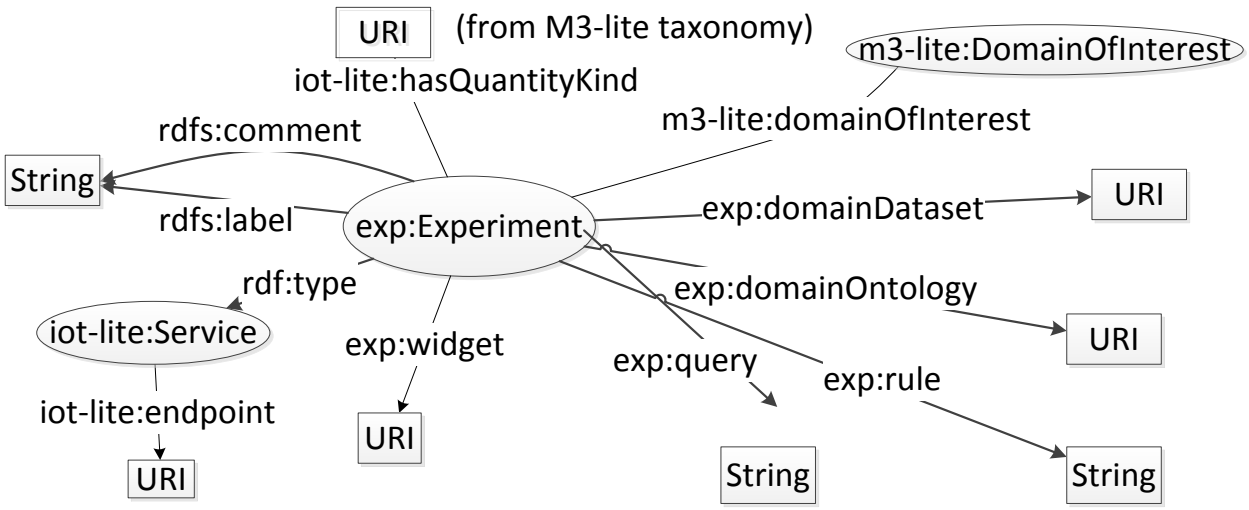

Figure 36. Experiment ontology

As a potential future work, we could add additional properties:

- *reasoningMechanism*, to clearly explain which approach is used in this application (e.g., machine learning, logic-based) to deduce meaningful information from IoT data.
- *communicationProtocol* (e.g., COAP, HTTP) to explain which protocols have been used to retrieve data.
- *dataFormat* (e.g., CSV, XML, RDF) to clearly explain which data format was used to represent sensor data. It can be used for statistics to detect which format is the most popular and used and then provide universal wrappers to ease the development of experiments.
- *accessTechnology* to describe which technologies have been used to send data through networks (e.g., Bluetooth, NFC, cellular technologies, etc.)
- serviceTechnology to describe which technologies have been used to implement the experiment (e.g, Web service: REST or SOAP, generic application).

4.4.2 Experiment ontology: a hub to align IoT ontologies

The experiment ontology has been an initial idea, also reused in WP3 to design the FIESTA-IoT ontology in order to reuse and align existing IoT ontologies relevant for this project (see deliverable D 3.1.1 for related information regarding ontologies).

Figure 37 explains one view of the FIESTA-IoT project which is unifying services, devices, systems, architectures and data through the use of ontologies:

- The **services/experiments** exploit data produced by devices/resources (e.g., sensors).The experimenters can use/access to services. The federation is required to unify existing services provided by different systems. Ontologies

such as OpenIoT and SAREF can be exploited to describe those concepts (see deliverable D 3.1.1 for more information).

- The **systems/testbeds** are connected to devices. The **testbeds** connect devices to systems. VITAL and W3C SSN can be exploited to describe those concepts.
- The **devices/resources** produce data. Ontologies such as OpenIoT, M3-lite, W3C SSN, IoT-lite can be exploited to describe those concepts.
- Unifying **data** is essential since it comes from heterogeneous testbeds. Ontologies such as SAO and M3-lite can be exploited to describe those concepts (see deliverable D 3.1.1 for more information).

Figure 37 shows the existing ontologies relevant for each bubble. When ontologies are overlapping, there is a need to unify them. For instance, the iot-taxonomy cloud be merged with the m3-lite to unify and enrich the taxonomy describing observation values/quantity kind, unit, feature of interest and sensor.

For instance, to model data, we analysed two ontologies/taxonomies: SAO ontology and the IoT taxonomy that we should unify to m3-lite.

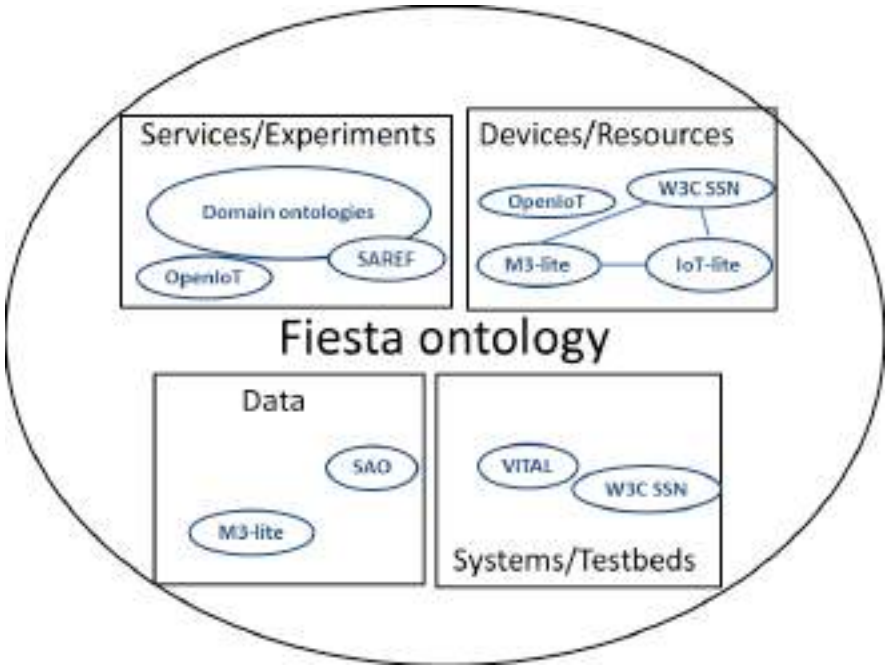

Figure 37. The FIESTA-IoT ontology overview for unifying IoT ontologies

4.5 Service Composition

Service composition would enable combining different services to obtain a more sophisticated service. We call a service an “experiment” and the composition of services can be provided through the concept of “Experiment-as-a-Service (EaaS)” Since semantic web technologies are employed within this project, an intuitive way is employing Semantic Web Services [McIlraith et al. 2001]. For instance, having two simple services such “Service 1: Executing a reasoning engine” and “Service 2: Executing a query engine” would enable a more sophisticated service “Service 3” by

combining previous simple services. The example depicted in Figure 38 shows that the service 1 would update the semantic data repository or triplestore with additional triples, from a body temperature, the fever concept could be deduced. The service 2 would enable to execute a SPARQL engine to select a sub set of RDF sensor data that we are interested in, for instance the home remedy lemon is associated to the concept fever in the triplestore. Finally, by executing service 1 and then service 2 would enable the composition of service which is service 3 and enables to build smarter IoT application. For instance, from a body temperature, service 3 enables to suggest home remedies when a fever is detected.

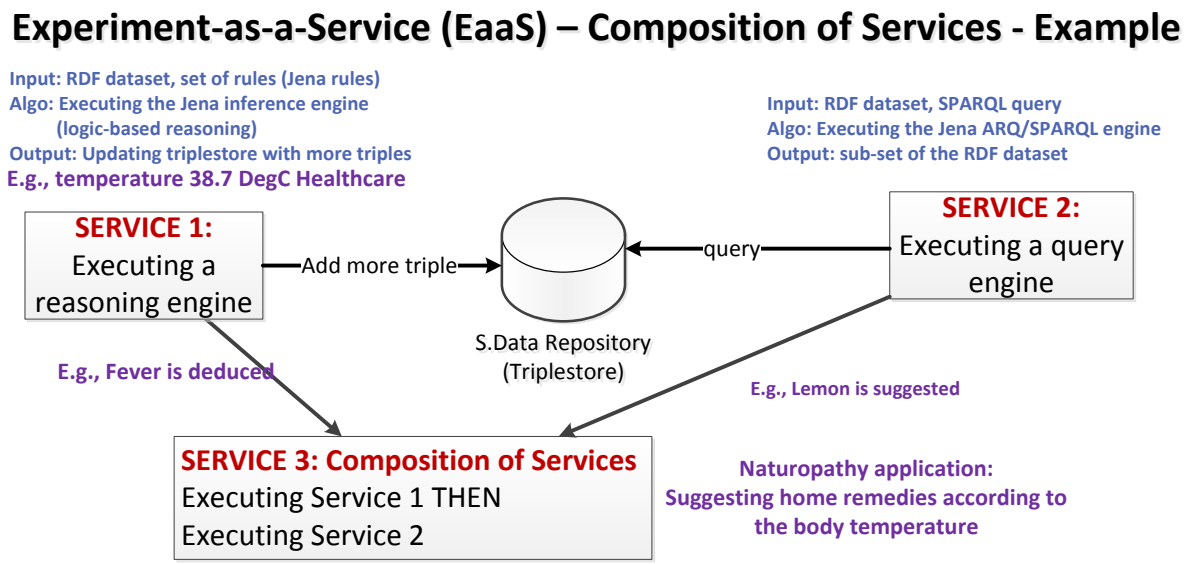

Figure 38. Composition of services example

To interact with the EaaS model, we designed the Experiment Registry Management (ERM) API that we explain in the next section.

5 EXPERIMENT REGISTRY MANAGEMENT (ERM) API SPECIFICATION

Experiment Registry Management (ERM) component exposes an API which 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

Error! Reference source not found. Error! Reference source not found. illustrates the main API primitives that support the Experiment Registry Management functionalities, while Table 6 **Error! Reference source not found.** provides more details about each one of the functions that comprise the API.

Table 6. 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:**saveExperimentServiceModelObject** (fismo:FISMO, femoID:String):String
POST:**deleteExperimentServiceModelObject** (fismoID:String):String
GET: **getALLUserExperiments** (userID:String):FEDSpec
GET:**getAllUserExperimentsDescreptions** (userID:String):ExpDescriptiveIDs
GET: **getExperimentDescreption** (femoID:String):FemoDescriptiveID
GET:**getExperimentModelObject** (femoID:String):FEMO
GET:**getExperimentServiceModelObject** (fismoID:String):FISMO

A FIESTA-IoT implementation SHALL implement methods of the Experiment Registry Management API as specified in Table 7 below:

Table 7. 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 Experiment Description Specification (FEDSpec) which includes all the User's preferences regarding the Experiment, request lifecycle and visualization preferences. 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 Experiment Description Specification. Requires as input a User ID.
getAllUserExperiments Descreptions	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 Descreption	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.

5.2 Object definition

The FEDSpec was described in detail in section 4.3 above and the XSD is provided in Appendix I below we can find the ExpDescriptiveIDs definition. In the figure below we can see the ExpDescriptiveIDs schema graph (the XSD is provided in Appendix I 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

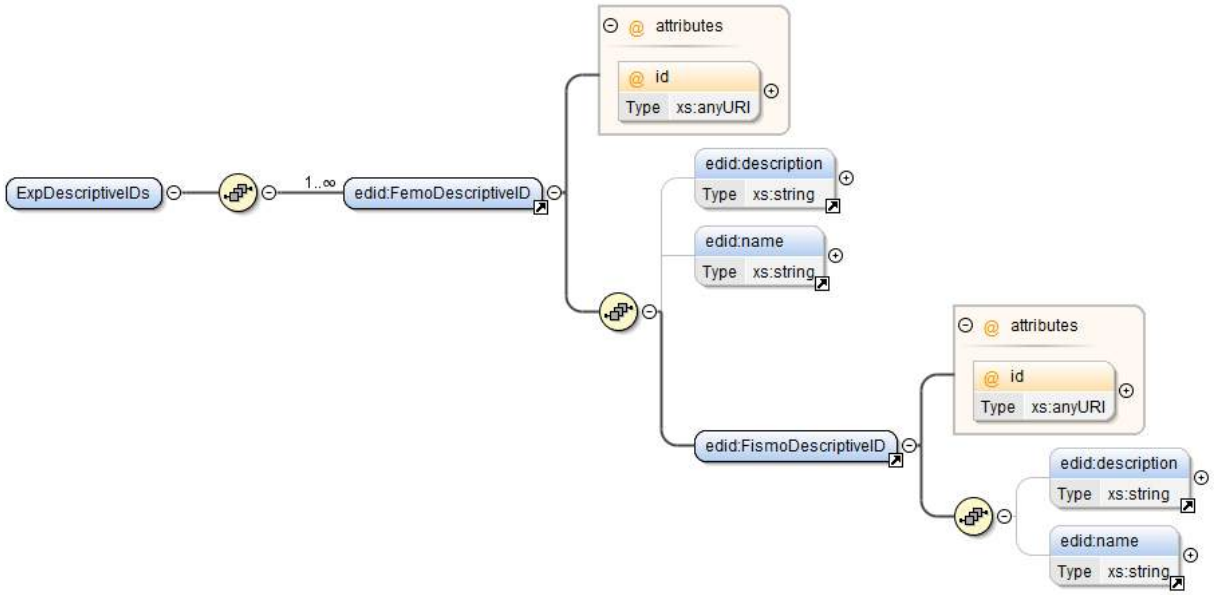

Figure 39. Experiment Descriptive IDs schema graph.

5.3 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 8. 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 8 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 9 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
saveExperimentServiceModelObject	SecurityException FismoValidationException NoSuchExperimentID ImplementationException
deleteExperimentServiceModelObject	SecurityException NoSuchServiceModelObjectID ImplementationException
getALLUserExperiments	SecurityException NoSuchUserID ImplementationException ResultTooLargeException
getAllUserExperimentsDescreptions	SecurityException NoSuchUserID ImplementationException ResultTooLargeException
getExperimentDescreption	SecurityException NoSuchExperimentID ImplementationException ResultTooLargeException
getExperimentModelObject	SecurityException NoSuchExperimentID ImplementationException ResultTooLargeException
getExperimentServiceModelObject	SecurityException NoSuchServiceModelObjectID ImplementationException ResultTooLargeException

6 EXPERIMENT REGISTRY MANAGEMENT (ERM) PROTOTYPE IMPLEMENTATION

6.1 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-iot.eu/platform/core/. The latest version of the components are under the “develop branch”[33] .

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

6.1.1 System Requirements

In order to run the prototype, you need to ensure that Java 8, WildFly container and the project management Maven are installed and/or available on your system. The prototype runs on Windows, Linux, Mac OS X, and Solaris. Before attempting to deploy and run the prototype applications, make sure that you have started WildFly.

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

6.1.2 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**:

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

mvn wildfly:deploy

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

Alternatively copy the produced (from the build process above) **experiment.erm-X.war** file from the **target** directory (**$PROJECT_HOME/target/**), into the **standalone/deployments** directory of the WildFly[34] 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/rest/experimentservices

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

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/getAllUserExperimentsDescreptions
http://[HOST]:[PORT]/experiment.erm/rest/experimentservices/getExperimentDescreption
http://[HOST]:[PORT]/experiment.erm/rest/experimentservices/getExperimentModelObject
http://[HOST]:[PORT]/experiment.erm/rest/experimentservices/getExperimentServiceModelObject

6.1.3 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 10 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

[34] http://wildfly.org/

7 AUTOMATICALLY GENERATING THE DSL

7.1 Node Red

The initial Node Red in service orientation as shown in the Figure 40 has a list of preinstalled nodes divided into 3 categories: input, output, function. The ‘input’ category has [Start, Stop, Timer, Testbeds and Semantic Discovery] nodes, the ‘output’ category has [debug, Chart, Popup, Resource Panel] nodes and the ‘function’ category has [function, Get Data, Set Data] nodes. There will be one more category which will be added dynamically once the user requested to add resource nodes this category is named as ‘Resource ‘. In Node Red every node or topology of nodes can act as a service.

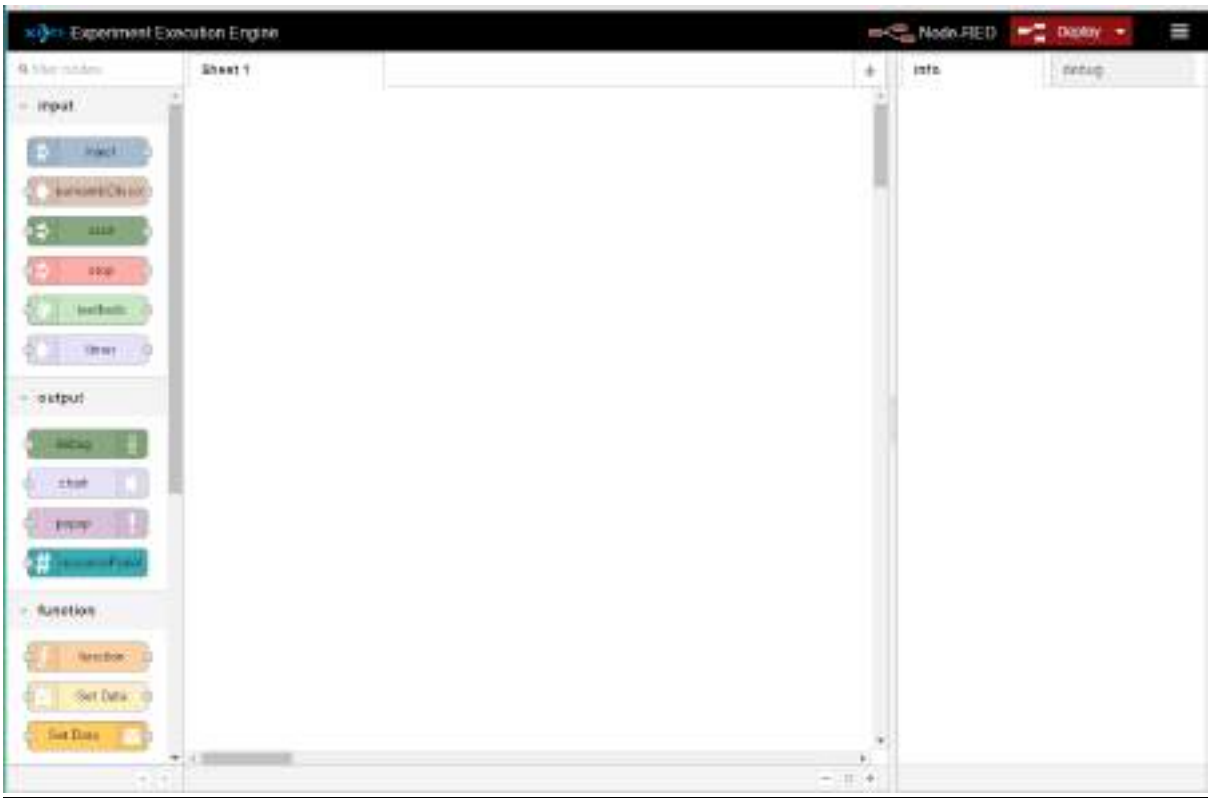

Figure 40. Node Red Overview

There are 12 initial nodes excluding ‘inject’ nodes. In the ‘input’ category the ‘Start’ and ‘Stop’ nodes are the nodes which send a Boolean msg.payload of true and false respectively. These nodes are used for the purpose of starting and stopping the flow process. The ‘timer’ node is used to relay msg.payload value that it initially received for a number of times in particular intervals which is defined before the deployment of the flow. The ‘testbed’ node at editorial phase allows the user to select a testbed provider out of the list of testbed provider and the list of location of each testbed provider’s which are predefined. This node will send an Uri of the selected testbed as msg.payload once it received any msg.payload. The ‘Semantic Discovery’ node (for now) is only a test node to send a SPARQL query and get response.

In the ‘output’ category the ‘debug’ node is one of the original nodes that comes with node red which is used for debugging propose and this node display whatever topic and msg.payload it receives to the debug tab. The ‘chart’ node using google charts

API will chart a graph out of the data it received. It has different chart types provided by the google charts. The ‘popup’ node acts just like the debug node but with a difference of popping up a new tab to display the msg.payload and topic. The ‘resourcePanel’ node is used to add dynamic nodes to palette under ‘resource’ category. This node can only be used together with the ‘testbeds’ node since this node needs a testbed URI input (valid).

In ‘function’ category the ‘function’ node is for the advanced users to write down their own code into the flow. The ‘Get Data’ node is used to get the response out of the resource nodes provided container API. This node is predefined only to parse the response of the resource nodes using it with any other node would generate a warning on deploy. The ‘Set Data’ node same as ‘Get Data’ expect it is used to set the container data rather than getting the data.

In case of dynamic palette addition there will be one more category called ‘resource’ which will have resource nodes these nodes will contain API for the resource container.

7.2 Generating the DSL

To generate DSL from the Node Red the user have to deploy the experiment build on the Node Red first which would result in the creation on flow file of json type as illustrated in the Figure 41.

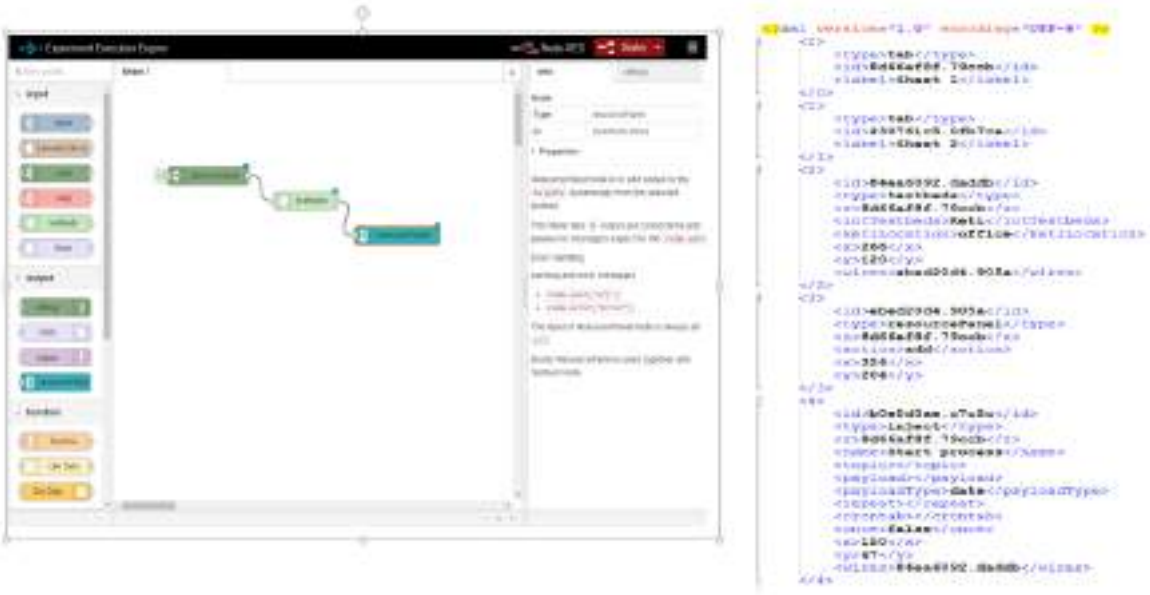

Figure 41. Experiment and flow.json in Node Red

This output is then mapped to the FIESTA-IoT DSL and will be uploaded to the meta-cloud which could them be used to recreate, modify or distribute as minute experiment.

8 CONCLUSIONS

In this deliverable the innovative concept **Experiment-as-a-Service (EaaS)** has been designed to aggregate and ensure the interoperability of data streams stemming from

different platforms or testbeds, as well as the need to provide tools and techniques or building applications that horizontally integrate diverse IoT solutions

We took inspiration from existing work having a complementary objective in order to design the EaaS concept. A FIESTA-IoT Meta-Cloud methodology and architecture have been designed. 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). The Experiment Registry Management (ERM) API has been designed to deal and easily interact with the EaaS Model. A concrete use case has been implemented through the NodeRed workflow editor to design experiments and generating the experiment workflow compliant with the EaaS Model/DSL.

We provided two solutions to implement the DSL: FEDSpec and the experiment ontology. For a fast implementation, we have chosen FEDSpec since the integration with NodeRed and other components provided by FIESTA-IoT are simple. The integration with the experiment ontology or a hybrid solution can be explained in the second version of this deliverable.

8.1 Future steps

As a future work, we plan to integrate the experiment ontology which would enable to provide more features such as reasoning and composition of services (see Deliverable [FIESTA-IoT D3.3, 2016]) and visualization of data (see Deliverable [FIESTA-IoT D4.3, 2016]. The experiment ontology can be itself explicitly integrated within the FEDSpec.

9 REFERENCES

[Aberer et al. GSN 2007] Karl Aberer, Manfred Hauswirth, and Ali Salehi. Infrastructure for data processing in large-scale interconnected sensor networks. In *Mobile Data Management, 2007 International Conference on*, pages 198–205. IEEE, 2007.

[Booth et al. WSDL 2007] David Booth and Canyang Kevin Liu. Web services description language (wsdl) version 2.0 part 0: Primer. *W3C Recommendation*, 26, 2007.

[Chen et al. 2005] Harry Chen, Tim Finin, and Anupam Joshi. The soupa ontology for pervasive computing. In *Ontologies for agents: Theory and experiences*, pages 233–258. Springer, 2005.

[Christopoulou et al. 2005] Eleni Christopoulou and Achilles Kameas. Gas ontology: an ontology for collaboration among ubiquitous computing devices. *International Journal of Human-Computer Studies*, 62(5):664–685, 2005.

[Compton et al. SSN 2012] M. Compton, P. Barnaghi, L. Bermudez, R. Garcia-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, et al. The ssn ontology of the w3c semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2012. http://www.w3.org/2005/Incubator/ssn/ssnx/ssn.

[Daga et al. 2015] Enrico Daga, Luca Panziera, and Carlos Pedrinaci. A basilar approach for building web apis on top of sparql endpoints. 2015.

[De et al. 2011] Suparna De, Payam Barnaghi, Martin Bauer, and Stefan Meissner. Service modelling for the internet of things. In *Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on*, pages 949–955. IEEE, 2011.

[FIESTA-IoT D2.4, 2015] D2.4 FIESTA Meta-Cloud Architecture and Technical Specifications. FIESTA-IoT Consortium, September 2016

[FIESTA-IoT D3.1.1 2016] D3.1.1 Semantic models for testbeds, interoperability and mobility support and best practices. FIESTA-IoT consortium. March 2016.

[FIESTA-IoT D3.3, 2016] D3.3 Concept and Development for IoT data analytics and IoT stream and service management. FIESTA-IoT Consortium, October 2016

[FIESTA-IoT D4.2, 2016] D4.2 Authentication, Authorization, Data Protection and Reservation of Resources. FIESTA-IoT Consortium, July 2016

[FIESTA-IoT D4.3, 2016] D4.3 Tools and Techniques for Managing Interoperable Data sets. FIESTA-IoT Consortium, October 2016

[FIESTA-IoT D4.4.1, 2016] D4.4.1 Infrastructure for Submitting and Managing IoT Experiments. FIESTA-IoT Consortium, October 2016

[Fujii et al. 2009] Keita Fujii and Tatsuya Suda. Semantics-based context-aware dynamic service composition. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2):12, 2009.

[Furno et al. 2014] Angelo Furno and Eugenio Zimeo. Context-aware composition of semantic web services. *Mobile Networks and Applications*, 19(2):235–248, 2014.

[Groth et al. 2014] Paul Groth, Antonis Loizou, Alasdair JG Gray, Carole Goble, Lee Harland, and Steve Pettifer. Api-centric linked data integration: The open phacts discovery platform case study. *Web Semantics: Science, Services and Agents on the World Wide Web*, 29:12–18, 2014.

[Guinard et al. WoT 2010] Dominique Guinard, Vlad Trifa, and Erik Wilde. A resource oriented architecture for the web of things. In *Internet of Things (IOT), 2010*, pages 1–8. IEEE, 2010.

[Gyrard et al. WebKnowledge 2016] Amelie Gyrard, Pankesh Patel, Amit Sheth, and Martin Serrano. Building the web of knowledge with smart iot applications. 04/2016 2016.

[Gyrard et al. S-LOR 2014] Amelie Gyrard, Christian Bonnet, and Karima Boudaoud. Helping IoT application developers with sensor-based linked open rules. In *SSN 2014, 7th International Workshop on Semantic Sensor Networks in conjunction with the 13th International Semantic Web Conference (ISWC 2014), 19-23 October 2014, Riva Del Garda, Italy*, 10 2014.

[Gyrard et al. Generator 2015] Amelie Gyrard, Christian Bonnet, Karima Boudaoud, and Martin Serrano. Assisting iot projects and developers in designing interoperable semantic web of things applications. In *IEEE International Conference on Internet of Things 2015 (iThings)*, 2015.

[Gyrard et al. DataWorkflow 2014] Amélie Gyrard, Christian Bonnet, and Karima Boudaoud. Enrich machine-to-machine data with semantic web technologies for cross-domain applications. In WF-IOT 2014, World Forum on Internet of Things, 6-8 March 2014, Seoul, Korea, Seoul, KOREA, REPUBLIC OF, 03 2014.

[Gyrard et al. oneM2MLanguage 2015] Amélie Gyrard and Christian Bonnet. A unified language to describe M2M/IoT data, 03 2015.

[Gyrard et al. SEG 3.0 Methodology 2016] Amelie Gyrard and Martin Serrano. Connected smart cities: Interoperability with seg 3.0 for the internet of things. In 30th IEEE International Conference on Advanced Information Networking and Applications Workshops, 2016, Crans-Montana, Switzerland., 2016.

[Gyrard et al. LOV4IoT V1 2016] Amelie Gyrard, Christian Bonnet, Karima Boudaoud and Martin Serrano. LOV4IoT: A second life for ontology-based domain knowledge to build Semantic Web of Things applications
4rd International Conference on Future Internet of Things and Cloud (FiCloud 2016), 22-24 August 2016, Vienna, Austria.

[Gyrard et al. LOV4IoT V2 2016] Amelie Gyrard, Ghislain Atemezing, Christian Bonnet, Karima Boudaoud and Martin Serrano. Reusing and Unifying Background Knowledge for Internet of Things with LOV4IoT. 4rd International Conference on Future Internet of Things and Cloud (FiCloud 2016), 22-24 August 2016, Vienna, Austria

[Gyrard et al. SemanticEngine 2015] Amelie Gyrard and Martin Serrano. A unified semantic engine for internet of things and smart cities: From sensor data to end-users applications. In IEEE International Conference on Internet of Things 2015 (iThings), 2015.

[Kefalakis et al. 2012] Nikos Kefalakis, John Soldatos, Stavros Petris, Sofoklis Efremidis. OpenIoT D4.1: D4.1 Service Delivery Environment Formulation Strategies, 2012

[Le-Phuoc et al. 2014] Danh Le-Phuoc, Hoan Nguyen Mau Quoc, Quoc Hung Ngo, Tuan Tran Nhat, and Manfred Hauswirth. Enabling live exploration on the graph of things? *Proceedings of the Semantic Web Challenge*, 2014.

[Maleshkova et al. 2009] Maria Maleshkova, Carlos Pedrinaci, and John Domingue. Supporting the creation of semantic restful service descriptions. 2009.

[Martin et al. OWL-S 2004] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, et al. Owl-s: Semantic markup for web services. *W3C member submission*, 22:2007–04, 2004.

[McIlraith et al. 2001] Sheila A McIlraith, Tran Cao Son, and Honglei Zeng. Semantic web services. *IEEE intelligent systems*, (2):46–53, 2001.

[Mokhtar et al. 2006] Sonia Ben Mokhtar, Damien Fournier, Nikolaos Georgantas, and Valérie Issarny. Context-aware service composition in pervasive computing environments. In *Rapid Integration of Software Engineering Techniques*, pages 129–144. Springer, 2006.

[Norton et al. 2010] Barry Norton and Reto Krummenacher. Consuming dynamic linked data. In *COLD*, 2010.

[Patel et al. 2013] Pankesh Patel, Animesh Pathak, Damien Cassou, and Valerie Issarny. Enabling high-level application development in the internet of things. In Marco Zuniga and Gianluca Dini, editors, *Sensor Systems and Software*, volume 122 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 111–126. Springer International Publishing, 2013.

[Patel et al. 2011] Pankesh Patel, Animesh Pathak, Thiago Teixeira, and Valérie Issarny. Towards application development for the internet of things. In *Proceedings of the 8th Middleware Doctoral Symposium*, MDS ’11, pages 5:1–5:6, New York, NY, USA, 2011. ACM.

[Pedrinaci et al. 2010] Carlos Pedrinaci, Dong Liu, Maria Maleshkova, David Lambert, Jacek Kopecky, and John Domingue. iserve: a linked services publishing platform. In *CEUR workshop proceedings*, volume 596, 2010.

[Perera et al. SaaS 2015] Charith Perera. *Dynamic Configuration for Sensing as a Service Model in the Internet of Things Paradigm*. PhD thesis, The Australian National University, 2015.

[Perera et al. 2014] Charith Perera, Prem Prakash Jayaraman, Arkady Zaslavsky, Dimitrios Georgakopoulos, and Peter Christen. Sensor discovery

and configuration framework for the internet of things paradigm. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, pages 94–99. IEEE, 2014.

[Perera et al. Survey 2014] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Context aware computing for the internet of things: A survey. *Communications Surveys & Tutorials, IEEE*, 16(1):414–454, 2014.

[Perera et al. SaaS 2014] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Sensing as a service model for smart cities supported by internet of things. *Transactions on Emerging Telecommunications Technologies*, 25(1):81–93, 2014.

[Perera et al. 2013] Charith Perera, Arkady Zaslavsky, Michael Compton, Peter Christen, and Dimitrios Georgakopoulos. Context aware sensor configuration model for internet of things. In *Proceedings of the 12th International Semantic Web Conference (Poster & Demo) (ISWC), Sydney, Australia, October, 2013*, 2013.

[Serrano et al. EaaS 2015] Martin Serrano, John Soldatos, Philippe Cousin, and Pedro Malo. Internet of things experimentation: Linked-data, sensing-as-a-service, ecosystems and iot data stores. *Building the Hyperconnected Society: Internet of Things Research and Innovation Value Chains, Ecosytems and Markets*, 2015.

[Speiser et al. LIDS 2010] Sebastian Speiser and Andreas Harth. Taking the lids off data silos. In *Proceedings of the 6th International Conference on Semantic Systems*, page 44. ACM, 2010.

[Speiser et al. 2010] Sebastian Speiser and Andreas Harth. Towards linked data services. In *Proceedings of the 9th International Semantic Web Conference (ISWC)*. Citeseer, 2010.

[Soldatos et al. 2015] John Soldatos, Nikos Kefalakis, Manfred Hauswirth, Martin Serrano, Jean-Paul Calbimonte, Mehdi Riahi, Karl Aberer, Prem Prakash Jayaraman, Arkady Zaslavsky, Ivana Podnar Žarko, et al. Openiot: Open source internet-of-things in the cloud. In *Interoperability and Open-Source Solutions for the Internet of Things*, pages 13–25. Springer, 2015.

[Serrano et al. 2014] Martin Serrano, Payam Barnaghi, and Philippe Cousin. Semantic Interoperability: Research Challenges, Best Practices, Solutions and Next Steps, ierc ac4 manifesto, 2014.

2016 FIESTA-IOT

APPENDIX I- DEFINED SCHEMATA

Table 11. 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 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 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 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 12: 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>
```