HORIZON 2020
The EU Framework Programme for Research and Innovation

FIESTA-IoT
www.fiesta-iot.eu

HORIZONS 2020 PROGRAMME

Research and Innovation Action – FIRE Initiative

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

FIESTA-IoT Meta-Cloud Architecture

Document Id:	FIESTAIoT-WP2-D24-FIESTA-IOTMetaCloudArchitecture (v2)
File Name:	FIESTAIoT-WP2-D24-FIESTA-IOTMetaCloud Architecture V2FINAL.doc
Document reference:	Deliverable D2.4 (v2)
Version:	V2
Editor:	F. Carrez (UniS)
Authors:	F. Carrez (UniS), Alireza Ahrabian (UNIS), R. Agarwal (INRIA) + Authors of D2.4 (v1)
Organisation:	UniS
Date:	30th of June 2018
Document type:	R
Dissemination level:	PU

Sociedad para el desarrollo de Cantabria – SODERCAN (Spain), Ayuntamiento de Santander – SDR (Spain), Korea Electronics Technology Institute KETI, (Korea).

DOCUMENT HISTORY

Rev.	Author(s)	Organisation(s)	Date (dd/mm/yyyy)	Comments
V2.0.1	François Carrez	UNIS		initial Draft
V2.0.2	François Carrez	UNIS	19/05/2017	Role definition + related System UC descriptions
V2.0.3	Francois Carrez	UNIS	12/10/2017	Introduction of Reasoner + Associated System UCs, VE section improvements
V2.0.4	Alireza Ahrabian	UNIS	18/10/2017	Introduction of Analytics FC + Associated System UCs
V2.0.5	Rachit Agarwal	Inria	06/02/2018	IoT Process Management FG + Associated System UCs.
V2.0.6	Francois Carrez	UniS	15/06/2018	Instantiation View
V2.0.7	Francois Carrez	UniS	22/06/2018	Finalization

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TERMS AND ACRONYMS

ACP	Access Control Policies
API	Application Program Interface
ARM	Architecture Reference Model
CRUD	Create/Read/Update/Delete
DAaaS	Data Analytics as a Service
DM	Domain Model
DMS	Data Management Service
DSL	Domain Specific Language
EaaS	Experiment-as-a-Service
EEE	Experiment Execution Engine
EMC	Experiment Management Console
ERM	Experiment Registery Module
ERS	Experiment Result Storage
FC	Functional Component
Fed4FIRE	Federation For Future Internet Research and Experimentation
FG	Functional Group
FIESTA	Federated Interoperable Semantic IoT/cloud Testbeds and Applications
FIRE	Future Internet Research and Experimentation
FM	Functional Model
FTP	File Transfer Protocol
FV	Functional View
GW	Gateway
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IEEE	Institute of Electrical and Electronics Engineers
IERC	IoT European Research Cluster
IM	Information Model
IoT	Internet of Things
IV	Information View
JSON	JavaScript Object Notation
KEM	Key Exchange and Management

KP	Knowledge Producers
KPI	Key Performance Indicator
OWL	Ontology Web Language
PDP	Policy Decision Point
PE	Physical Entity
PEP	Policy Enforcement Point
RA	Reference Architecture
REST	REpresentational State Transfer
RDF	Resource Description Framework
RDP	Raw Data Producer
RFC	Request For Comments
r-IoT Service	Resource-centric IoT Service
RM	Reference Model
SDR	Semantic Data Repository
SSRD	Semantic Service & Resource Descriptions
SUC	System Use-Cases
TDB	Triplestore Database
TPI	Testbed Provider Interface
TPS	Testbed Provider Service
TRR	Testbed and Resource Registration
TTP	Trusted Third Party
UC	Use-Cases
VASP	Value-Added Service Provider
VE	Virtual Entity
ve-IoT Service	Virtual Entity-centric IoT Service
WP	Work Package
XML	eXtensible Markup Language

1 POSITIONING

1.1 FIESTA-IoT

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 not less important the security aspects in IoT). IoT Advances have drawn a common big challenge that focuses on the integration of the IoT generated data. The 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 able to offer them as experimental services in order to optimise 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 Internet of Things data from heterogeneous systems and environments and their entity resources (such as smart devices, sensors, actuators, etc.), this vision of integrating IoT platforms, test-beds and their associated silo applications within cloud infrastructures is related with several scientific challenges, such as the need to aggregate and ensure the interoperability of data streams stemming from different IoT platforms or test-beds, 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 (WPs): 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 test-beds to test and feedback the platform and tools generated, thus open calls for those tenders will be issued that are also part of the WP1 activity and is called selection of third-parties.

WP2 focuses on stakeholder’s requirements and the analysis on IoT Platforms and Test-beds in order to define strategies for the definition and inclusion of Experiments, Tools and Key Performance Indicators (KPIs). The activities in this WP2 are focused on studying the IoT Platforms and Test-beds and the specification of the Experiments, the detail of the needed tools for experimentation and the KPIs for validate 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 and establishes the Certification Programme Specifications that will drive the global market confidante and certification actions around IoT experimentation model.

WP3 package focuses on providing technologies, interfaces, methods and solutions to represent the device and network nodes of the test-beds as virtualized resources. The virtualized resources will be represented as services and will be accessible via common service interfaces and Application Program Interfaces - APIs (i.e. the FIESTA-IoT Test-bed 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 test-bed devices and resources.

WP4 will implement an infrastructure for accessing data and services from multiple distributed diverse test-beds in a secure and test-bed agnostic way. To this end, it will rely on the semantic interoperability of the various test-beds (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 Experiment 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 deploy and deliver 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 would define a complete set of experiments to test the developments coming from other WPs (mainly WP3 and 4), covering all the specifications and requirements of WP2. Developments will be tested over available IoT environments and/or smart cities platforms. WP5 would also provide evaluation of the key performance indicators defined for every experiment/pilot. The final deployed experiments will include a subset of those coming from WP2, 3 and 4, 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 programme, 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, aiming 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 work package focuses on ensuring that FIESTA-IoT suite, models and tools engages 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 report on their satisfaction with the services offered in FIESTA-IoT will be put in place at the end of the project.

1.2 WP2 Overview

This Work Package covers the FIESTA-IoT requirements engineering activities and will produce the requirements associated with test-bed-agnostic experimentation, as well as with the Experiment-as-a-Service model to designing and conducting experiments. WP2 is composed of five different tasks (depicted in Figure 1), which tackle distinct aspects of the FIESTA-IoT EaaS Experimental Infrastructure:

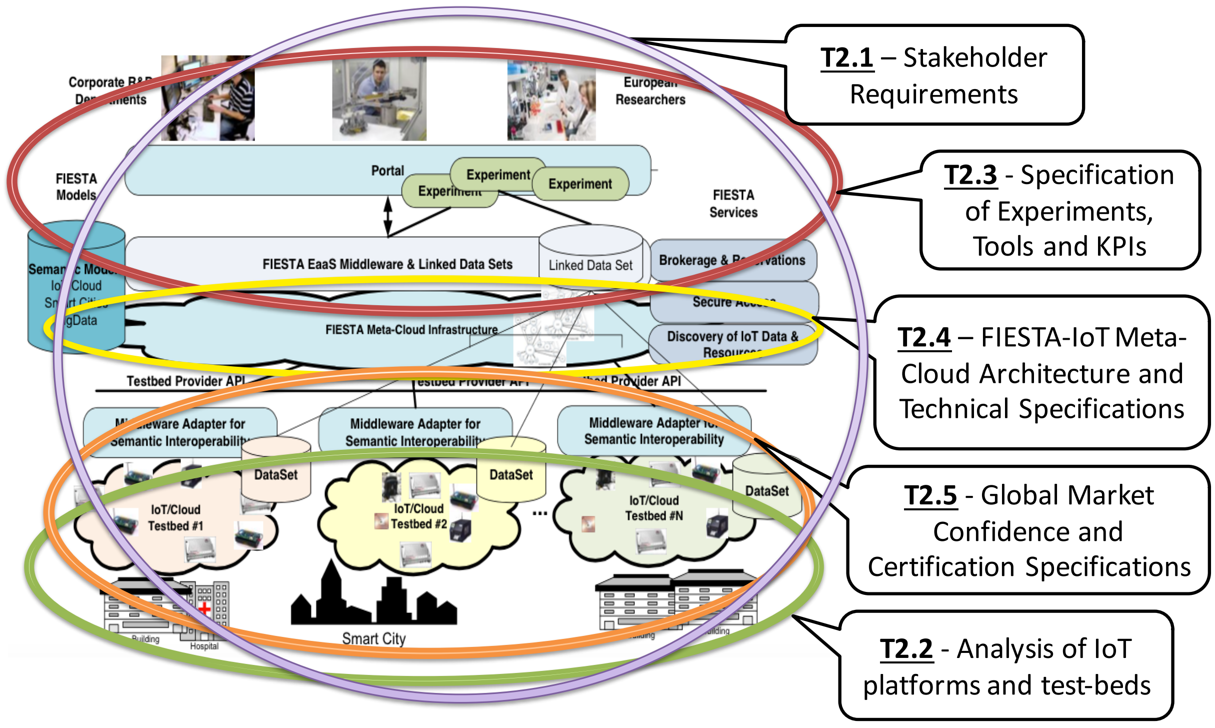

Corporate R&D Departments
European Researchers
FIESTA Models
Portal
Experiment
Experiment
Experiment
FIESTA Services
FIESTA EaaS Middleware & Linked Data Sets
Linked Data Set
Brokerage & Reservations
Secure Access
FIESTA Meta-Cloud Infrastructure
Discovery of IoT Data & Resources
Testbed Provider API
Testbed Provider API
Testbed Provider API
Middleware Adapter for Semantic Interoperability
Middleware Adapter for Semantic Interoperability
Middleware Adapter for Semantic Interoperability
DataSet
DataSet
DataSet
IoT/Cloud Testbed #1
IoT/Cloud Testbed #2
IoT/Cloud Testbed #N
Building
Hospital
Smart City
T2.1 – Stakeholder Requirements
T2.3 - Specification of Experiments, Tools and KPIs
T2.4 – FIESTA-IoT Meta-Cloud Architecture and Technical Specifications
T2.5 - Global Market Confidence and Certification Specifications
T2.2 - Analysis of IoT platforms and test-beds

Figure 1: WP2 Overview

The WP2 Tasks cross all aspects of the FIESTA-IoT Infrastructure. They are:

Task 2.1. Stakeholder Requirements: This task is responsible for gathering and processing all Stakeholder requirements (using the Volere Requirements specifications (Volere)). The involved stakeholders include: the IoT test-beds to be integrated, the experiment providers, and also researchers and experimenters. Also, external projects (such as Open-IoT and Fed4Fire) will provide requirements so, to prepare FIESTA-IoT for the Open-calls. This task will produce a set of requirements that will be used by all other WP2 tasks.

Task 2.2. Analysis of IoT platforms and Test-beds: This task is focused on the Test-beds and IoT Platforms, analysing and describing what they do and how they do it. It will also use the set of test-bed requirements produced in T2.1 to better understand if each test-bed can fulfil the stakeholders' requirements. This task will then, model the Test-beds and IoT Platforms in functional blocks using the IoT Architecture Reference Model (ARM) model from IoT-A project (IoT-A, 2013). It will gather what type of information they provide, and how they provide this information so that Task 2.4 can take this into account when developing the FIESTA-IoT Architecture. The outcome of this task will provide a basis for WP3.

Task 2.3. Specification of Experiments, Tools and KPIs: This task will specify all planned experiments and extrapolate from it the needed tools to execute those experiments. It will use the experiment related requirements produced in T2.1 and analyse them in terms of the tools that need to be provided from FIESTA-IoT to the experimenters. It will also specify the KPIs of each experiment so that later validation can occur. The result of this Task will be used as input to WP5.

Task 2.4. FIESTA-IoT Meta-Cloud Architecture and Technical Specifications: This Task will define the FIESTA-IoT Meta-Cloud Architecture, leveraging on the IoT-A ARM, and the technical specifications that will drive all the development work of the project. It will use information from previous tasks to identify the main building blocks, design & technology choices, and specify the functional blocks of the FIESTA-IoT architecture needed for achieving FIESTA-IoT's technical objectives. This architecture will serve as a base for all of the development phase of the project and more specifically for WP4.

Task 2.5. Global Market Confidence and Certification Specifications: This task is intended to study and define the global market confidence and certification specification. This means that this task is responsible to define the certification process, and the set of requirements that are required for a test-bed to comply, in order to be integrated into FIESTA-IoT. The outcome of this task will be used in WP6.

As described in the previous tasks description, the outcomes of each task will be used by other tasks of this WP2, or be used as inputs for the work in other WPs.

These relations between WP2 tasks and other WPs are depicted in Figure 2.

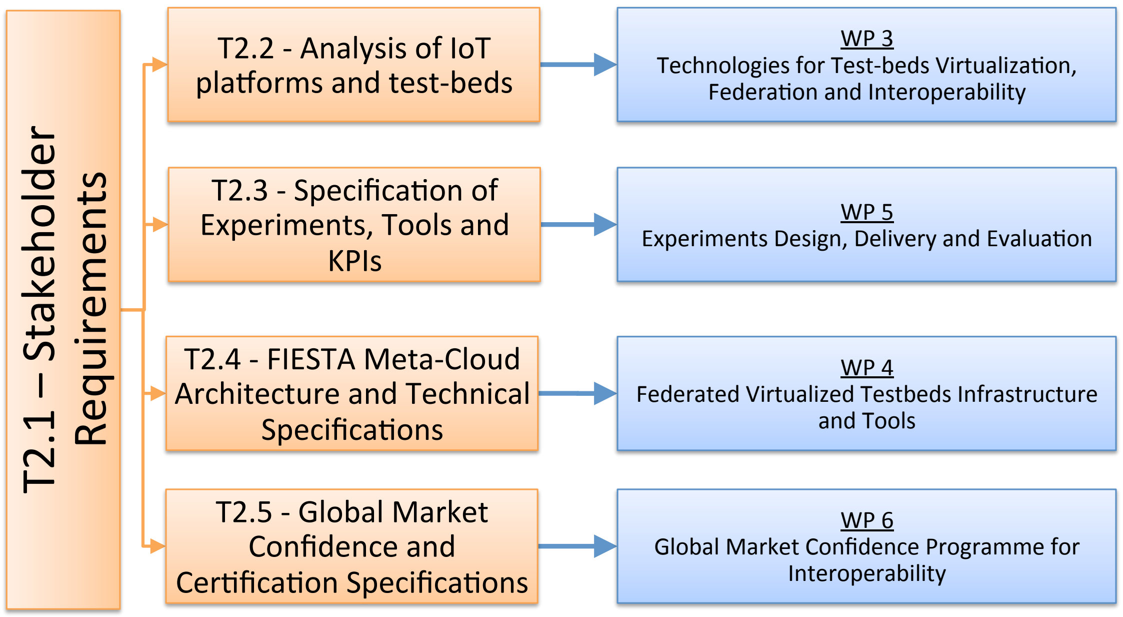

T2.1 – Stakeholder Requirements
T2.2 - Analysis of IoT platforms and test-beds
T2.3 - Specification of Experiments, Tools and KPIs
T2.4 - FIESTA Meta-Cloud Architecture and Technical Specifications
T2.5 - Global Market Confidence and Certification Specifications
WP 3 Technologies for Test-beds Virtualization, Federation and Interoperability
WP 5 Experiments Design, Delivery and Evaluation
WP 4 Federated Virtualized Testbeds Infrastructure and Tools
WP 6 Global Market Confidence Programme for Interoperability

Figure 2: Relationship between WP2 tasks and with other WPs

In reference to the FIESTA-IoT project general objective(s), WP2 has a set of sub-objectives defined activities that are described as follow:

1) Determination of Stake Holder requirements.
2) Description of IoT Platforms and test-beds in order to facilitate their integration into FIESTA-IoT infrastructure.
3) Specification of planned experimentation and its executing tools, and the KPIs that will be used for validation.
4) Definition of the FIESTA-IoT Meta-Cloud architecture and the technical specifications required for the development WPs
5) Definition of the Global market confidence and Certification specifications

The Work Package 2 will also result in five deliverables, which will be directly linked with the objectives and tasks of the WP. Each Deliverable will be an outcome of each Task, meaning that Deliverable D2.1 will be provided at the end of T2.1 with the results of that specific task. The following table details the set of deliverables to be expected from WP2, with reference to the related tasks, the responsible partner for each deliverable and all other contributors.

Table 1: WP2 Deliverables

No.	Deliverable	Responsible Partner	Contributors
D2.1	Stakeholders Requirements	UNPARALLEL	NUIG-DERI, NEC, UNICAN, SODERCAN, SDR
D2.2	IoT Platforms and Testbeds Analysis	Com4Innov	KETI, UNICAN, UNPARALLEL, AIT, NUIG-DERI, INRIA, NEC
D2.3	Experiments, Tools and KPIs Specification	UNPARALLEL	UNICAN, INRIA, NEC, NUIG-DERI, AIT, ITINNOV, SODERCAN
D2.4	FIESTA Meta-Cloud Architecture and Technical Specifications	UNIS	AIT, NUIG-DERI, UNICAN, ITINNOV, KETI
D2.5	Global Market Confidence and Certification Programme Specifications	EGM	AIT, SODERCAN

1.3 Audience

This deliverable addresses the following audiences:

- **Researchers and engineers within the FIESTA-IoT consortium**, which will take into account the various requirements in order to research, design and implement the architecture of the FIESTA-IoT Meta-Cloud Architecture.
- **Researchers on Future Internet Research and Experimentation (FIRE) focused on IoT and cloud computing systems experimenters at large**, given that the present deliverable could be a useful reading for researchers studying alternative IoT technologies and applications, along with indications and requirements towards building/establishing experimental architectures.
- **Members of other Internet-of-Things (IoT) communities and projects (such as projects of the IERC cluster)**, which can find in this document a readily available requirements analysis for experimentation-like IoT services and tools. For these projects, the document could provide insights into requirements and technological building blocks enabling the convergence between utility/cloud computing and the Internet-of-Things for enabling experimentation as a service.

1.4 Terminology and Definitions

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

Table 2: Terminology and Definitions table

Term	Definition
Characteristic	An inherent, possibly accidental, trait, quality, or property of resources (for example, arrival rates, formats, value ranges, or relationships between field values).
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 `(IoT-A, 2013)`. 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 `(IoT-A, 2011)`. **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 `(IoT-A, 2011)`.
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 `(IoT-A, 2013)`.

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) (IEEE, 2007).
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 (PE)	Any physical object that is relevant from a user or application perspective. (IoT-A, 2011). 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. (Haren, 2009)
Resource	Computational element that gives access to information about or actuation capabilities on a Physical Entity (IoT-A, 2011).
Stakeholder	An individual, group, or organization, who may affect, be affected by, or perceive itself to be affected by a decision, activity, or outcome of a project (Project Management Institute, 2013)
Test-bed	A test-bed is an environment that allows experimentation and testing for research and development products. A test-bed provides a rigorous, transparent and replicable environment for experimentation and testing (Gavras, 2010)
Federated test-beds	A test-bed federation or federated test-beds is the interconnection of two or more independent test-beds for the creation of a richer environment for experimentation and testing, and for the increased multilateral benefit of the users of the individual independent test-beds (Gavras, 2010)
Interoperability	The ability of two or more systems or components to exchange information and use the information that has been exchanged (IEEE, 1990)
Experimentation facility	An experimentation facility can be understood as an environment with an associated collection of tools and infrastructure that sits on top of one or several test-beds and can be used to conduct experiments to assess and evaluate new paradigms, architectural concepts and applications (MyFIRE, 2011)

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 (Soukhanov, Ellis, & Severynse, 1992)
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 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 (VE)	Computational or data element representing a Physical Entity. Virtual Entities can be either Active or Passive Digital Entities (IoT-A, 2013).

1.5 Executive Summary

This deliverable is the second and final version of the System Architecture for the FIESTA-IoT platform, which is aiming at federating a large number of test-bed across the planet in order to offer experimenters with a unique experience of dealing and experimenting with a large number of semantically interoperable data sources.

The architecting process leading to this first version followed the Architectural Reference Model methodology promoted by the IoT-A project (Fp7 “light house” project on Architecture for the Internet of Things). It therefore consists of a set of Views that are in tern dealing with “logical” functional decomposition (Functional View), data structuring and annotation, data flows and inter-functional Component interactions (Information View) and ultimately the deployment of those logical components onto concrete software components (Instantiation View).

This architecture is inclusive in the sense it can accommodate under its federation a large number of test-beds with various capabilities (some being semantic-enabled already, some not). It offers full semantic interoperability: all assets of the test-bed (resources, IoT Services, Virtual Entities) are semantically annotated and described; they are searchable using either powerful data query languages or simpler APIs. FIESTA-IoT is therefore able to offer the greatest test-bed agnostic experience to both expert users (semantically skilled) and more basic experimenters as well.

This final version updates the deliverable D2.4(v1) with the latest software developments and technical decisions leading to the concrete system architecture as explained in Section 6. A mapping table also shows how those concrete components maps to the logical FCs introduced in Section 4.

2 INTRODUCTION

The purpose of this deliverable is to provide a first version of the Architecture of the FIESTA-IoT platform. As already covered in Section 1, the whole architectural process has been spread among different tasks of WP2. In order to start the architecture as such, i.e. describing the architectural views and perspectives, we leverage important results already brought by tasks T2.1, T2.2 and T2.3:

- **Stakeholder requirements and Experiment requirements:** collection and Analysis of Functional and Non-Functional Requirements are among the very first phase in the whole architecture process as defined in the ARM-associated Guidance.
- **Description of available test-beds:** Helped formalizing the existing gaps of the available test-beds, especially in regards to the objectives of the project (full semantic support to name just one) – some elements of answer about those existing gaps can be found for instance in Section 4.4

Those two inputs gave us both a top-down (requirements) and bottom-up (analysis of the existing test-beds) approaches to sketch the architecture and in particular the Functional Components (FCs) it is ultimately made of.

Sketching this first version of the FIESTA-IoT Architecture we have been focussing on the very core of it first that are the IoT Service, Virtual Entity, Communication, Management and Security Functional Groups - FGs (see the explanations given in the beginning of Section 3). We have touched as well the Service Organisation and IoT Process Management Functional Group, but those two will be much refined along with the progress of WP5.

Drafting this architecture we have paid particular attention to some aspects and also considered some assumptions/constraints, directly derived from the project objectives, which we describe non-exhaustively below:

- **Compliance to the IoT Architectural Reference Model (IoT ARM)** from IoT-A: "Compliance" is probably too strong in this context but we did try to follow as much and strictly as possible the whole architecting methodology released by the FP7 "light house" project about Architecture for the IoT. The purpose of Section 3 is to go in the detail of the process we followed;
- **Full support of semantic:** The FIESTA-IoT platform is fully semantic-enabled, so we need to put in place all mechanisms needed to support semantic (languages, ontologies and tools). A related consideration is that we naturally do not want to exclude test-beds, which are not semantic-ready from being part of the FIESTA-IoT federation; we had hence to come up with an architecture that needs also to "enhance" those test-beds –capability-wise– in order to pull them to the level of FIESTA-IoT standard;
- **Compliance to FIESTA-IoT set of ontologies:** Test-beds which are not semantics-ready will have to comply to the ontologies defined in FIESTA-IoT so to ensure full semantic interoperability;

- **Logical Functional Decomposition:** The decomposition into components is a logical one; meaning that when the platform physical components will be implemented and deployed, there might not be a direct mapping;
- **Technology Agnostic:** We tried to get as much as possible agnostic to any implementation/design choices. For instance we do not mention Resource Description Framework (RDF) / Ontology Web Language (OWL), RDF/JSON-LD, REpresentational State Transfer (REST) and SPARQL in the text but being possible options among others;
- **Various roles:** We defined different FIESTA-IoT end-user roles and defined the way they will interact with the FIESTA-IoT platform (see Section 4.1);
- **Accommodate different levels of skills:** While most of interfacing between end-users and the test-beds can be handled by a battery of IoT Services (some served by FIESTA-IoT and some by the test-beds themselves) we considered that providing "direct" access to data using complex but powerful data-centric query languages could be a convenient choice fitting certain kinds of actors with high semantic skills.
- **Message Bus:** A Message Bus is used for test-bed to FIESTA-IoT FCs communication (as part of the Communication FG).

This initial version of the FIESTA-IoT architecture has the following limitations (which will be fully covered and complemented with a 2nd – and final – version):

- The Information View provides an extensive (still not exhaustive) list of system use-cases. It does not provide any information about how the data is structured and annotated, as discussions about options concerning that matter are still on-going in WP3;
- The Functional View focuses only on the core-aspects of the platform like communication (between the FIESTA-IoT platform and federated test-beds and federation;
- No interfaces. High Level and Concrete interfaces are still being discussed with relevant WPs. It is worth noting that some interfaces will be directly delivered from the IoT-A D1.5 (Annex C) document (IoT-A, 2013);
- The Function View needs leveraging the – currently on-going – work of WP4 on experiment environment (modelling language, execution) in order to enrich the Process Management Functional Group.

3 INTRODUCTION TO THE META-CLOUD ARCHITECTURE

3.1 Introduction to the ARM and associated methodology

A primary decision of the FIESTA-IoT project is to follow (as much as possible) the IoT *Architectural Reference Model* (ARM) as defined in the IoT-A project (the lighthouse FP7 European Project on IoT Architecture) (IoT-A, 2013). Following the IoT ARM means in particular adopting and following as closely as possible the overall methodology that defines the steps leading to the actual architecture and to stick to the Reference Model as defined in the ARM (especially the Domain Model - DM and the Information Model (IM) which are considered as fixed). The architecting process will then consist of (in order):

- Requirement collection and analysis/processing (called requirement mapping): this part of the process is described in the previous Section;
- Elaboration of the Physical Entity and Context Views;
- Elaboration of the Functional View;
- Elaboration of the Information View;
- Elaboration of the Instantiation View.

The ARM and associated methodology were already introduced quite thoroughly in Deliverable D2.2 (FIESTA-IoT D2.2), so we do not reproduce the same description here with the same level of detail (please refer to that deliverable for a more complete description then) but do remind the key concepts that are referred to in this deliverable.

The main parts of the ARM consist of the Reference Model, Reference Architecture (RA) and a side part consisting of the associated methodology:

1. The IoT *Reference Model* (RM): consists of a set of models (namely the Domain, Information, Functional, Communication and Security/Trust/Privacy Models) that are rather static in the sense they are not expected to be brought any modification. The FIESTA-IoT architecture must comply to those models, especially to the IoT Domain Model (because it identifies the key concepts of the IoT Domain and the relations between those concepts), the Information Model (because it defines a meta-model of how to structure information in the IoT System) and the Functional Model - FM (as it predefines a functional layered architecture for the IoT);
2. The IoT RA consists of a set of Views and Perspectives - as defined by Rozansky and Woods (Rozanski&Woods, 2011)– that actually define the FIESTA-IoT Architecture; therefore the main objective of this Architecture document is ultimately to describe those Views and Perspectives in detail (perspectives will exhibit design and technology choices typically that are used to meet the non-functional requirements along different dimensions like security, interoperability, performance, resilience etc.). The Functional View (see Sections 3.3.1 and 4) will focus on the decomposition into *Functional Components* while the Information View (see Sections 3.3.2 and 5) describes information flows, interaction between components and structure of

information, in compliance with the Information Model (see Section 3.2.2 and Figure 4). The Deployment View (see Section 3.3.6) comes later on and shows how the "logical" components part of the Functional View are deployed within the developed concrete software modules (one module can indeed implement more than one functional component);

3. Guidance: that defines the overall process used to derive a concrete architecture out of the ARM. The requirement mapping exercise following the requirement collection phase (FIESTA-IoT D2.3) in particular helped to derive a preliminary Functional View.

3.2 The IoT Reference Model

In this section we present a quick reminder of the main concepts introduced by the different models of the IoT RM. For more detail please refer either to D2.3 or to (IoT-A, 2013).

3.2.1 Domain Model

The purpose of the IoT Domain Model (Haller et al., 2013) (IoT-A, 2013) as proposed by IoT-A is to introduce the concepts pertaining to the IoT Domain (see Figure 3) and the different relationships between those concepts. Among the different concepts introduced by the DM, it is important to present a reminder of the following main ones:

Physical Entities (PEs): 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. Examples of PEs from the SmartSantander test-bed are buses and traffic lights; from the Surrey test-bed, floors and offices in the ICS building.

Virtual Entities (VEs): VEs are at the heart of an IoT system. They represent the PEs in the virtual world. Aspects of the PE are captured by VE properties, and using sensors and actuators allows one to bridge the physical and logical worlds and then to act on (or read about) properties. At the level of the VE, we will consider a special kind of service, called a VE Service, which is used to manipulate or access those properties. It is important to mention here that it is not compulsory that test-beds provide modelling of PEs into VEs and manage the associated VE Services. However it is highly important that FIESTA-IoT provides the means for doing so. Actually the activity of defining VEs and their associated properties and then binding these properties to sensor readings for instance can be endorsed by other classes of FIESTA-IoT actors (see Section 4.1), the goal for whom, would be to bring added value services to experimenters.

IoT Devices: In FIESTA-IoT, IoT Devices are the hardware supporting the sensing and actuation functions. Micro-controllers, batteries, ROM memory etc. are also Devices (but without the IoT prefix).

IoT Resources: IoT Resources are the software embedded in IoT Devices that provides the raw readings (for sensors) and actuations. The IoT Domain Model advises not accessing directly resources, but on the contrary to access corresponding Resource-centric IoT Services (see below).

IoT Services: We can consider different kinds of IoT services depending on their level of abstraction:

- Resource-centric IoT services (r-IoT Service) are exposing the IoT Resources using standardized interfaces and possibly adding metadata to the raw reading available at the resource level. They all connect to a sole resource (sensor or actuator). For instance getting the reading of a temperature sensor (e.g. via a REST interface) is accomplished through an r-IoT Service;
- VE-centric IoT Services (ve-IoT Service) are associated to the VEs and are used for accessing VEs attributes/status or to access VE-level services not directly connected to VEs attribute or situation. In the Functional View the VE Service FC deals with such accesses. Getting the value of the "hasTemperature" property of a room_VE is an example of a ve-IoT Service;
- Integrated IoT Service are combinations of the two above when combining different readings from different sensors (e.g. "secured" room can depend on lock/unlock status, presence indicators and light status).

Note: In the rest of this document IoT Services and VE Services are to be understood as respectively r-IoT Service and ve-IoT Service.

All kind of IoT Services described above ought to associate with service descriptions that can be used to discover particular sensing/actuation capabilities.

Services: Services (without IoT prefix) are associated to VEs but do not relate to specific properties as illustrated in the example above. Services are not part of the IoT Domain Model but could be added to the global picture for the sake of clarity. For instance autonomous objects (with cognitive capabilities) may expose services that do not relate de facto to any of their VE properties.

User: Different kinds of users are expected to interact with the FIESTA-IoT platform. Section 4.1 identifies different roles and explains what kind of interactions there are implementing with the platform. It is worth noting as well that test-beds may endorse more than one role.

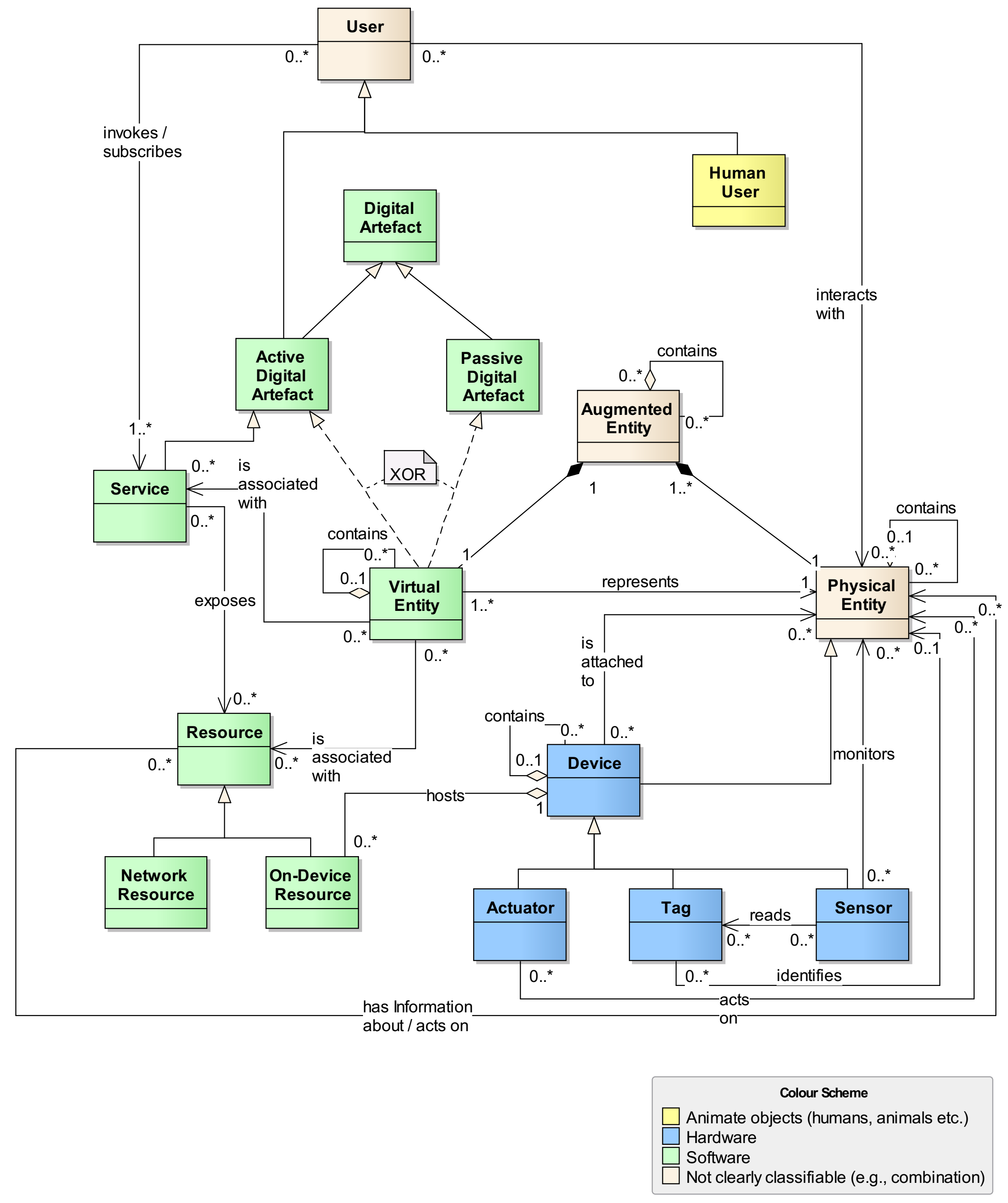


Figure 3: IoT Domain Model (as in IoT-A D1.5)

3.2.2 Information Model

The Information Model (IM) (see Figure 4) focuses on the description of the structure of Virtual Entities as a representation in the cyber space of physical entities. The representation of the information (either it is encoded in eXtensible Markup Language - XML, RDF, binary or any other format) is kept away from the Information Model and left to the architect's choice, as part of the semantic interoperability perspective (as a design choice).

The central part of the IM consists of the structure of the VE which is modelled using a set of attributes and which are associated (via the association relationship) to the

Service Description. These associations are essential in the IoT IM as they make the binding between a VE property – which as the name suggests is at the VE level – and a corresponding Resource-centric IoT Service (meaning a service exposing a resource), which, as the name also suggests, is at the Resource level. This association must be managed in a dynamic way so that the binding between a VE property (attribute) and a Resource (via an IoT Service) can vary along the time axis.

The Attribute is the aggregation of one-to-many Value Containers. Each of those containers contains one single value and one-to-many metadata (e.g. time stamp, location, accuracy, etc.).

VEs are described using a Service Descriptions where each Service would be characterised (e.g. by its interface) or any useful information that a look-up service can exploit (e.g. the FIESTA-IoT IoT Service/Resource Registry, see Section 4.3.5.1).

As an IoT Service is exposing Resources which are themselves hosted by Devices, the IM authorises Service Description to contain 0-to-many Resource Description(s) and Resource Description to contain 0-to-many Device Description(s). The structure of Descriptions is not constrained by the IM and therefore left to the architect's own choice.

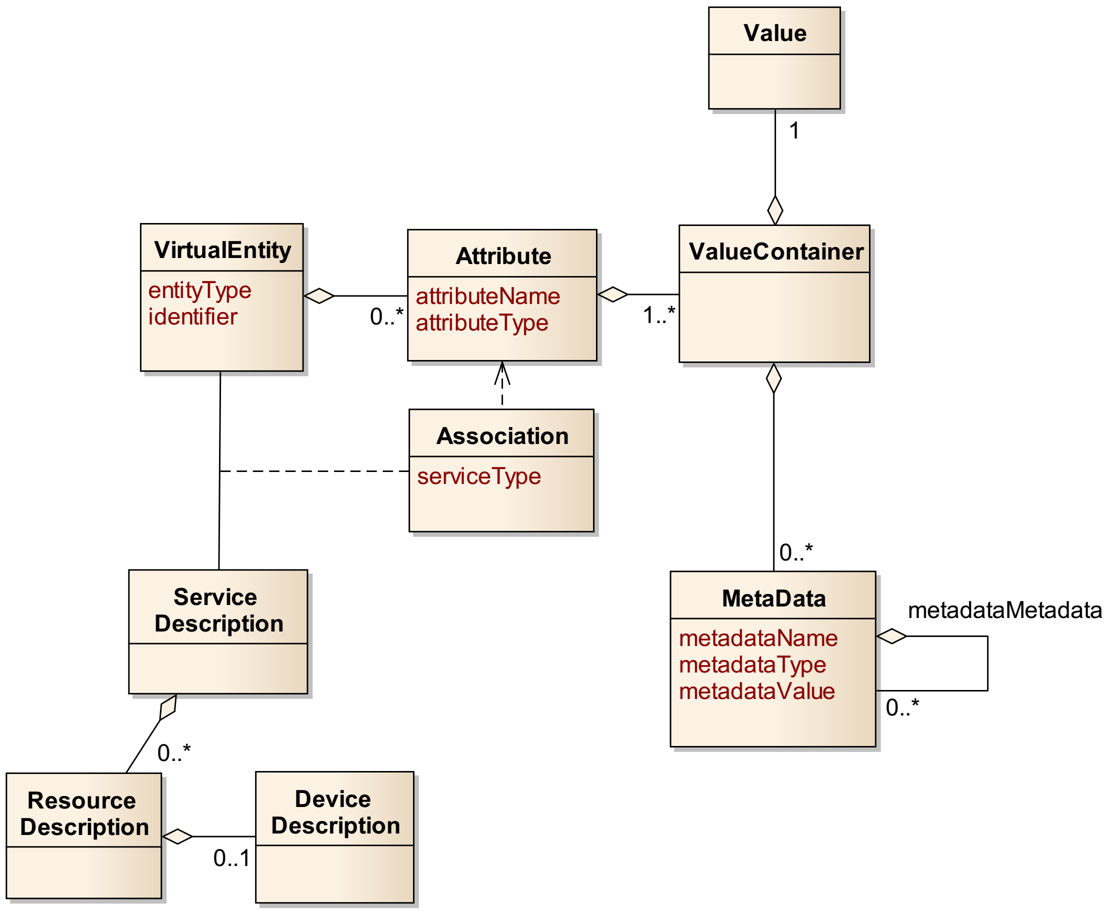


Figure 4: IoT Information Model (as in IoT-A D1.5)

3.2.3 Functional Model

The Functional Model (FM) (see Figure 5 below) proposed by IoT-A corresponds to a service-oriented approach of IoT. It identifies 7 main Functional Groups (FGs) and 2 additional ones that are kept outside the scope of the IoT ARM. The FM has been already described in Deliverable D2.2 (FIESTA-IoT D2.2) but it is worth reminding here the purpose of the different FGs as they are central to the functional decomposition achieved later on in Section 4.

The Functional Groups are defined as follows (IoT-A, 2013):

- ***IoT Process Management FG:*** The purpose of this FG is to allow the integration of process management systems with the IoT platform. For example, the formal definition of a pan-test-bed experiment (as a process) would fall into this category;
- ***Service Organisation FG:*** This FG is responsible for composing and orchestrating services, acting as a communication hub between other FGs. The execution of an experiment described within the IoT Process Management FG would take place in this FG, like any other kind of choreography/orchestration engine. Added-value services like aggregators or reasoners would be also part of this FG as they heavily rely on IoT Services/VE Services;
- ***Virtual Entity FG:*** This FG relates to VEs as defined in the IoT Domain model, and contains functionalities such as discovering VEs and their associations with Resource-centric IoT-services. This FG also allows access to the VE-centric IoT Service offered (formally “associated with”) by a Virtual Entity. In FIESTA-IoT those VE Services can be accessed via a VE endpoint;
- ***IoT Service FG:*** The IoT Service FG contains functions relating to Resource-centric Services. Those services expose the resources like sensors and actuators and provide the means for reading sensor values or actuating. It also contains storage capability functionality. More specifically the IoT ARM states that: “A particular type of IoT Service can be the Resource history storage that provides storage capabilities for the measurements generated by resources”;
- ***Communication FG:*** The Communication FG is used to abstract the communication mechanisms used by the Devices. Communication technologies used between applications and other FGs is out of scope for this FG as these are considered to be typical Internet technologies. A central message bus offering publish/subscribe functionalities would also be part of this FG as we will see when describing the FIESTA-IoT Functional View;
- ***Security FG:*** The Security transversal FG is responsible for ensuring the security and privacy of IoT- compliant systems. The management of security itself is also part of this FG;
- ***Management FG:*** The Management transversal FG contains components dealing with configuration, faults, reporting, membership and state. It should be mentioned here that this FG works in tight cooperation with the Security FG.

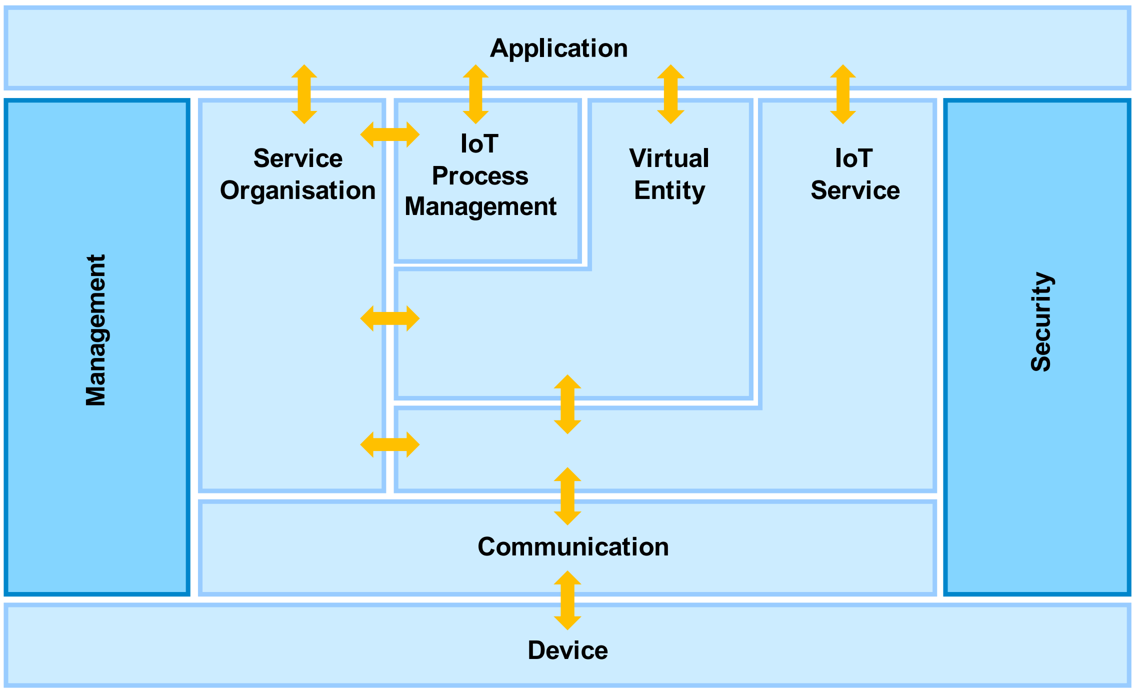


Figure 5: IoT Functional Model

3.3 IoT Reference Architecture

The IoT ARM consists of different views as introduced earlier, i.e. Functional View, Information View and Deployment View. There are two additional Views which are not *stricto sensu* part of the IoT ARM: Physical Entity View and Context View; however they are part of the guidance chapter of the IoT-A Deliverable D1.5 (IoT-A 2013) and those views are clearly essential when elaborating the IoT system architecture. Different viewpoints may be used for describing the different views (see Section 4.4.1 of (FIESTA-IoT D2.2) for the definition of a viewpoint according to Rozanski and Woods (Rozanski&Woods, 2011))

3.3.1 Functional View

In this section we provide a reminder (See Figure 6 below) of the “native” IoT-A Functional View as it shows an illustrative example of functional decomposition resulting from a thoroughly conducted requirement analysis (in that particular case it was the analysis of the IoT-A Unified Requirements). We recommend referring to the IoT ARM final architecture deliverable D1.5 (IoT-A, 2013) in order to get more detail about the purpose of each of the Functional Components shown in that Functional View.

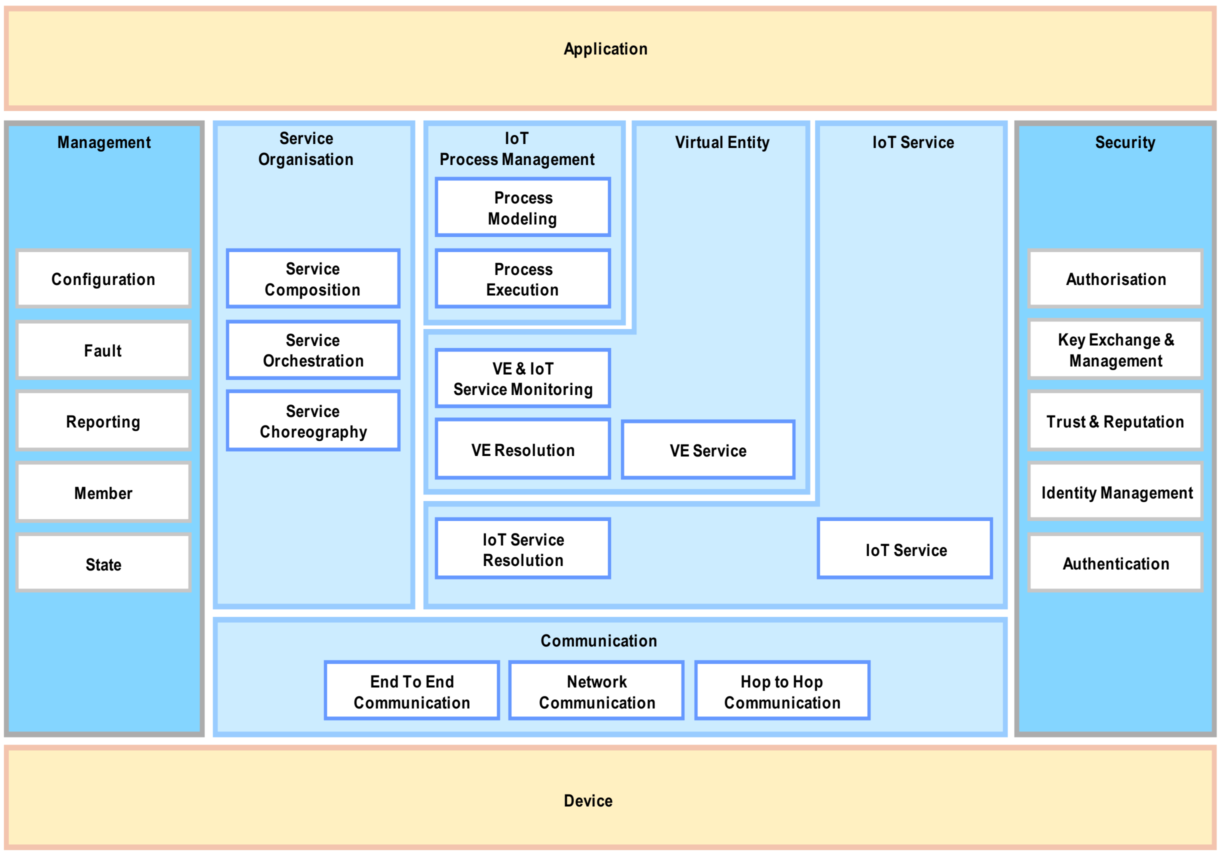


Figure 6: IoT Functional View (as in IoT-A D1.5)

When elaborating the reverse-mapping from various existing FIESTA-IoT test-beds towards the IoT Functional View it is important to try to keep as much as possible to the list of components already identified, however it is perfectly possible and allowed to introduce new ones.

Along with a description of the FCs within FGs it is equally important to get a concise –and still precise– description of the different FCs implemented in the various test-beds and to understand also very clearly how they interact with and position w.r.t. the other components. At this point in time we concentrate on a clear textual description, but in the architecture document System Use-Cases (see Section 5.1) we will formally elucidate those inter-component interactions.

3.3.2 Information View

We use various viewpoints for describing the information view:

- Information flow: shows how information flows between components;
- System use-cases: elucidate usage patterns and explicitly shows interactions between components;
- Structure of information: in the case of FIESTA-IoT we will describe the used ontologies;
- Sequence diagrams.

Note: In this version of the Architecture, only the two first viewpoints are used.

3.3.3 Physical Entity View

This view gives information about the Physical Entities of interest for the IoT System and how those PE are represented within this IoT system:

- List of PEs and their associated properties that can be observed, in addition to the associated actuations that can be applied to them;
- List of devices that are instrumented in order to bridge the physical properties to the cyber world;
- How the sensors/actuators are associated to the PEs and their location; if a PE is in the scope of a camera or if a PE is attached to a temperature sensor for instance.

3.3.4 IoT Context View

The IoT Context View consists of the Context View and the instantiated Domain Model.

According to Rozansky and Woods (Rozanski&Woods, 2011) the Context View describes "the relationships, dependencies, and interactions between the system and its environment (the people, systems, and external entities with which it interacts)". This view focuses on formalizing the boundary between the system and its environment (outside world) and shows how the system interacts with other IT systems, organizations and end-users/administrators etc. using the IoT system.

3.3.5 IoT Instantiation View

The purpose of this view is mainly to elucidate how the Functional Components (and related functionalities) introduced within the Functional View are eventually implemented into concrete components. Key aspects of the Instantiation View are 1/ the concrete architecture which was derived from the logical functional architecture and 2/ the mapping existing between logical and concrete functional components. As written earlier this view is not exactly part of the IoT-ARM model however it becomes essential as soon as more than one implementations of the logical architecture are achieved. It also provides a viewpoint for verifying that all functional FCs and functionalities were covered during the implementation phase.

3.3.6 IoT Deployment View

The Deployment View main purpose is to describe how the different concrete functional components and hardware (including gateways, sensors, actuators etc.) are deployed in the "real life". This view will in particular take care of:

- The physical association between object and hardware;
- The mapping of logical Functional Components (as they can be found in the Functional View) onto concrete Software Components as they were eventually implemented.

4 FIESTA-IOT FUNCTIONAL VIEW

Before describing the different components that are part of the Functional View, it is important to define, on the one hand, the roles and the associated duties that actors involved with the FIESTA-IoT platform may endorse and, on the other hand, the different kinds of platform configurations and capabilities that can co-exist under the umbrella of the FIESTA-IoT federation:

4.1 Roles in FIESTA-IoT

This short section introduces a taxonomy of actors dealing with FIESTA-IoT Federation Platform and explains the main interactions (in the form of Use-cases) taking place between them and the FIESTA-IoT platform as a system. Before getting more into the details of each role, those FIESTA-IoT roles consist of:

- **Raw-Data Producers (RDPs)**: are responsible for producing the raw-data, the metadata of which complies with the FIESTA-IoT ontologies and recommendations. This can happen in two different ways: production of raw-data as Observations to the Message Bus or alternatively answering an IoT service request that is aiming at returning data (still as in the form of Observations). This process does not involve combining several data-sources, hence the adjective “raw”. In addition, Raw-data Producers are responsible for describing and publishing IoT Service and Resource semantic descriptions, either locally or at the FIESTA-IoT level, depending on the Class they belong to (see Section 4.2);
- **Virtualizers**: provide a Virtual Entity layer (Virtual Entity and VE Services) on top of the more basic IoT Service layer. More precisely the roles of virtualizers are to:
    - o *Create VEs*, model them in terms of Properties/Attributes, semantically describe them accordingly following the FIESTA-IoT ontology, and register them to the VE Registry FC;
    - o *Manage Association relationships* between VEs (precisely VE properties) and Resources;
    - o *Define the policy* as for updating the values associated with VE properties and delegates the updating activity to the VE Manager;
    - o *Publish VE-related observations to the Message Bus and store VE-related observation to Data endpoint.*
- **Value-Added Service Providers (VASPs)**: are providing value-added services (e.g. reasoners -other than the FIESTA-IoT-supplied one-, custom analytics algorithms or any useful generic enablers) that in turn can be combined and used by experimenters (or service composers). VASP’s may also consume data but not necessarily. Value-added services can be bound to VE properties creating then de-facto new VE Services;
- **Knowledge producers (KP)**: The purpose of KP’s is to produce higher-order data (or Knowledge) out of the data available at the FIESTA-IoT platform. Different strategies may then be used:

  - Accessing/Retrieving data from the Data endpoint and producing the resulting knowledge to the Data endpoint;
  - Accessing the data from the Message Bus and re-injecting the inferred knowledge to the Message Bus;
  - Provide an IoT Service that leverage existing IoT service provided at the platform level in order to generate new content.

  Regardless to the strategy chosen by the KP, this process of generating new knowledge comes with the creation of virtual resources which have to be described and stored at the IoT Service/Resource Registry FC level. Having done so, they may also want to reflect this new knowledge as new part of the VE properties; they may then create new properties and bind them (via associations) to the newly created IoT Services that when run, result in the production of knowledge that characterises the new VE properties;

- **Experimenters**: They are using the services and are consuming data provided by the FIESTA-IoT Meta-Cloud Data Endpoint FC for the sake of their own business. They do not store any data within the Meta-Cloud but may publish experiment to FIESTA-IoT and make them available to the community as a service.

Test-beds are typically involved at least in the Raw-Data production role; they may be as well endorsing the Virtualizer role, in case they are willing to provide added-value ve-IoT/ Services on top of their basic Raw-Data production activities.

The *Experiment-As-A-Service* (EAAS) concept is captured by the VASP role. An experimenter willing therefore to expose his experiment as a service will create a VA Service description etc… as explained for the VASP role below (See section ???).

Experimenters are FIESTA-IoT platform users (unless they endorse one of the three other roles), while RDP's, KP's, VASP's are FIESTA-IoT platform contributors, providing FIESTA-IoT with added-value (raw data, services, knowledge, Virtual Entities,..).

4.1.1 Raw-Data Producers

Raw-data production is an activity primarily undertaken by test-bed owners, as the test-bed generally own the IoT Resources. Many test-beds are expected to undertake this role and only this role, leaving more sophisticated tasks to other 3rd parties. In FIESTA-IoT, raw-data is produced (or can be requested) in two different ways, as follows:

1. Raw-data is produced by the test-bed as an observation and is fed to the Message Bus (i.e. asynchronous or publish/subscribe mode). This happens at the initiative of the test-bed, usually following a policy implemented within the Resource Manager. Following this production schema, a third-party will be able to get this information either via a subscription to the Message Bus OR by querying data either at the Meta-Cloud Data Endpoint (for Class-II & -III testbeds) or at Class-I testbeds local data endpoint (as all observations produced towards the Message Bus eventually get stored either locally or at the FIESTA-IoT level). In both case the consumption of the raw data by a third party is asynchronous.

2. Raw-data is returned to a third-party upon IoT Service REST request (i.e. synchronous or request/response mode). Two sub-cases can be considered:
    a. Quasi-synchronous: the data is returned by the IoT Service as a service output (answer to the REST request).
    b. Asynchronous: the data is returned via the Message Bus (preferred method when the IoT Service is not aimed at returning only one "current" value but a bigger bunch of data (e.g. historical data)

Apart from producing data, Raw-data producers have to engage in few more interactions with the FIESTA-IoT platform (incl. Step 2 above):

- Testbed registration;
- Resource and IoT Service Registration;
- Answering IoT Service invocation (Class-II and -I);
- Answering a data query (Class-I only).

We have split the RDP role into three categories depending of the level of sophistication they propose natively in their implementation and proposed interfaces.

1. Class-I RDP: provide local storage of data, IoT Service and resource description with semantic support (and FIESTA-IoT Ontology compliant). They also provide a Data endpoint and IoT Service endpoint;
2. Class-II RDP: provide local storage in proprietary format. They provide an IoT service endpoint. To be part of the federation they need implementing a semantic Annotator and storing semantically annotated data at the FIESTA-IoT Meta-cloud data endpoint. Semantic descriptions of IoT Services must be stored at the FIESTA-IoT level as well.
3. Class-III RDP: natively come with IoT Service only. In order to be part of the federation they also need to implement a semantic annotator and to store data at the FIESTA-IoT Level. Semantic descriptions of IoT Services must be stored at the FIESTA-IoT level as well.

4.1.2 Virtualizers

4.1.2.1 Definition and UCs

Virtualizers are responsible for creating and maintaining Virtual Entities in FIESTA-IoT platform, including serving VE Services used for accessing object level properties. The creation of a VE Service involves the modelling of a physical object into the cyber-world and doing so, Virtualizer creates and associates VE Properties (a.k.a. Attributes) to the VE. (S)He also proposes a set of object level IoT Services – called VE Services- that allow to retrieve object level information about the VE (i.e. property values) or actuate on the VE (resulting in physical actuation) by eventually setting up actuation attribute values. Finally Virtualizers publish regularly VE observations to the VE repository in order to maintain historical data about VE properties.

In order to deal with mobility aspects and because values associated with VE properties are based ultimately upon values of Resource somehow associated (or even physically attached) to corresponding Physical Entities, the (potentially temporary) relation between VE properties and underlying resources is captured through a special object called Association. Setting-up and maintaining associations is also part of the Virtualizer's duties.

In order to offer more flexibility we define two classes of Virtualizer: the full-fledge Virtualizer a.k.a. Class-I Virtualizer that fully complies to the definition above and a depleted version that provides only VE Services without neither storing nor producing VE Observations. This second option, a.k.a. Class-II Virtualizers, fits very well Class-II Raw Data Producers who are willing to add an extra VE-layer at lower cost.

The use-cases we are considering for the Virtualizer roles are therefore:

CLASS-II VIRTUALIZER:

- Create a VE, which includes:
  - Model a VE: Following the Information Model described already in Section ???, this activity (which is not supported by any FIESTA-IoT tool) consists of plunging a physical object into the cyber-world by defining the set of object properties that need to be captured in order to model the physical object as much as necessary (application-wise).
  - Create VE Semantic Description
- Register a VE: VEs are registered either locally (in the case of Class-I Virtualizers) or at the FIESTA-IoT VE Registry.
- Create VE Services
  - Which includes creating VE Service Semantic Descriptions
- Register VE Service, which extends Create VE Service: VE Service can be registered either locally (in the case of Class-I Virtualizers) or at the FIESTA-IoT VE Registry.
- Serve VE Services, which extends Register VE Service: a VE Service is a single point of entry for accessing VE properties and applying a get (for sensor and actuators) and a SET (for actuators only). Serving VE Services also means creating a VE service endpoint.
- Create Associations: create a binding between atomic VE Properties and IoT Service ID
- Maintain Associations, which extends Create Associations: maintaining association allows to maintain as well atomic VE properties (meaning properties that are directly associated with one single IoT Service – one single IoT Resource).
- Maintain VE properties values: depending on the Virtualizer own policy, running either internal VE Service logics for composite VE properties (meaning relying on more than one IoT Service) or querying appropriate IoT Service according to associations in order to maintain a cache. Answering VE service request may then rely on the cache values.

CLASS-I VIRTUALIZER:

Includes CLASS-II VIRTUALIZER use-cases +

- Produce and store VE related observations: following the updating of VE properties values, the Virtualizer publishes those information to the message bus and stores observations locally in its VE property triple store.

Figure 7: Virtualizers UML use case

4.1.2.2 Example

In this example we consider the case of a Class-I Virtualizer which aim at creating VEs over a city for the sake of providing sensor-based information. A Grid is laid upon the city map and each of the cell in the grid is represented by a VE. Since a virtualizer's aim is to leverage the Raw-data producer role with the adding of VEs, we do have in this example an existing Raw-data producer that has indeed spread a large number of sensors (in this example SO2 sensors) over the city that provide information about pollution levels around each of the sensor.

Each VE is therefore aiming at making an average of all readings coming from sensors deployed within the corresponding geographical cells, available as a VE service.

In order to build-up a VE, the Virtualizer needs in particular to figure out which are the resources (yellow sensors) that pertain to its quadrant (see Figure 8 below) using some FIESTA-IoT Resource discovery features.

Figure 8: Quadrants & Sensors in Santander

Since the Virtualizer is aiming at providing VE services that can be used either to access/read VE property (sensor) or to set/read value to properties (actuation) there is a need for choosing the strategy to be implemented as for updating the VE properties and serving the VE services:

1. The Virtualizer may choose to regularly update value of properties linked to sensors so that the VE service it is exposing simply return the current value of the property
2. The Virtualizer may also choose to compute the average value at the time access to the corresponding property is requested

Clearly the first version provides the most recent value while the second one provides the current value (knowing that querying all resources and computing the average value still takes some additional time).

Following the UC UML description above, the Virtualizer must go through the following steps:

1. Describing the VE in term of properties, for example:
    a. Geographical position of the cell
    b. Averaged pollution levels in SO2 (gathered from raw data produced in the cell by Raw Data Providers)
2. Registering the VE to the VE Repository, in the same way it will

3. Create, describe and register a VE Service
4. Finding out the set of resources that pertain to a given VE and maintain that set (meaning it has to subscribe to resource creation and deletion and reflect detected changes upon the set). At the initial set-up, the Virtualizer does a discovery of all resources of type “SO2 sensor” matching the geographical position of the cell and create for each one of this resource an association between the VE property and the resource. Then every time the Virtualizer receives a Resource creation/deletion it will automatically create a new association or invalidate existing association.

4.1.3 Knowledge Producers (KP)

The basic idea behind knowledge producing is to create Knowledge out of raw data or any other existing knowledge already inferred and therefore available. Because this new knowledge is going to be stored and published as any raw data observation is, we propose to introduce the notion of Virtual Resource. Virtual Resources are a special kind of IoT Resource which are not hosted by a physical device and which are intended to somehow rely on existing (and actually device-bound) IoT Resources. As any IoT Resource they are semantically described, registered and hence discoverable.

Like any IoT Resource, Virtual Resources are also exposed by IoT Services.

Different techniques may be used in order to “transform” or “infer” new knowledge out of raw-data. This can be achieved e.g. using Reasoners, Rule-based Systems, Complex Event Processing or any analytics algorithm.

If the KP is making use of the built-in FIESTA-IoT reasoner, it will be responsible for editing and injecting the rules within the reasoner for execution (via a dedicated API) and for telling the reasoner which data sets (either Live data or stored data) should the rules reason upon.

Alternatively an Value-added IoT Service (see the Value-added Service Provider) may be also used in the form of an algorithm that accesses IoT services associated with lower-level IoT Resources, using or not using the FIESTA-IoT Service composition toolkit.

Like Class-II & -III RDPs do with “primitive” resources, KP’s which do not manage Semantic registration locally need to register their Virtual IoT Resource Description to the FIESTA-IoT platform as they. Alternatively KP which are managing semantic registrations and knowledge storage locally, may register themselves (their endpoints) to FIESTA-IoT platform as Class-I RDPs do; in this case the local KPs IoT Service/Resource end-point is declared at the FIESTA-IoT level.

We therefore make a distinction here between Class-I KP’s and Class-II KPs. As the description above suggests, Class-I KP corresponds to Class-I RDPs who are able to store and manage locally semantic resource/service descriptions and to serve services locally as well (local REST service endpoint). Class-II KPs do like Class-II RDPs do as far as registration and storage are concerned, meaning they do rely on FIESTA-IoT assets. However they don’t need semantic annotators, as all KP Class-I & Class-II are both semantic-enabled.

In order to be more specific, KPs have to:

- **Create Virtual IoT Resources** which consists in particular of creating Semantic Descriptions. This activity is NOT supported by FIESTA-IoT
- **Create an IoT Service** that exposes the Virtual IoT Resource: this can be done in different ways:
    - To use the Composition toolkit from FIESTA-IoT and create a composite IoT Service that relies on existing IoT Services that expose the Resources the Virtual resource is based on. In this particular case the IoT service Composer FC provided by FIESTA-IoT may be used. Alternatively any other composition tool may be used, however the execution of the composed service would NOT be supported by FIESTA-IoT;
    - To use any sort of service logic that either invokes IoT services or queries/exploits data produced by those IoT Resources. The service logic is then hosted outside FIESTA-IoT and served from outside as well (e.g. in Application FG). This option is using only API for accessing available IoT Services at the FIESTA-IoT interface (REST requests);
    - Use Reasoner available at the FIESTA-IoT platform.
- **Register Virtual IoT Resource** to the FIESTA-IoT Resource & IoT Service registry (Class-II KP) or alternatively (Class-I KP) register the Virtual resource locally AND an IoT Service/Resource endpoint to the FIESTA-IoT Resource & IoT Service broker, and Virtual Resource description locally (like a Class-I testbed does with "primitive" IoT Resource);
- **Register IoT Service** to the FIESTA-IoT Resource & IoT Service registry or locally in case of Class-I KP (the endpoint would be already registered from previous use-case)
- **Serve IoT Service**: this is outside the scope of FIESTA-IoT. The KP must provide an IoT Service endpoint thru which the IoT Service can be invoked in order to retrieve new knowledge e.g.;
- **Produce knowledge**: Class-II KPs publish knowledge towards FIESTA-IoT platform using the Message Bus according to their own production policy (producing knowledge without being explicitly requested by a knowledge consumer to do so is not compulsory though); The sampling and producing rate should be part of the Resource description in order to facilitate the resource selection process. Class-I KPs publish knowledge towards FIESTA-IoT platform using the Message Bus but also store locally the produced knowledge within their own semantic store.

Figure 9: Knowledge Producer UML use case

4.1.4 Value-Added Service Providers (VASP)

Value-Added Service Providers –as their name suggests- are a class of actors that aim at building and providing services through the FIESTA-IoT platform that either is a custom and original service (not relying on the platform enablers) or a service that leverages already existing services (either r-IoT Services, ve-IoT Services, c-IoT

Services or even Experiments (as-a-service) – see Experimenter role below). Those Value-added services are provided from outside FIESTA-IoT platform although been actually described within the FIESTA-IoT resource/IoT Service repository, so that they can be easily discovered by potential service users (e.g. Experimenters).

The set of use-case relating to the VASP role is following:

- **Create VA Service**: consist of implementing a VA Service, using possibly some of the enablers or existing services provided by the FIESTA-IoT platform. Consequently this step may be preceded by a look-up/discovery at the FIESTA-IoT repositories;
- **Create a VA Service Semantic Description**: doing so, the VASP describes semantically the service it is provided to the FIESTA-IoT community, so that it can be easily discovered;
- **Register VA Service**: The service is then formally register to the platform and then part of its discoverable and available assets;
- **Serve VA Service**: This step consists of executing the service and answering any service invocation (behind a REST endpoint).

Figure 10: Value-Added Service Provider UML use case

4.1.5 Experimenters

An experimenter is a class of FIESTA-IoT users that are conducting so-called experiments using the FIESTA-IoT platform APIs. There are basically two classes of experiments in FIESTA-IoT:

1. Standalone experiments: Those experiments are built using the FIESTA-IoT APIs only and lie in the Application FG. In particular they don't find support for execution from the FIESTA-IoT platform and can't be described nor registered within FIESTA-IoT. The results of such experiments -should they be data centric- can't be fed back to FIESTA-IoT message bus, like KP's do for instance;
2. FIESTA-IoT-supported experiments: those experiments are supported by a toolkit, make use of FIESTA-IoT APIs and can be described and published for further reuse (implementing then the concept of an "Experiment As A Service").

4.2 Test-bed taxonomy

We have identified different classes of test-beds that can be potentially part of the FIESTA-IoT federation. We have chosen to define test-beds categories around two different roles which are usually the main focuses of test-beds, keeping hence the Knowledge Producer role outside their scope:

1. **Raw-Data producers:** this is the primary and main focus of test-beds, producing data relying on a usually large set of IoT Resources;

2. **Virtualizers:** this role can also be undertaken by test-beds if they are willing to provide data consumers with a Real-World object/thing layer to. Of course, part of their duties is then to maintain object property values and provide VE Services in order to access those properties or achieve actuation upon those objects/things

The discrimination criterion between those different test-beds will be based on the ability to annotate and store data with semantics and the ability to virtualize.

Those classes are described in the overall architecture schema together with the different components that they have to integrate in order to be FIESTA-IoT compliant:

- **Class-I test-beds {Class-I RDP & Class-I Virtualizer}**: Those test-beds are fully compliant to FIESTA-IoT (still at the condition they do comply with the FIESTA-IoT ontologies (FIESTA-IoT D3.1)) and do not necessitate any integration of any additional components. They store locally semantically annotated data. They also manage locally VE semantic descriptions, IoT Service/Resource semantic descriptions and endpoints. They provide data endpoints for direct data queries (either raw data or VE-related). As we can see, all descriptions are semantically described and compliant to the FIESTA-IoT schemas;

- **Class-I$^-$ test-beds {Class-I RDP & Class-II Virtualizer}**: The difference with Class-I is that Class-I$^-$ do not store VE-related data, and relies on FIESTA-IoT as for their storage;

- **Class-II test-beds {Class-II RDP}**: Those test-beds were initially not semantic-ready; still they used to store their data locally with some annotations in a non-semantic format (e.g. Json). In order to comply with the FIESTA-IoT rules, they will have to integrate/implement few functional components (see some examples of such additional components in Section 4.4). Class-II test-beds will replicate their data, after it has being semantically annotated according to the FIESTA-IoT schema, to the FIESTA-IoT data repository. As a consequence they do not offer a data endpoint locally; queries to data originating from that test-bed will be answered by the central FESTA-IoT data repository directly;

- **Class-II$^+$ test-beds {Class-II RDP & Class-II Virtualizer}**: They are test-bed Class-II which were not considering natively virtualization but which have considered endorsing the Virtualizer role when joining the FIESTA-IoT federation; This kind of test-bed implements the bare minimum as they provide the VE Service endpoint but do not handle the storage of any VE-related data;

- **Class-III test-beds**: those test-beds were initially neither semantic-ready nor storing any data locally; In order to be part of the FIESTA-IoT federation they will have to integrate few additional FCs;

The following section describes all Functional Components that are considered for this first release of the architecture. This list might be updated when updating to the final version of the FIESTA-IoT Architecture.

4.3 Functional Group and Component Descriptions

Remember that the Functional View at this stage is a logical view. An existing or to-be-developed component may actually endorse more than one logical role spanning even more than one FG, for instance if a decision is taken to implement only one Registry dealing with both VEs and IoT Service/Resources.

4.3.1 Management FG

This FG is made of 2 FCs as shown in the Figure 11 below.

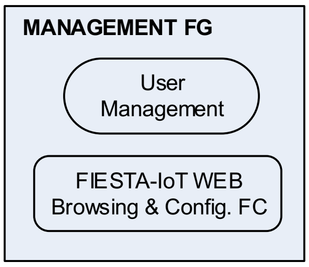


Figure 11: Management FG and FCs

4.3.1.1 User Management FC

This component is responsible for registering new FIESTA-IoT users within the FIESTA-IoT management database. FIESTA-IoT users can sign up to use the FIESTA-IoT services via a Graphical User Interface (GUI); they can also use the GUI to update their personal user information. The registration process includes the issuing of security credentials (however the management of keys and authentication/access enforcement points are at the Security FG side).

4.3.1.2 Web Browsing & Configuration FC

This FC is a web application that builds and provides the FIESTA-IoT actors with a graphical interface for interactively discovering, manipulating and configuring (Create, Read, Update and Delete - CRUD operations) Virtual Entities, Resources and Services. It heavily relies on the two, respectively VE-centric and IoT Service/Resource-centric, Web Front-end Sub-FCs.

4.3.1.3 Experiment WEB Browsing and Configuration FC (covers EMC & Experiment Editor)

This FC allows creation, configuration and execution control of the experiment via a WEB portal.

4.3.1.4 Testbed WEB Configuration FC

This FC offers an interface to testbed for registration to the platform. It also provides means for IoT Resource registration.

4.3.1.5 Testbed & Platform Monitoring

This FC offers a high-level view about the status of different testbeds. More specifically this FC allows experimenters to know which testbed is pushing data to the platform and which resource has published observations recently. It also allows a testbed to check whether their data are properly inserted to the platform.

4.3.1.6 TPI Configurator

The TPI Configurator is a web interface used by the TP in order to define a data production policy to be used by the TPI DMS when it operates in polling mode.

4.3.2 IoT Process Management FG

4.3.2.1 Experiment modelling FC

This component allows modelling either through graphical interface or scripting an experiment. It relies in particular on the *Domain Specific Language* (DSL) created towards facilitating the experiment description. An Experiment description contains a modelling object that defines metadata about the experiment and a service object that includes parameters that defines how and when an experiment should be executed and the query that defines the experiment.

The Experiment Modelling FC, interacts with Experiment Execution Engine and provides it with the desired parameters for execution from the DSL. This component also has access to storage capabilities, so that the experiment description and the results of the experiment can be persisted.

4.3.3 Service Organisation FG

This FG and the IoT Process Management FG (see next section) are dedicated to components that are used as tools for modelling, creating and supporting the execution of -on the one hand- experiments that are used by experimenters to access and make use respectively of data available at the FIESTA-IoT platform (and federated test-beds) and the myriad of IoT services also available. On the other hand they can be used by added-value service providers in order to create added-value services that in turn can be used for the creation of smarter experiments.

The FCs of the Service Organization FG will be confirmed/enriched in a second version of the deliverable as the technical work of the corresponding Work Package makes progress.

4.3.3.1 Experiment Execution Engine

This component is responsible for executing the experiment (see above) based on the parameters specified in the experiment DSL on the Meta-Cloud. A web console supports this component an enables experimenters to perform tasks such as subscribe/unsubscribe to certain discoverable experiments, start/pause the execution of an experiment and to view basic statistics about the experiment execution.

4.3.4 Virtual Entity FG

In this Functional Group we present the VE Registry Functional Component and also the sub-FC it is made of.

4.3.4.1 Virtual Entity Registry FC

This main Virtual Entity Registry[1] FC allows the creation/management of VEs and management of associations between VEs and IoT Resources (via IoT Services). Through an association, a VE Property is bound to one or more IoT Services that expose underlying resources (as the Resource / IoT Service associations are one-to-many).

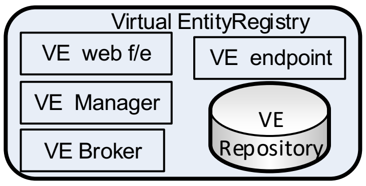
It also supports the storage and look-up of any Virtual Entity semantic description managed within FIESTA-IoT by the Virtualizers actors (whether the role is endorsed by a test-bed or by a 3rd party). It also stores information about VEs and the look-up and retrieval of that information (information about physical objects and properties).

This component provides in addition a VE endpoint that exposes VE Services to FIESTA-IoT users. VE Services are used to work at VE level directly, meaning that they allow to act (get/set) on VE properties without any explicit reference to the underlying IoT Services/Resources.

Accessing VE should be also possible in an interactive mode as it may be necessary to browse VE before designing experiments. See Management FG (Section 4.3.1) for browsing capabilities (at VE and Resource level). VE Registry is relevant to experiments that consider the nature of manipulated physical objects. A THING-agnostic experiment focussed on data only (statistics, machine learning etc.) is most probably using the IoT Service Resource Registry FC and Data Repository FC only.

The VE Registry FC is made of the following sub-FC (see also Figure 12 below):

- **VE Manager Sub-FC**: offers interfaces for creation/registration, association management and VE look-up;
- **VE Web Front-end (f/e) Sub-FC**: supports the VE/IoT Service/Resource web-client, situated at the Management FG side;
- **VE Broker Sub-FC**: is used in order to forward look up requests to local Class-I VE endpoints after they have registered themselves to this component;
- **VE Endpoint Sub-FC**: this component provides an entry point to accessing VE Services associated with VEs managed within FIESTA-IoT (REST based for instance);
- **VE Repository**: is a database that contains VE descriptions, associations and VE related data (properties and their values)



[1] This FC corresponds to the VE Resolution FC in Figure 6

Figure 12: VE Registry and sub-FCs

4.3.4.2 Virtual Entity Manager Sub-FC

This Functional Component is responsible for many aspects (see below) relating to the Virtual Entities and offers many corresponding supporting interfaces:

- Updating VE properties automatically according to Virtualizers instructions.
- Creation of VEs and VE description set-up;
- Creation and management of Associations;
- Accessing VE properties/values on VE-endpoint behalf;
- Running VE Services on VE-endpoint behalf.

4.3.4.3 Virtual Entity Web front-end (f/e) Sub-FC

This FC supports the FIESTA-IoT user for discovering/browsing via a GUI the VEs federated by the FIESTA-IoT platform and possibly managed by the underlying FIESTA-IoT test-beds. It can also be used as a front-end to the VE creation and association registration processes. Those three aspects are described in more detail in the three following sub-sections. This component serves also the general purpose Web client for graphical/interactive access to VEs, Resources and IoT Services located at the Management FG side.

4.3.4.3.1 Browsing Virtual Entities

Browsing a VE means discovering VEs using an interactive and graphical web client in order to gain access to VE's characteristics:

- VE unique identifiers;
- VE semantic descriptions;
- VE properties and types;
- VE Services that can be used in order to get a property value (e.g. underlying sensor value) or set a property value (e.g. underlying actuator value);
- Bindings between VE properties and the underlying IoT Service and resources they are exposing (it should be then possible to go deeper in the detail of the resource/IoT Service descriptions themselves (see the Association set-up below)).

VE maybe nested in different levels, the containment relation between VE shall be reflected as well (in the spirit of nested folders in the case of file system browsing).

For example, in Santander, one may create a VE for a bus line with some attribute about its itinerary that consists of many Bus VEs and Bus Stop VEs with associated properties.

4.3.4.3.2 Creating Virtual Entities

Creating a VE is a process carried out by virtualizers using the VE web front-end FC. They may also create a VE using the VE Creation API offered by the VE Manager

Functional sub-FC of the VE registry FC (see Section 4.3.4.1 below). The purpose of the registration is to describe the Virtual Entity semantically and to register it within the VE Registry FC (and associated database). The VE semantic descriptions must include in particular:

- A unique VE identifier;
- Information about any "containment" relationship to any already existing VEs (in case of nested virtual entities);
- Information about the VE properties;
- Information about the Physical Entity that the VE is representing in the Cyber world (unique ID –if any-, object class, etc.).

4.3.4.3.3 Creating Associations

Creation/Management of Association is the third aspect tackled by the VE Web Front-end. It supports the manual association of VE properties to IoT Services / Resources. Association creation is also possible via a dedicated API provided by the VE Manager Sub-FC of the VE Registry FC (see Section 4.3.4.2)

4.3.4.4 Virtual Entity Broker Sub-FC

The broker sub component is used to forward any VE-related request to all Class-I test-beds under the FIESTA-IoT federation. It is also responsible for compiling answers from them with local answers.

4.3.4.5 Virtual Entity endpoint Sub-FC

The Virtual Entity endpoint offers an endpoint (for instant REST-based) to accessing VE properties and VE Services under the FIESTA-IoT federation. It relies on API provided by the other sub-FCs, in particular the VE Manager Sub-FC.

4.3.5 IoT Service FG

In this Functional Group we are addressing two main Functional Components which are respectively the IoT Service/Resource Registry and the Meta-Cloud Data Endpoint. As we did for FG, we describe now in turn the FCs and associated sub-Functional Components (Sub-FC).

4.3.5.1 IoT Service/Resource Registry

The IoT Service/Resource Registry[2] FC provides an API for registering a Resource and the associated IoT Services within a registry with associated metadata. This particular API can be used either for registering composite IoT services defined by the IoT Service Composer FC (Added-value Services in particular) or by test-beds which do not handle locally the definition of the IoT Services that expose their resources (See Class-II & -III Test-beds scenarios).

This registry allows also to look-up IoT Services exposing resources based on various criteria (based on metadata).

[2] This FC correspond to the IoT Service Resolution FC in Figure 6

Note: At the time this version of the deliverable is written, there is no decision yet how the registry will eventually be implemented. However if all IoT Services and resource descriptions are stored as RDF triples and stored inside a triple store, a query may be in the form of a SPARQL query (for actors able to handle the complexity of SPARQL queries) via a dedicated SPARQL endpoint. Another possibility (which can perfectly co-exist with the previous one) is to provide an IoT Service for querying IoT Services using a When/What/Where interface[3]. Such an interface is much less powerful than a SPARQL interface, however it has the advantage of being easy to use, especially by non-semantic experts. Its other advantage is that it can be used by test-beds which are not natively using RDF, but rely on other standards e.g. JSON. A look-up request would be analysed by the test-bed, and the Semantic annotator would be then used to translate the test-bed answer into RDF for instance.

This registry is a federating one, meaning that it will forward requests to the test-bed registries and compile/aggregate answers coming back from them, with additional constraints like for instance restricting the volume of answers.

This component is made of a few sub Functional Components as follows (see also Figure 13 below):

- **Resource Broker:** receives IoT Service/Resource look-up queries from FIESTA-IoT end-users (in the broad sense), dispatches to Class-I test-bed (when applicable) and compile answers. It also provides a query interface (e.g. SPARQL endpoint) in order to deal with those kinds of requests;
- **Resource Manager:** manages the reservation of IoT Service/Resources, reports on availability, stores/retrieve semantic description of IoT Services and Resources;
- **IoT Service/Resource Web front-end Sub-FC**: supports the VE/IoT Service/Resource web-client situated at the Management FG side.

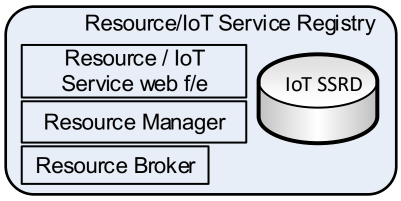


Figure 13: IoT Service/Resource Registry and sub-FCs

4.3.5.2 Resource Broker Sub-FC

As introduced above the Resource Broker component is the front-end offered by the IoT Service/Resource Registry to the FIESTA-IoT end-user for accessing data stored at that side and look-ups; typical end-users being experiments or Web IoT-Service/Resource Web front-end for instance. As for accessing the data from the database, the Broker will rely on the Resource Manager component.

[3] E.g. Give me all observation relating to a phenomenon `x` (What) occurring in location `y` (Where) at time-interval `t` (When)

4.3.5.3 Resource Manager Sub-FC

The Resource Manager is the only point of entry to the database that stores IoT Service and Resource descriptions. It is envisioned that it will use a Jena interface to the data store (wherever it is Triple store or SQL-like database) for storing, managing and retrieving data. This component serves ultimately both the Web Front-end and Resource Broker Sub-FCs.

4.3.5.4 IoT Service/Resource Web Front-end Sub-FC

This FC supports the FIESTA-IoT user for discovering/browsing via a GUI the IoT Services and Resources federated by the FIESTA-IoT platform and possibly managed by the underlying FIESTA-IoT test-beds. It can also be used as a front-end to the registration processes associated with IoT Services and Resources. This component serves also the general purpose Web client for graphical/interactive access to VEs, Resources and IoT Services located at the Management FG side.

4.3.5.5 Meta-Cloud Data Endpoint FC

This component[4] offers FIESTA-IoT users interfaces to query data managed within FIESTA-IoT (either it is locally stored and managed at the FIESTA-IoT side or remotely stored and managed by Class-I test-beds). It is about exclusive interaction between FIESTA-IoT user and stored data, meaning here annotated data coming from raw-data producers but also knowledge coming from knowledge producers.

The Meta-Cloud Data Endpoint FC aims at managing and storing data published by the Class-II & -III test-beds and is the central point where data queries are resolved. When a data request comes, it will resolve it locally (purpose of the Data Manager sub-FC, see Section 4.3.5.5.1) and also propagate the request to the Class-I test-bed data endpoint (Class-I test-bed data-endpoint are registered to the FIESTA-IoT platform beforehand).

Two types of interface are provided at this level:

- A pure data-query specialised one (e.g. SPARQL) which directly addresses the database;
- A less powerful but more user-friendly set of APIs e.g. What/Where/When-type API (WP3 API) that exploits specific metadata.

This component is made of several sub-components (see Figure 14).

4.3.5.5.1 Data Manager Sub-FC

Data Manager Sub-FC is responsible for storing the data within the Semantic Data Repository sub-component, upon Class-II & -III test-beds' request and to answer incoming data-query requests along the two interfaces afore-described.

[4] Strictly speaking this FC would be considered as an IoT Service in Figure 6

4.3.5.5.2 Semantic Data Repository Sub-FC

Semantic Data Repository Sub-FC is storing data managed at FIESTA-IoT level (coming from Class-II & -III test-beds).

4.3.5.5.3 Data Broker Sub-FC

Data Broker Sub-FC forwards the data queries to the FIESTA-IoT Class-I test-beds (e.g. towards their SPARQL endpoints or local WP3 APIs) and aggregates answers. It also manages the volume of received data according to experiment constraints. It may have also to prioritize according to the origin of data as well.

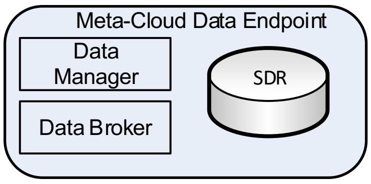


Figure 14: Meta-Cloud Data Endpoint and sub-FCs

4.3.5.6 Reasoner FC

The Reasoner is a functional component that is aimed at applying a set of rules to a pre-loaded data set (resulting from a SPARQL request to the meta-cloud data repository) in order to produce new data. This new data can be then either stored to the Meta-cloud Data repository (if the data is of interest for the community) or otherwise just pushed to the message bus along a topic pre-defined by the experiment when invoking the reasoner. Few system use-cases in Section ??? illustrate the different options. The reasoner works on historical data only as it uses data extracted from the Meta-cloud data repository.

4.3.5.7 Analytics FC

The FIESTA-IoT Analytics component seeks to provide the experimenter with a set of data analysis methods as a web service. In particular the Analytics component provides a range of techniques that can be grouped in the following categories, namely: 1) pre-processing methods, 2) supervised learning, 3) unsupervised learning algorithms and finally 4) other techniques and methods (examples of techniques belonging to this class are spectral and dependence estimation methods). An overview of the techniques for each data processing category is shown in Figure 15.

Figure 15: Data analysis tools categories

4.3.5.8 Semantic Validator FC

The Semantic Validator is a platform component meant at checking randomly that data pushed to the IoT Registry is valid, meaning that it complies with the FIESTA-IoT ontology.

4.3.6 Communication FG

4.3.6.1 Message Bus FC

A message bus provides the FIESTA-IoT eco-system with a communication channel following the Pub/Sub paradigm. This communication channel is used for various purposes:

- Publishing data to the FIESTA-IoT Meta-Cloud Data Endpoints (Class-II & -III test-beds);
- Back channel used by IoT Services for eventually answering data upon IoT Service invocation, depending on their ultimate purpose (Actuation IoT Service would probably not use this back channel for instance).

The message bus in addition to delivering the data also provides an interface for subscription management and publishing.

FIESTA-IoT data consumers (i.e. knowledge producers, experiment), can then subscribe to data according to some filtering/routing criteria (e.g. topics or conditionals) and access data in an event based manner.

The data produced by all-Class test-bed must comply with the FIESTA-IoT ontologies (FIESTA-IoT D3.1.1).

4.3.6.2 TPI DMS

The TPI DMS is a component that plays an essential role in the process of flowing data from the RDP towards the platform. It works in two different modes:

- Polling: in the polling mode, the TPI DMS relies on the polling policy that has been defined by the TP using the TPI configurator. It will then essentially get data from the RDP and publish it to the Message Bus. Depending on the nature of the TP, this data can be a bunch of data (historical data), the most recent or even the actual current data depending on how the TP implements its *Testbed Provider Services* (TPS);
- Pushing: in that case there is no policy involved (at least at the platform level). The TP decides on its own how and how often data should be pushed towards the platform. In that second case the TPI DMS is just a relay between the RDP and the Message Bus.

4.3.7 Security FG

The Security FG follows quite strictly the recommendation for IoT-A native FCs as shown in Figure 6. The Figure 16 below introduces the Functional Components used for dealing with Security in FIESTA-IoT. This FG is made of several components already identified in the IoT-A Native Functional view:

- AuthN: Authentication of FIESTA-IoT Users;
- AuthZ: Access-Control policies, decision and enforcement;
- KEM: Key Exchange and management;
- TTP: Trusted Third Party (for generation of Security Certificates).

Those different FCs are described here after in more detail.

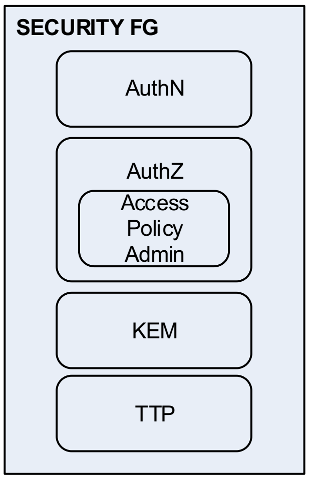


Figure 16: Security FG with FCs

4.3.7.1 Authentication (AuthN) FC

This AuthN[5] component is responsible for enforcing the authentication of registered FIESTA-IoT users. Based upon a user credential passed to the AuthN FC we can make an assertion about the identity of the user i.e. that they are a known FIESTA-IoT experimenter (i.e. AuthN (user_credential) → assertion).

The AuthN component interacts with the User Management FC when the user registers in order to produce the user credentials (such credentials can take multiple forms e.g. both username/password and an X509 certificate).

When access to a resource or service is requested by a user, the request is captured by a decision point as whether to grant or deny the request based upon whether the requester is authorized to do so. At this point, the AuthN FC can be contacted by the decision point to assert the authenticity of the requester.

4.3.7.2 Authorization (AuthZ) FC

The AuthZ[6] component makes decisions about access control requests (intercepted at access decision points) based upon Access Control Policies (ACPs).

Access control can be applied at the level of look up—for instance:

- for a request to search for a list of VEs/resources in a specific domain or part of a domain (test-bed) meeting some criteria; or
- for direct access to resources via their interfaces.

Access is denied if the assertions made about the request and requester do not comply with the Access Control Policy. Usually the three parameters of an access request include an assertion (data guaranteeing for the occurrence of an authentication of a user client at a particular time using a particular method of

[5] AuthN correspond to the one proposed in Figure 6 as part of the Security FG

[6] AuthZ correspond to the one proposed in Figure 6 as part of the Security FG

authentication), the targeted resource and requested operation (read or write for instance).

4.3.7.3 Access Policy Administration FC

This sub-component of AuthZ FC provides a Policy Administration Point to the AuthZ FC where access policies are defined and managed. A GUI is provided to the owner of resources who wish to protect access; using the GUI the owner can create new access policies, attach them to resources, update access policies and delete access policies.

The Access Policy Administration FC will typically be used by test-beds signing up to FIESTA-IoT and registering their resources. When they register a resource, they can assign an access policy (applying a default rule e.g. all FIESTA-IoT experimenters can perform a read operation on the API, or define their own policy using the GUI).

Access Policies defined in this FC will be followed by the AuthZ component when making access control decisions.

4.3.7.4 Key Exchange and Management FC

The Key Exchange and Management (KEM) component manages the exchange of security information between two parties. In particular, it ensures that the keys required to construct a secure and trusted communication channel is carried out in a secure manner.

When a test-bed registers with FIESTA-IoT, the keys required to establish the secure channel between FIESTA-IoT and the test-bed are created and exchanged between the two parties.

In FIESTA-IoT, the HyperText Transfer Protocol Secure (HTTPS) communication protocol will be used to secure communication between test-beds and FIESTA-IoT services and components.

4.3.7.5 Trusted Third Party Authority (TTP) FC

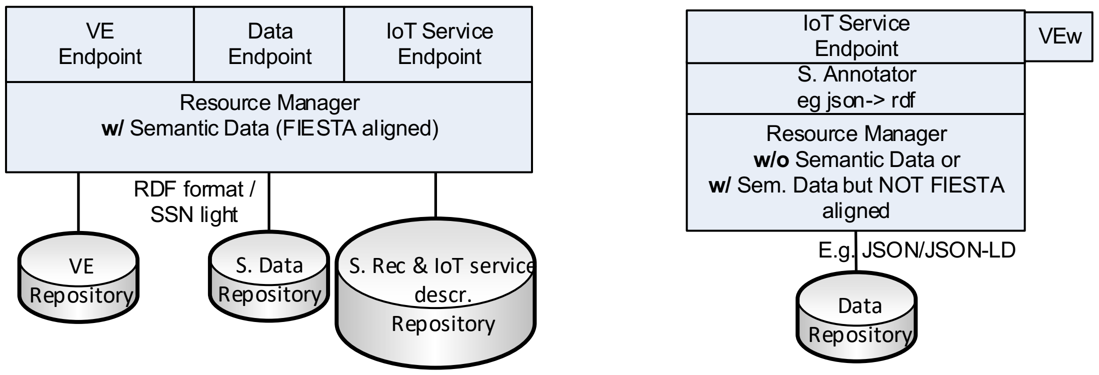
The TTP FC is used to generate RSA key pairs used in certificate based authentication and the subsequent x509 certificate. Such credentials may be needed by both experimenters signed up to use FIESTA-IoT services (hence, experimenters can be given a certificate during the sign-up process) and test-bed interfaces, and also for the trusted communication channels between test-beds and FIESTA-IoT (as described in the KEM FC).

4.4 Impact on existing test-beds (impact of federation)

Adopting the functional decomposition and federation principles described in the former sub-sections has an impact on the existing test-bed inner architectures and initial technology choices (formerly made outside the federation).

In order to align with both FIESTA-IoT requirements and proposed architecture those test-beds have to bring modifications either to their existing components or to their architecture by adding new components.

The following of this section provides more detail about those modifications and in particular provides a list of FCs that might be needed at the test-bed level to ensure it becomes part of the FIESTA-IoT ecosystem. The Figure 17 below shows how those components need to deploy in case of Class-I & -II test-beds


VE Endpoint
Data Endpoint
IoT Service Endpoint
Resource Manager w/ Semantic Data (FIESTA aligned)
RDF format / SSN light
VE Repository
S. Data Repository
S. Rec & IoT service descr. Repository
IoT Service Endpoint
VEw
S. Annotator eg json-> rdf
Resource Manager w/o Semantic Data or w/ Sem. Data but NOT FIESTA aligned
E.g. JSON/JSON-LD
Data Repository

a) Class-I Test-bed **b) Class-II Test-bed**

Figure 17: Test-bed upgrades for FIESTA-IoT compliance

It is worth noting in addition that semantic-enabled test-beds, and non-semantic enabled test-bed using a semantic annotator MUST comply to the FIESTA-IoT ontologies in order to maximize semantic interoperability.

- **VE endpoint:** The VE endpoint provides a unique access point to VE Services that provides access to VE properties (e.g. via a REST interface) or any other services a VE may provide. This VE endpoint is an alternative to the implementation of VEs as autonomous software entities that would provide the same kind of REST interface; instead of implementing multiple VEs, the test-bed may provide a VE front-end that takes care of implementing for instance the binding between VE properties and underlying associated test-bed resources. Alternatively it may implement the same functionality than the VE manager at the FIESTA-IoT level i.e. polling regularly resource readings and updating VE properties accordingly (all being stored locally as well, like FIESTA-IoT does with the VE Repository);
- **IoT Service / Resource endpoint:** this component does the same duty as the VE endpoint at the IoT Service / Resource level. The test-bed then offers a unique access to querying resources (sensors) / updating resources (actuators) via a standardized interface, e.g. REST; the IoT Service may use the Message Bus as a back channel for sending the result of the request back to the requesting party;
- **Semantic Annotator**: the Semantic Annotator (S. Annotator) is used to statically translate data and metadata expressed in a non-semantic supported format (e.g. Json format) into a semantic format such as e.g. RDF/OWL. Semantic data is then injected to the Semantic Data (S. Data) Repository as a set of triples, at least at the interface of his component. The database itself can be either full triple store or SQL database offering a SPARQL endpoint. As already stated earlier, the Semantic Annotator MUST comply to the FIESTA-

IoT agreed ontologies in order to maximize semantic interoperability `(FIESTA-IoT D3.1.1)`;

- **Resource Manager (with Semantic Data)**: this component is responsible for enforcing the publishing policy of data from the test-bed point of view:
    - Storage: data collected from sensors is enriched with metadata and stored in a triple format through or instance a JENA Interface to a Jena-enabled database (either full triple store or relational database). In addition, the resource manager may need to be able dealing with complex (enough) publishing strategies dictated by the experiment(er) (e.g. read and store value from resource xyz from 8am – 6pm all working days every 2 minutes) whether the final data destination is the local test-bed storage or the FIESTA-IoT level message bus;
    - Manage sampling/publishing rate: This component has also to deal with special request from experiments to sample and publish data at certain rate, with start/end time. Potentially several request need to be accommodated so that each experiment can retrieve data that fit its own specifications;
    - Publishing: If requested to do so the Resource Manager will also publish the annotated data to the Message Bus. Like for Storage, the Resource Manager might have to follow a publishing strategy dictated by the experiment(er);
    - Event-based data publishing: publishing data to the message bus along the topic specified by the subscriber;
    - Support other components: finally the Resource Manager supports the VE endpoint and IoT Service/Resource endpoint in retrieving needed data.
- **Resource Manager (without Semantic Data):** its role is quite similar to the previous one but without semantic data support:
    - Storage: Storage of data is made in a non-semantic, yet structured data format;
    - Manage sampling/publishing rate: This component has also to deal with special request from experiments to sample and publish data at certain rate, with start/end time. Potentially several request need to be accommodated so that each experiment can retrieve data that fit its own specifications;
    - Publishing: The resource manager needs to replicate all locally stored data in the cloud, relying then on the S. Annotator for making the data FIESTA-IoT compliant with semantic;
    - Event-based data publishing: publishing data to the message bus using the S. Annotator and along the topic specified by the subscriber during Resource reservation process;
    - Support: another option for the Resource Manager is to support data queries and responding with semantic data going through the Semantic Annotator.

- **Data endpoint (e.g. SPARQL) (a.k.a. TPS):** they are provided by test-beds that store semantic data within dedicated semantic storage or just manage to cache more recent data. Such endpoint is part of a full semantic-enabled storage package, which can in addition provide a JENA[7] `(Jena)` interface for creating easily Semantic Data which is then serialized and stored as either RDF triples or table entries (in the case of relational database back-end).
- **VE Wrapper (VEw):** For test-beds which are not willing to deal locally with Virtual Entities, the VEw allows for publishing VEs to the FIESTA-IoT VE repository (VE and properties) and to bind VE properties to IoT Services. Being present at FIESTA-IoT level such VEs can be discovered and reference to proper IoT service be found.

Figure 18 below provides a complete picture of the first version of the FIESTA-IoT system architecture.

Figure 18: FIESTA-IoT System Architecture (v2)

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

5 FIESTA-IOT INFORMATION VIEW

Based on the Information Model (part of the IoT Reference Model, see Section 3.2.2) the Information View aims at providing details about how the information is actually coded, serialised and handled within the target IoT system. Indeed, the IM does not give any indication on how objects, resources, devices and associated attributes and description must be encoded. It stays at an upper level, giving only indications concerning what needs to be modelled and which inter-concepts associations need to be implemented within the IoT system. Implementation matters stay at the architect side, who in turn enjoys some freedom as far as him/her choices remain compliant to the IM constraints. This section now adopts several Viewpoints according to the Rozanski and Woods (Rozanski&Woods, 2011) terminology and elucidates several aspects pertaining to information and information flows within the FIESTA-IoT architecture.

5.1 System Use-cases and Sequence Diagrams

This section aims at identifying typical interaction pattern between the FC described in the Functional (logical) View (Figure 18).

Before starting with the explanation of the system use cases, it is worth highlighting a couple of assumptions that we have made so as to keep the figures as simple as possible.

1. Experimenters might have different connection points to the FIESTA-IoT infrastructure (i.e. Web Browsing & Configuration FC, Experiment Interpreter FC, VE registry FC, Meta-Cloud Data Endpoint FC or IoT Service Registry FC). Since the three last ones will be the actual cornerstone of the meta-architecture (the former ones will, after all, have to go through these FCs to access the underlying test-beds), for the sake of simplicity and intelligibility, we will focus on the system use cases from these three FCs, disregarding the previous connections. In other words, the figures in this section will only represent the sequence of messages between experimenters and these three core FCs, assuming that the intermediate elements would just forward the messages without adding any value to the use cases;
2. We have also separated the user authentication and access control (i.e. tasks pertaining to the security realm) from the user information plane (i.e. VE, IoT Resources/IoT Services and data). These steps will be utterly necessary for guaranteeing the minimum levels of security every time an experimenter wants to access the FIESTA-IoT federation (and a test-bed injects new pieces of information into the meta-cloud). Basically, any of the use cases described below will need a previous step for authentication and authorization of the requests described.

5.1.1 Security-based use-cases

5.1.1.1 Experimenter registration/ Identity Management

In order for an experimenter to use the FIESTA-IoT services and the FIESTA-IoT test-beds, they must be known to (and accountable by) FIESTA-IoT, i.e. FIESTA-IoT can authenticate the experimenter. In this use-case, we consider that FIESTA-IoT is the sole identity provider in the test-bed federation i.e. experimenters register with FIESTA-IoT and provide FIESTA-IoT with the credentials (username, password) that will be used for authentication.

The following is the expected behaviour for a new user registering with FIESTA-IoT (this use-case behaviour is illustrated in Figure 19):

1. The experimenter selects a sign-up link on the FIESTA-IoT portal web page (In the logical FIESTA-IoT architecture; the AuthN component exposes the identity management functionality via the web portal);
2. The experimenter fills in his/her information including e-mail address and password (security credentials);
3. Based upon the information, FIESTA-IoT decides whether to allow the experimenter to register. The experimenter will be sent an email at his/her registered email address for verification of his/her identity. When the user verifies via the link, registration is complete and the use case continues.
    a. If the experimenter does not verify the link sent to the e-mail address within a given time period then all information about the experimenter entered so far is removed from the FIESTA-IoT databases;
4. The experimenter information is stored in the Member Database (part of the User Management FG).

The experimenter is now free to authenticate (log-in) with FIESTA-IoT and use FIESTA-IoT services and test-beds. The other sub use-cases of the experimenter identity management are:

- The experimenter can update his/her account info at any time using the FIESTA-IoT portal;
- The experimenter can delete his/her account from the system at any time using the FIESTA-IoT portal;
- The FIESTA-IoT administrator can delete an experimenter’s account from the system at any time using the FIESTA-IoT administration portal.

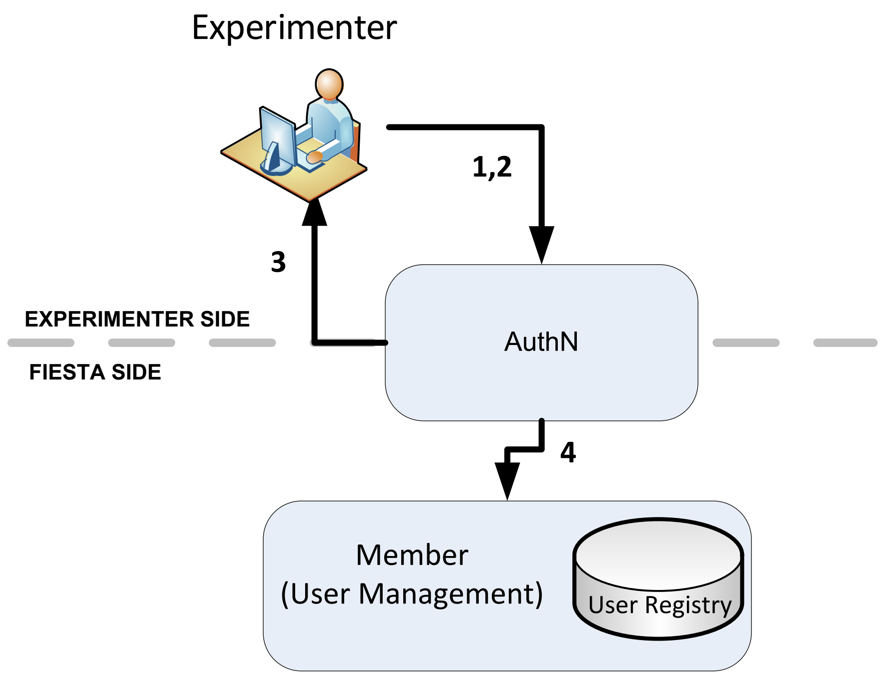


Figure 19: Experimenter Registration use-case

An extension of this use-case that will be considered in the final iteration of the architecture, is that of *federated identity management*. Experimenters who have a user account at a test-bed or 3rd party in the FIESTA-IoT federation can authenticate with FIESTA-IoT using their test-bed credentials. When the experimenter signs in to FIESTA-IoT, they are redirected to authenticate with their own identity provider (trusted by FIESTA-IoT)—once authenticated they are free to leverage the FIESTA-IoT services.

5.1.1.2 Resource Management

Each test-bed controls access to their resources using access control policies. Each incoming request to use test-bed resources is checked against these policies to determine if the request will be granted or denied. An example policy may be "An experimenter from organisation X can access the resource", or "An experimenter with write privileges may upload new resources to the test-bed".

In the first version of the architecture we consider two use-cases for controlling resources:

- Test-bed owners manage their existing resources at the test-bed level. Each test-bed is a set of static resources to be protected. Test-bed owners can add to these over time by registering new resources in the test-bed. Such resources will typically be data resources and IoT services to perform actions on IoT systems. The test-bed owners, when registering resources, will add control policies for these resources—this is illustrated in Figure 20. When a test-bed adds a resource as per described in the use-case "*Test-bed registers an IoT service/resource*" (described in Section 5.1.4.1) the service registry notifies the AuthZ component which then applies the test-bed's default access control policy to the newly registered resource (step 3). The test-bed administrator can directly alter both specific resource policies and default policies by interacting with the AuthZ component via the Access Policy Administration Tool (steps 4-7).

- Resource owners (e.g. users using crowdsourcing mobile applications) register their resource with a test-bed directly (i.e. dynamic resource registration), specify access control policies, revoke access at any time etc. Such resources are dynamic (as these living subjects come and go). Hence, in this case the owner specifies policies rather than the test-bed. For example, a resource owner may state that his/her private data uploaded to a test-bed may only be read by a subset of experimenters. In the simple case, the resource owner may grant control to the test-bed and this case reverts to the case above. Hence, the behavior is identical to that in 5.1.4.1 except that the stakeholder changes from the test-bed administrator to the experimenter who owns the resources. As before, the test-bed registers the resource to FIESTA-IoT when the experimenter dynamically adds it to the test-bed (steps 1-2), and the test-bed policy is applied (step 3). However, now the resource owner can update the access policy (steps 4-7).

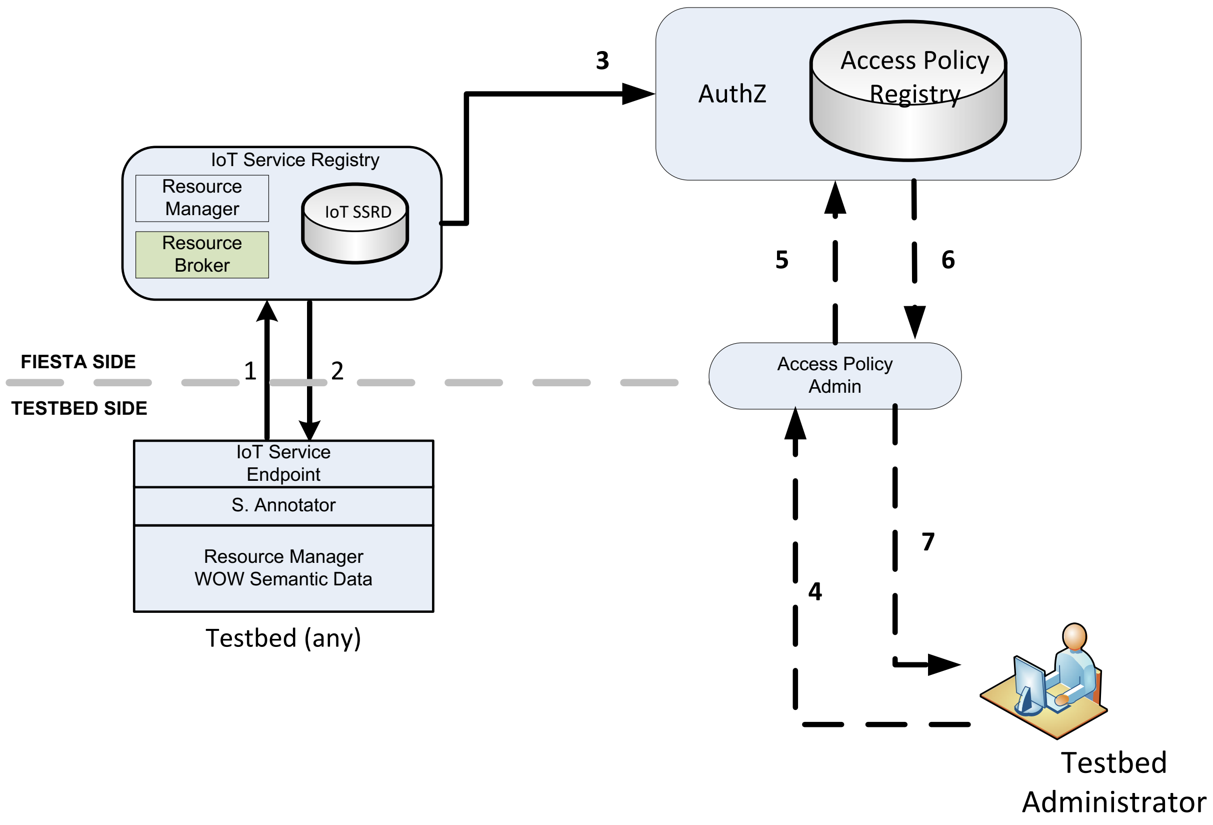


Figure 20: Test-bed administrator registers new access control policies

5.1.1.3 *Protected Resource Access*

Resources in the FIESTA-IoT federation are protected and require authorization in order to be accessed by only FIESTA-IoT experimenters. Such resources are:

- The FIESTA-IoT services e.g. IoT Service and Resource Browser, FIESTA-IoT experiment modeling;

- The test-bed resources e.g. access to FIESTA-IoT wrapped test-bed endpoints (VE, data, IoT Service).

Each request to use one of the above protected resources is checked in order to ensure that the request is from an authenticated experimenter, and that they are authorized to perform the request. Resource access in FIESTA-IoT follows a traditional Policy Enforcement Point (PEP) Pattern – requests are intercepted by the PEP (at FIESTA-IoT endpoints) and these are sent to a Policy Decision Point (PDP) component, which forms part of the logical AuthZ component). The PDP implements the grant/deny decision when evaluating the request against the access policy.

Resource Access is managed in two ways in FIESTA-IoT: fully managed by FIESTA-IoT or the test-bed manages access on their endpoints.

5.1.1.3.1 Fully managed by FIESTA-IoT

The test-bed trusts FIESTA-IoT to forward only authorized requests to the test-bed. A secure trusted channel from FIESTA-IoT to the test-bed API is managed by the KEM and TTP components. FIESTA checks that only authorized requests are forwarded to the test-bed APIs i.e. FIESTA-IoT's AuthN and AuthZ functions check the requested behaviour before the request is forwarded. The test-bed does not need to make any security decision concerning request from this connection.

This case is suited to lightweight test-beds with limited computational resources that do not need strong protection. This behaviour is illustrated in Figure 20 (where the security protection is applied to the "Experiment queries/retrieves Data (Class-I test-bed case) use-case" (described in Section 5.1.5.2) :

1. The experimenter signs on via the FIESTA-IoT AuthN service;
2. The experimenter receives an authenticated session to utilise FIESTA-IoT services and resources;
3. The experimenter accesses a meta-cloud data endpoint to retrieve data. This is a point in the architecture to make a policy decision. Hence, the request is intercepted before it is forwarded;
4. The request is forwarded to the AuthZ to compare the request and user information against the resource's policy;
5. If granted the use case proceeds otherwise an unauthorized access message is sent to the experimenter;
6. The request is then sent across the secure channel (HTTPS encrypted) established between the test-bed and FIESTA-IoT via the exchange of digital certificates;
7. The data value is retrieved;
8. The result of the request is sent to the experimenter.

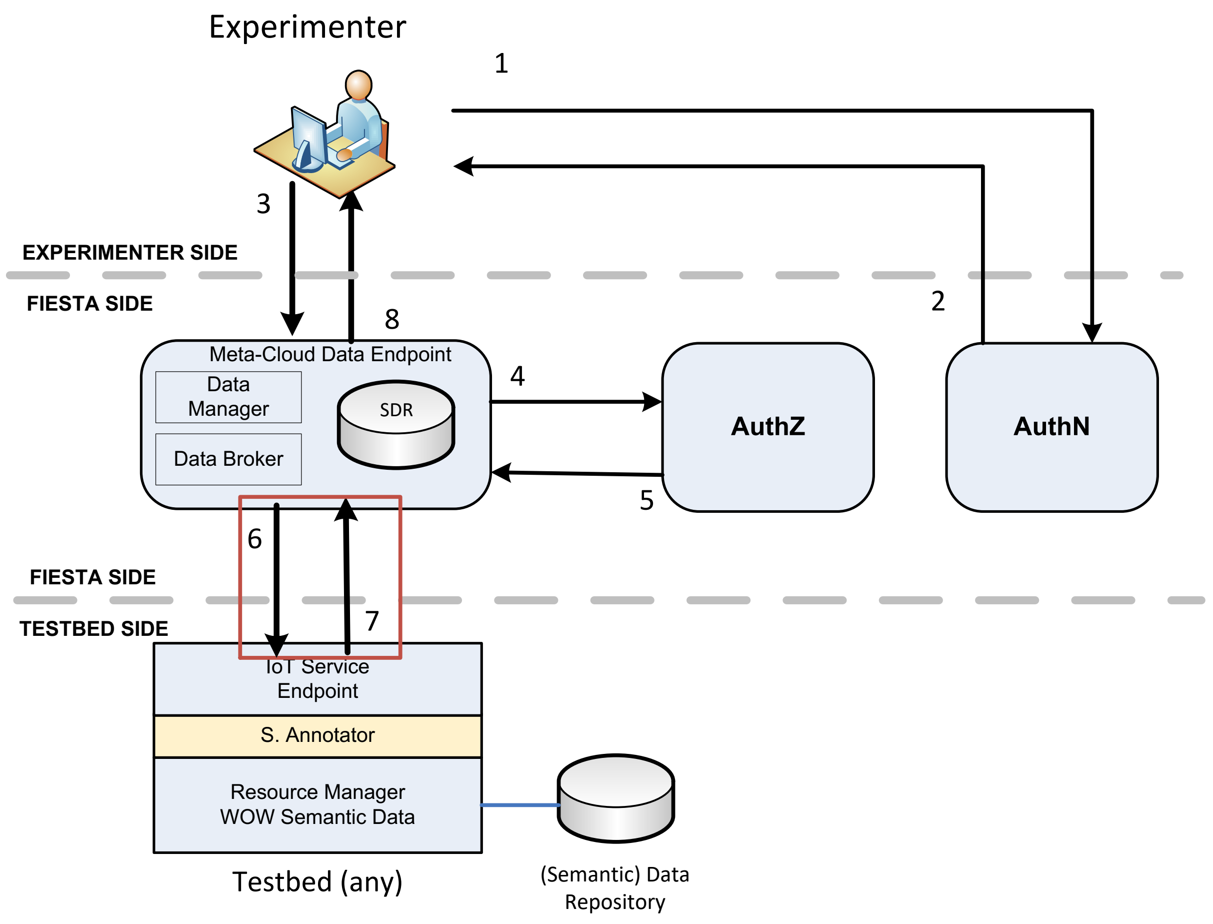


Figure 21: Secure Access to Protected Resource

5.1.1.3.2 The test-bed manages access on their endpoints.

Hence, the behaviour pattern is the same as described in Figure 21 but instead of the AuthZ component operating at the FIESTA-IoT side it operates at the test-bed (i.e. it does not need to trust FIESTA-IoT to execute access control). Hence, the PDP executes on site.

- The request is received at the PEP endpoint. Where the session is not already authorized: the authentication is checked with the FIESTA-IoT AuthN component, redirecting the request for the experimenter to input their credentials if needed;
- The authentication information is passed onto the PDP function and request is evaluated to be granted or denied.

This case is suited to test-beds who may want greater control over FIESTA-IoT access—for example, where the resources are shared beyond FIESTA-IoT and there is a need to ensure FIESTA-IoT does not starve other users of resources.

5.1.1.4 Test-bed becomes part of the FIESTA-IoT federation

A test-bed must perform a series of actions to ensure that it technically applies (i.e. can be used) within the FIESTA-IoT federation. Central to this, it must ensure that the test-bed follows the agreed security mechanisms of the federation. Hence, the following is the sequence of events that take places:

- The test-bed administrator registers with the federation and receives authentication credentials with test-bed admin rights that will allow him/her to access the FIESTA-IoT administration services including the security component of FIESTA-IoT e.g. the AuthN, AuthZ and KEM components. This action is performed by an initial online web form request followed by an e-mail exchange between the FIESTA-IoT admin and the test-bed admin to establish the authenticity of the approach;
- The test-bed admin signs in to the FIESTA-IoT administration services using the credentials and then requests key exchange, i.e. it receives FIESTA-IoT's public key, and uploads its own public key as a registered test-bed;
- The test-bed registers its resource and service interfaces (exposed with HTTPS endpoints). FIESTA-IoT connects to these endpoints and tests that a secure and trusted communication channel is established between the two parties;
- When registering the resources in the prior step, the test-bed admin also signs in to the AuthZ component and selects access policies to be applied; where necessary creating new access policies.

5.1.2 Process Management use-cases

5.1.2.1 An experimenter creates a new experiment

In order to interact with the FIESTA-IoT platform, an experimenter should first define what they want to do and then define/create the experiment. FIESTA-IoT platform provides two possibilities for an experimenter to define/create the experiment. These include:

- Creating an experiment using Experiment editor: When an experiment choses Experiment Editor to defined an experiment, an experimenter basically creates an instantiation of the DSL. As within this DSL there are many entities, an experimenter can manually fill in these entries using a user-friendly web interface. However, if the experimenter does not want to manually fill in these entries, an experimenter can create the DSL instance using their favourite XML editor and upload the instance. It is noteworthy to add that an experimenter has to first login to FIESTA-IoT platform to access these features.
- Directly sending queries to FIESTA-IoT Meta-Cloud: If an experimenter chooses not create the DSL instance but rather directly connect to FIESTA-IoT Meta-Cloud, they can also do so. However, in this case they will have to create an execution engine like system and execute it on their own side. This execution engine like system would basically send periodic request to FIESTA-IoT Meta-Cloud for the data. In this case also, It is noteworthy to add that an experimenter has to first login to FIESTA-IoT platform to access the FIESTA-IoT Meta-Cloud.

For both the above cases it is essential for an experimenter to follow best practices defined by the FIESTA-IoT platform both for the SPARQL queries and for the instance of the DSL `(FIESTA-IoT D5.4)`. The above two cases are illustrated using Figure 22 where the interactions are as follows:

1. The experimenter signs on via the FIESTA-IoT AuthN service.
2. The experimenter receives an authenticated session to utilise FIESTA-IoT services and resources.
3. The experimenter accesses the desired service (Experiment Editor (Figure 22a) or Meta-Cloud endpoint (Figure 22b)). This is a point where the request is intercepted by the authorization service before it is forwarded to the desired service.
4. The request is forwarded to the AuthZ to compare the request and user information against the policies;
5. If requested is granted the experiment editor or the Meta-Cloud endpoint is made accessible to the experimenter where the experimenter can create the new experiment. If the request fails an unauthorized access message is sent to the experimenter;
6. When the request is granted, as mentioned in step 5, the experiment editor interface or the Meta-Cloud endpoint is made available to the experimenter. This request is sent across the secure channel (HTTPS encrypted) established between the Experimenter and FIESTA-IoT via the exchange of digital certificates;
7. The Experimenter creates experiment either using Experiment editor or Meta-Cloud. The Experiment editor then saves the created experiment in the ERM repository for experiment execution while as Meta-Cloud endpoint has no such capabilities the experimenter needs to call the Meta-Cloud endpoint every time they have to execute an experiment.

Figure 22: (a) Creating a new Experiment using Editor, (b) Creating a new Experiment using Editor

5.1.2.2 An experimenter executes an existing experiment

Once the experiment is defined using either of the two methods, execution of the experiment is next. For the case when experiment is created using Experiment Editor, the experimenter has to perform certain steps before the Execution Engine of FIESTA-IoT platform starts executing the defined DSL instance. The experimenter has to access management console, chose the experiment, then chose the service they want to start the execution for. After choosing the service, the experimenter manually starts the service using the management console UI. An experimenter can perform various execution related tasks like pause and delete. An experimenter other than executing self-defined experiment services can execute experiment services that are defined by other experimenters and are discoverable. In such a case, the management console creates an instance of the discoverable service and associates them to the experimenter. The experimenter then follows the same process as defined above to start the execution of the service. The results of the experiment execution in both cases are sent to experimenters on the URL defined by them. In case the URL is not available due to some reasons, the results are store internally in the ERS. The experimenter then has to access the ERS in order to download the results.

This is illustrated using Figure 23 where the interactions are as follows (we assume that the experiment is already created using the experiment editor and that all interactions are authorized):

1. The experimenter signs on via the FIESTA-IoT AuthN service.
2. The experimenter receives an authenticated session to utilise FIESTA-IoT services and resources.
3. The experimenter then opens the EMC.
4. Internally the EMC requests the experiment details (metadata) from the ERM.
5. The ERM sends the response from ERM and EMC displays the content to the experimenter.
6. The experimenter then selects the experiment service to enables the execution
7. EMC then calls the EEE API to start the execution of the service.
8. The EEE requests ERM for details of the service, for example schedule and the query
9. The ERM responds with the needed attributes.
10. After receiving the response, the EEE sets the schedule, notifies the EMC for the service status and sends the query for execution to the Meta-Cloud endpoint. In case the schedule needs execution at a fixed interval the EEE sends the query to the Meta-Cloud at the needed interval. It is here the experimenter can also change the status of the services by manually pausing the service. In such a case EMC send a pause request to EEE.
11. IoT-Registry executes the query and sends the response back to the EEE.
12. The EEE then forwards the response to the Experiment Data receiver component that executes on the experimenter side. In case the Experiment Data receiver is not available the results are stored in the ERS.
13. The Experimenter requests ERS for the desired results.
14. Upon the receipt of the request the ERS send back to the experimenter the stored results and deletes them from the repository.

Figure 23: Experiment Execution

In the case when the experimenter directly accesses the FIESTA-IoT Meta-Cloud, they have to define the execution schedule and the process. As this is not controlled by the FIESTA-IoT, it is out of scope of this deliverable. Nevertheless, it is worth mentioning that when the queries are directly sent to the FIESTA-IoT Meta-Cloud, results are directly sent to the experimenter and are not stored within FIESTA-IoT platform.

Note that the above cases are when the experimenters schedule the experiment for getting the data using periodic execution.

Other than the above case, experimenters also have a possibility for real-time access of data. In such a case, experimenters first have to know the FIESTA-IoT specific sensor ID for which they want real time data. This can be known after executing a specific query only once.

After the ID is known, the experimenter can use a dedicated API within FIESTA-IoT platform and the ID to get the most recent/real-time observation from the resource. This case is illustrated using Figure 24 where the interactions are as follows:

1. The experimenter signs on via the FIESTA-IoT AuthN service.
2. The experimenter receives an authenticated session to utilise FIESTA-IoT services and resources.
3. The experimenter accesses a Meta-Cloud data endpoint and asks for the list of sensors that produce real time data.
4. The request is forwarded to the AuthZ to compare the request and user information against the resource’s policy.
5. If the request is granted, Meta-Cloud data endpoint executes the needed query. If the request fails an unauthorized access message is sent to the experimenter.
6. After execution of the query, a list of sensor IDs producing real time data is sent back to the experimenter
7. The experimenter chooses the sensor ID they want to get real time data for and sends a request to Meta-cloud endpoint to process the request.
8. Step 4 is repeated.
9. Step 5 is repeated
10. Upon a success, the Meta-cloud maps the received ID to the original sensor ID from the testbed and makes a call to the endpoint of the sensor.
11. The real time data is forward to Meta-Cloud. The Meta-Cloud then converts the response into FIESTA-IoT ontology compliant RDF and hashes back all the IRIs
12. The Meta-Cloud sends the RDF to the experimenter.

Figure 24: Experimenters accessing real time data

5.1.2.3 An experimenter accesses an experiment result

Once the experiment service is executed and results are available the Execution Engine sends the results to the experimenter at the defined URL in the instance of the DSL. If the URL is not reachable then the Execution engine stores the results so that experimenters can download the results later on. For an experimenter, to receive the results, they have to create an API that execution engine can uses to send/upload the results. Once the results are sent to the experimenter, they are owned by the experimenter can be accessed freely by them. It is noteworthy to mention again that an experimenter has to first login to FIESTA-IoT platform to access these results. This case is illustrated using Figure 25 where the interactions are as follows:

1. If the experiment results are sent via experiment data receiver then there are no interactions with the FIESTA-IoT platform. Experiment data receiver is local to experimenters. The steps 1' and 1'' represent local interactions. Nevertheless, if the results are stored in ERS then the experimenter needs to sign on via the FIESTA-IoT AuthN service.
2. The experimenter receives an authenticated session to utilise FIESTA-IoT ERS services.
3. The experimenter calls the ERS services.
4. Internally, ERS Authorization module to check the requests to identify if the experiment has authorization to access the ERS services.
5. The ERM sends the response from ERM and EMC displays the content to the experimenter.
6. If requested is granted, the ERS process the request. If the request fails an unauthorized access message is sent to the experimenter.
7. The ERS sends the response with the stored data and deletes the sent data from the store.

Figure 25: Accessing Experiment results

If an experimenter directly accesses the FIESTA-IoT Meta-Cloud, then the experimenter can access the results via response of the HTTP call to the FIESTA-IoT Meta-Cloud.

5.1.3 Management use-cases

5.1.3.1 An experimenter manages an existing experiment

Experimenters can manage their experiment that they have created. For experiments that are created using Experiment Editor an experimenter can manage the experiment using experiment management console and experiment editor. Using Experiment management console, the experimenter can schedule (start), pause, delete, poll, view execution history, or add discoverable services to the experiment. On the other hand, using experiment editor an experimenter can create/update an instance of the DSL. This case is illustrated using Figure 26 where the interactions are as follows (we assume that the experiment is already created using the experiment editor and that all interactions are authorized):

1. The experimenter signs on via the FIESTA-IoT AuthN service.
2. The experimenter receives an authenticated session to utilise FIESTA-IoT services and resources.
3. There are two ways to manage an experiment. One is via changing the experiment DSL using experiment editor and another one is using EMC to manage execution of the experiment (step 3').
4. When experimenter choses to manage the experiment by modifying the DSL via Experiment Editor, the experiment editor check with EEE for any executing

service and notifies EEE if there is any change so that EEE can update the execution of the services in the experiment. Nevertheless, if the experimenter chooses EMC, he can only change the status of services or delete the execution of a service. If this is the case then EMC notifies EEE to update the status of the service and take necessary actions.

5. Either way, EEE performs the necessary action and notifies either Experimetn Editor or EMC (step 5').
6. The experiment editor or the EMC (step 6') then notifies the experimenter about the changes.

Figure 26: Managing an experiment

5.1.4 Resource/IOT Services oriented use-cases

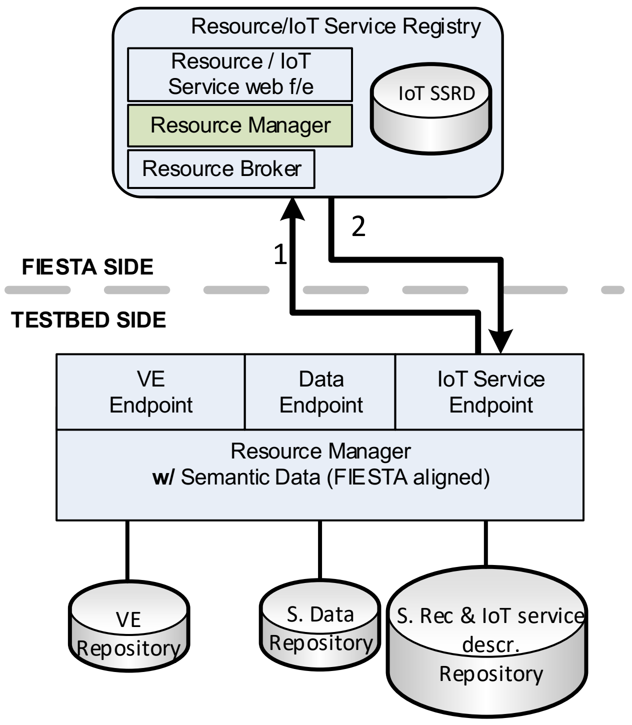
In this section, we present a number of essential system use-cases related to the use of IoT Services and resources. We start with the registration process by the test-beds part of FIESTA-IoT federation, then follow with look-up/discovery and reservation process (by FIESTA-IoT users) and end-up finally with the actual use of those IoT Service/Resources by FIESTA-IoT users.

5.1.4.1 Test-bed registers an IoT service/resource

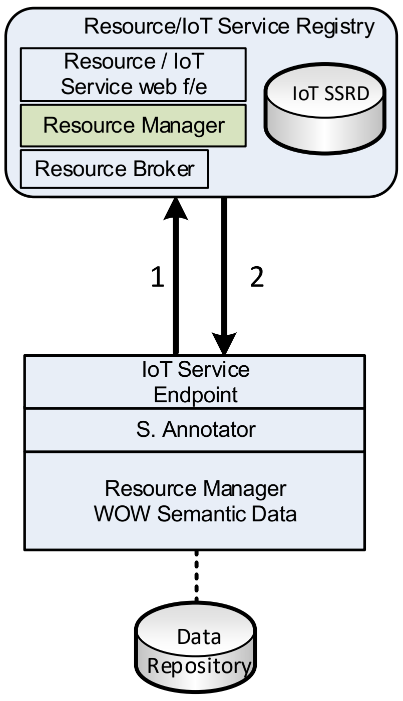
It is worth highlighting that every Resource/IoT service must be associated with a semantic description aligned with FIESTA-IoT's ontologies.

On the first hand, resources/IoT Services pertaining to Class-I test-beds are stored locally at the test-bed but the corresponding IoT Service/Resource endpoint needs to be registered with the FIESTA-IoT IoT Service Registry Resource Broker FC, indicating that subsequent resource searches will be brokered by the IoT Semantic Service & Resource Descriptions (SSRD) towards the Class-I test-bed Resource Manager.

Moreover, semantic descriptions of Resources/IoT Services pertaining to Class-II & -III test-beds need to be stored directly at the FIESTA-IoT side within the IoT SSRD Repository, thus replicating the info and fulfilling the semantic annotations that will define the FIESTA-IoT ontologies proposed in WP3.


Resource/IoT Service Registry
Resource / IoT Service web f/e
Resource Manager
Resource Broker
IoT SSRD
1
2
FIESTA SIDE
TESTBED SIDE
VE Endpoint
Data Endpoint
IoT Service Endpoint
Resource Manager w/ Semantic Data (FIESTA aligned)
VE Repository
S. Data Repository
S. Rec & IoT service descr. Repository

a) Class-I: Resource registration (only registers test-beds' Resource Manager endpoint)


Resource/IoT Service Registry
Resource / IoT Service web f/e
Resource Manager
Resource Broker
IoT SSRD
1
2
IoT Service Endpoint
S. Annotator
Resource Manager WOW Semantic Data
Data Repository

b) Classes-II & -III: Resource registration (full description of resources/IoT services)

Figure 27: Resource/Service registration sequence diagram

To illustrate this, Figure 27 describes these 2 alternatives, whose sequence of messages is described as follows:

1. This first message is actually a Resource/IoT Service Registration request, and will be addressed to the Resource Manager allocated in the Resource/IoT Service Registry FC. Two possibilities arise here: as for Class-I test-beds (Figure 27a), they will store their information locally (at test-bed level), hence they will become an extension of the FIESTA-IoT meta-cloud (working as a distributed system). Thus, they will only register the IoT Service Endpoints into the FIESTA-IoT SSRD. On the other hand, Class-II & -III test-beds (Figure 27b), by default, do not comply with the FIESTA-IoT's ontologies, will have to semantically annotate their resources/IoT services descriptions and replicate them into the FIESTA-IoT level. This way, it is deemed necessary to include a Semantic Annotator entity at the test-bed level. Then, the message will be headed towards the FIESTA-IoT Resource Manager, which will be in charge of their storage into the FIESTA-IoT SSRD;
2. Once the registration is performed, the IoT Service Registry FC sends back an acknowledgment to the test-beds, confirming that everything has gone ok or informing about any potential fault.

5.1.4.2 Experiment makes reservation of resource(s) and request asynchronous publishing of data

The reservation process takes place after a prior test-bed-agnostic resource discovery and selection at the level of the FIESTA-IoT IoT Service/Resource registry (Resource manager sub-component).

Through this reservation process the experiments specifies an expected publishing policy from the test-bed responsible for the selected resources as follows:

- **Start time:** When the publishing of readings should be started by the Testbed ultimately responsible for the resource;
- **Sampling rate:** at which rate the data should be accessed by the resource and published towards the experiment;
- **Routing information:** which is the filtering/routing criteria to be used;
- **Stop time:** When the publishing of Data should be ending.

Test-beds' Resource Managers are responsible of implementing the publishing policies and check if they would conflict with other already requested policies. If for some reason the policy cannot be implemented, a negative answer would be returned back to the experimenter.

Where the reservation is successful an access control policy is additionally put in place (created and registered with the AuthZ component for the duration of the reservation). This access policy can then ensure authorized access to the reserved resources is achieved i.e. it is access by the experimenter requesting the reservation. For example, Access Control policies could ensure sole access for an experimenter or group of experimenters in the reservation period.

They are also responsible for accessing the resources according to the before-mentioned publishing policy and for publishing readings (semantically annotated) to the Message Bus using the Routing criteria specified at the time of the resource reservation (see Section 5.1.4.3).

This use case does not explicitly use IoT Services as the experiment delegates to FIESTA-IoT (and ultimately to the test-bed) the publishing of data it is interested in an asynchronous way.

To illustrate how FIESTA-IoT will deal with this reservation process, Figure 28 represents the sequence of messages exchanged between the various elements. Namely, the content and meaning of each one is briefly depicted below.

1. The experimenter identifies his/her requirements (described above) and send a reservation requested addressed to the test-bed's Resource Manager, where the decision about granting/not granting will be made. In the eyes of the test-bed, it will be FIESTA-IoT who is actually requesting the access to the resources, not an end-user;
2. Assuming that the test-bed is able to support such capacity reservation, an ACK message will be sent back. Otherwise, a negative acknowledgement would be addressed to the experimenter, thus voiding the following steps;

3. If the step above is successful, as commented above, a new access control policy is registered onto the AuthZ FC, binding the experimenter ID and the set of resources/VE reserved;
4. Finally, a confirmation reaches the experimenter, who can start subscribing to the already acknowledged resources/IoT Services.

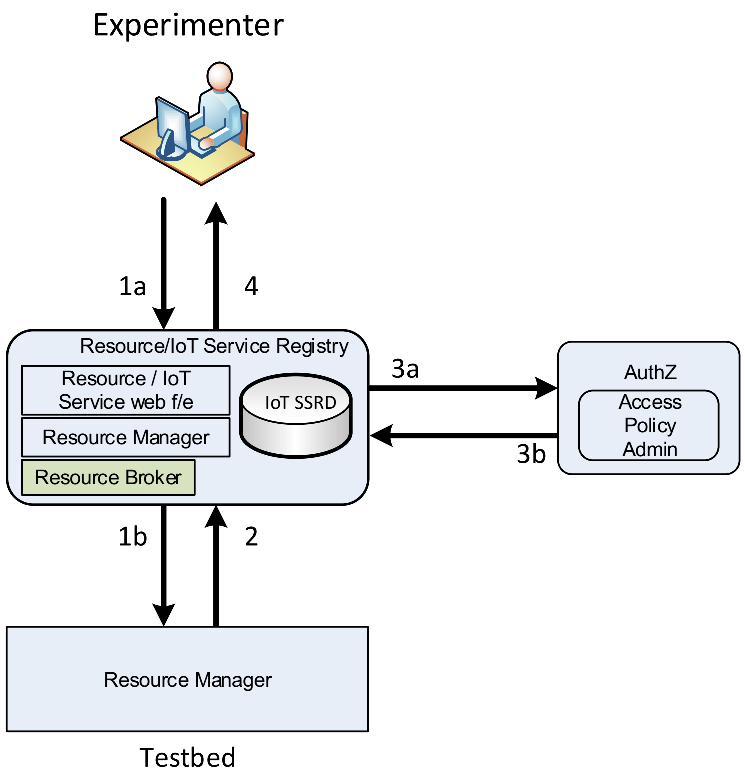


Figure 28: Resource reservation sequence diagram

5.1.4.3 Experiment subscribes to asynchronously pushed data streams

This system use-case contemplates the use of an asynchronous publish-subscribe (a.k.a. Pub/Sub) service, where experimenters subscribe to those resources/IoT services in which they are interested in. Once this subscription is acknowledged, each time the corresponding resources[8] (asynchronously) generate an observation/measurement, the FIESTA-IoT architecture is in charge of delivering the information to the experimenters (and all of them who are subscribed to the determined service). Moreover, this use-case is tightly linked to the reservation of resources (Section 5.1.4.2) , since experimenters will need to reserve/notify their subscriptions prior to start receiving data, having to specify as well the time interval during which they are going to be listening to the FIESTA-IoT Message Bus, whose role will be essential in this type of asynchronous service.

Then, if both reservation and subscription processes have been successfully acknowledged, experimenters will be aware of any future event arisen in any of their

[8] The reader might take into account that these subscription policies might be bounded to VEs as well, where an experimenter subscribes to e.g. all the new illuminance observations generated throughout a street "X" in the city of Santander.

subscribed services. Figure 29 represents the sequence diagram that is prior to start receiving asynchronous message notifications, whose steps are explained below.

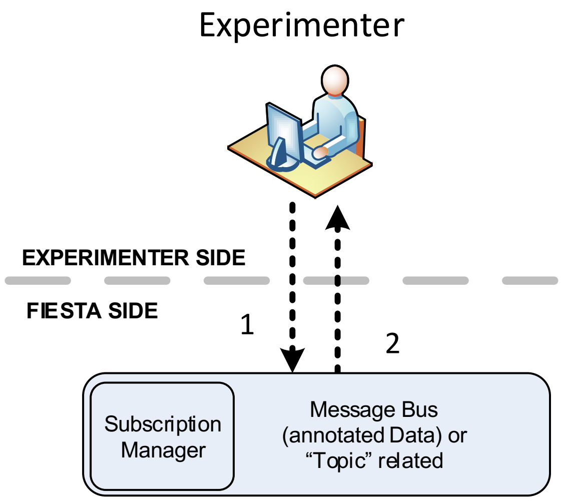


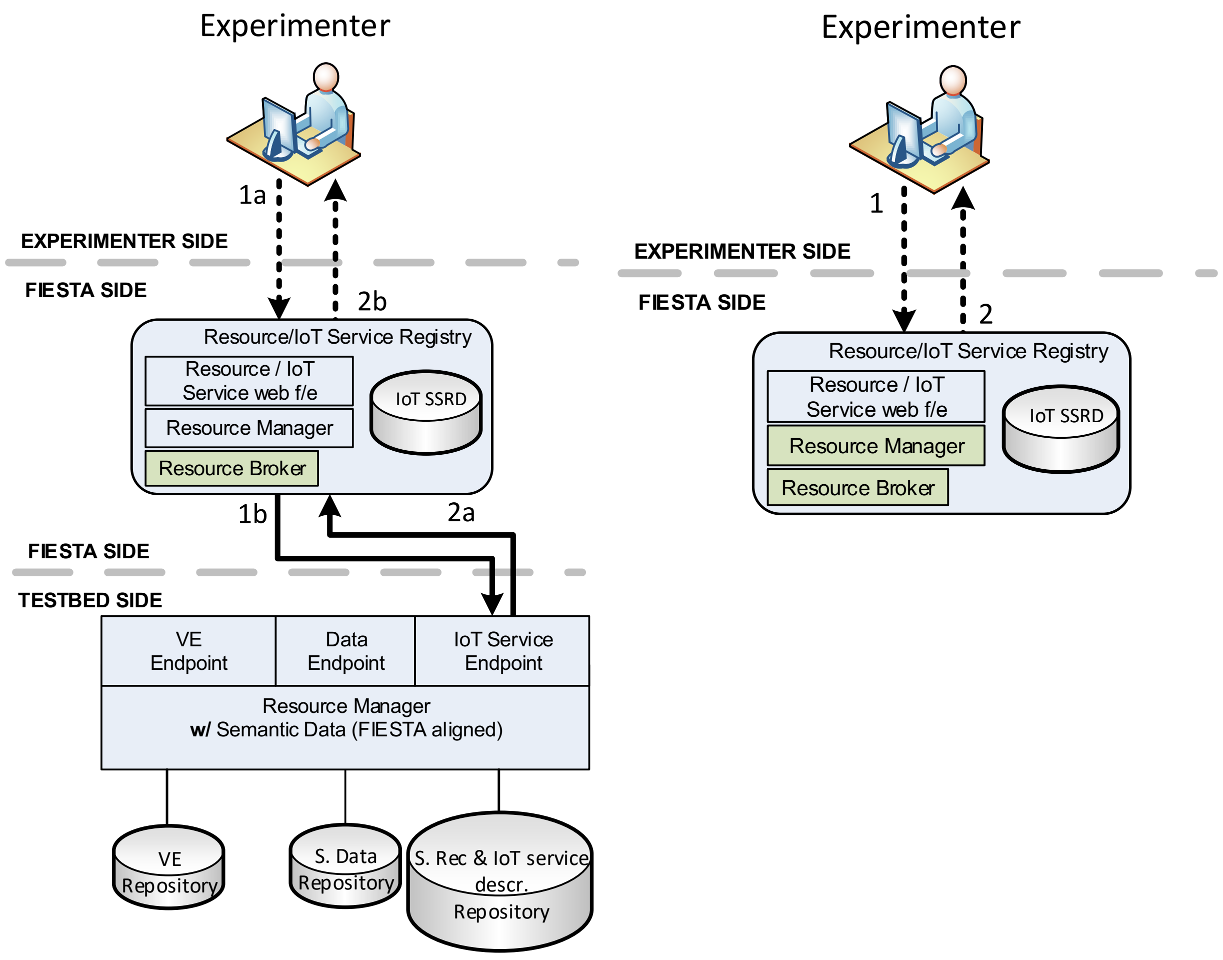
Figure 29: Experimenter subscription to data sequence diagram

1. The experimenters send a subscription request towards (directly or not) the Meta-Cloud Data Endpoint. In order to match the Experimenter and the IoT Services that he/she is subscribing to, it is deemed necessary the usage of a topic-based system, through which the Subscription Manager will be able to pair, upon the arrival of information belonging to a particular resource, the subscriber(s) to which the message must be forwarded;
2. The experimenter receives an ACK from FIESTA-IoT, containing a "Subscription ID", used by experimenters to listening to the Message Bus in the time negotiated during the Reservation stage.

5.1.4.4 Experiment looks up resources/IoT Services (Discovery[9])

Probably, the first step an experimenter might take is to search the resources/services available in the FIESTA-IoT meta-test-bed. Assuming that this concrete search is resource and not VE-oriented, they will (either through the Experiment Interpreter, which will likely contact the FIESTA-IoT Web Browsing & Configuration FC or directly querying the IoT Service & Resource Registry) finally reach the IoT Service/Resource Registry. Figure 30 illustrates the sequence of messages that will be involved into the resource/service look-up, whose individual description is briefly resumed below.

[9] This use-case assumes prior description and registration of IoT Services and Resources



a) Class-I: Resources and services are remotely managed by test-beds themselves

b) Class-II & -III: Resource/Service description are stored at the FIESTA-IoT SSRD

Figure 30: Resources/IoT services discovery sequence diagram

In the same way as for the previous case, there are two options for the resource/IoT Service discovery (recall that end-users can not only retrieve all the available resources, but also generate more complex queries selecting e.g. based on location, phenomena, time intervals, etc.). As an example, an experimenter might want to retrieve all the resources/IoT services with the capability for measuring temperature in the area of e.g. a rectangle defined from its North/West and South/East corner coordinates. The response to this query will include the IoT descriptions, including the address of the endpoint able to invoke the IoT Service.

On the left side (**Figure 30**a), Class-I test-bed follows a top-down approach where the FIESTA-IoT architecture plays the role of an intermediary between the experimenter and the “fully compatible test-bed”, thus brokering and forwarding the messages between the endpoints. Moreover, the right side on the picture (**Figure 30**b), represents the situations (i.e. Class-II & -III test-beds) where all the descriptions are replicated and stored into the FIESTA-IoT SSRD. In this latter case, the look-up operation will be typically supported by the combination of the Resource Broker (recall that it is the interface upwards the end-users) and the Resource Manager, which will be in charge of interacting with the IoT SSRD and generating (if there is any) the response to the query. The explicit sequence of message is described below:

1. The experimenter generates a query (i.e. SPARQL) that aims at retrieving a list of resources/IoT Services that comply with the requirements that shape such request. It is addressed to the Resource Broker, who, depending on the type of underlying platform, will either forward it to the corresponding test-bed's Resource Manager through its IoT Service Endpoint (for Class-I test-beds, which will lead to the two-fold 1a and 1b messages) or the Resource Manager (for Class-II & -III test-beds);
2. From the results of the query execution in the corresponding repositories, that is, either test-bed's SRD (Class-I) or FIESTA-IoT SSRD (Class-II & -III), a response is generated and sent back to the experimenter. It is worth recalling that, for the latter case, the semantic annotator has to tailor the resource/IoT Service descriptions defined in the FIESTA-IoT's ontologies (which will be addressed in WP3).

5.1.4.5 Experiment invokes IoT Services

From the results gathered in the previous use case (i.e. resource discovery), an experimenter has now the information he/she needs to start retrieving data from the resources that are exposed by their IoT Services. Thus, assuming that the IoT Service endpoints are known, experimenters just need to invoke these services through these addresses and wait until the data is received, following the sequence diagrams described in Figure 31, which differentiates between the two different ways that we have been observing throughout this section.

1. For Class-I test-beds (Figure 31a), the IoT Service Endpoint does deal with the incoming request locally at test-bed level and sent back the observation with all the metadata inside. Otherwise, for Class-II & -III test-beds (Figure 31b) the observation is retrieved from the underlying resources but, unlike for the previous case, the format of the measurements needs to be then translated to the FIESTA-IoT semantic format, using the Semantic Annotator. As can be appreciated, the FIESTA-IoT architecture acts as an intermediary between experimenters and test-beds so, after receiving the message from the end-user (message 1a), it just relies on the Broker to forward the information downwards to the corresponding test-bed's IoT Service (message 1b).
2. Finally, the Resource Broker sends back the response to the experimenter (message 2a), passing across the FIESTA-IoT meta-architecture (message 2b).

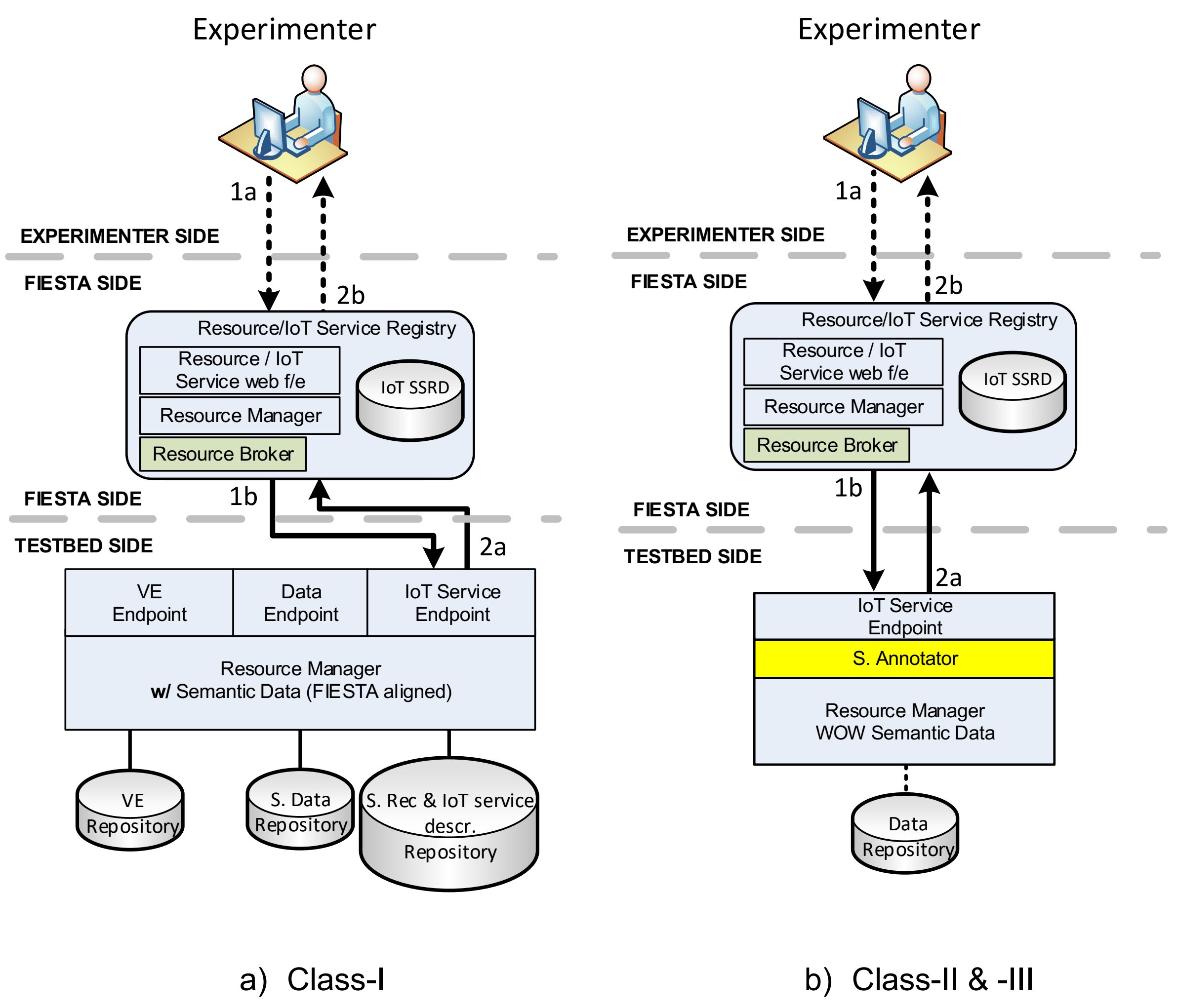


Figure 31: IoT Service invocation use case sequence diagram

5.1.4.6 Experiment creates and describes and then stores a new set of rules within the Rule repository

An experiment can define a set of rules and describe it semantically so that it can be shared and discovered by other experiments. The three steps involved are 1/ to edit the rules (not supported by FIESTA-IoT) 2/ to describe the set of rule following a FIESTA-IoT dedicated ontology (editing is not supported) and 3/ to store the set of rules along with its description to the FIESTA-IoT Rules repository.

5.1.4.7 Experimenter discovers an existing set of rules from the Rule repository

5.1.4.8 Experiment invokes the FIESTA-IoT Reasoner FC

This system use-case illustrates the way an experiment can use the reasoner in order to infer new data from existing and pre-stored data. The work achieved by the reasoner is articulated around three main steps which are 1/ to retrieve a bunch of data from FIESTA (either it is stored at Raw-data producers class – I or centrally inside the Meta-cloud Data Repository), 2/ to load a set of rules (based on the invoker's instructions) within the reasoner, 3/ to run the set of rules upon the loaded data and finally 4/ to deal with the inferred new data.

Information passed on to the reasoner by the experiment (thru the Reasoner API) includes (some maybe ignored): topics along which the inferred data is produced [optional], specification of the data to be retrieved by the reasoner (in the form of a SPARQL sentence), a reference to a set of rules (either discovered from the rule repository or previously created and then stored to that repository). In the special case the Reasoner is invoked by a Knowledge-producer, newly created knowledge would be stored within the central data repository so that any FIESTA-IoT user can make use of it (for e.g. running high-end[10] experiment).

5.1.4.9 Experiment invokes the FIESTA-IoT Analytics FC

The experimenter can interact with the Analytics functional component via the following steps:

1. The experimenter (1a) creates a standalone experiment and first selects a list of techniques along with the relevant parameters, followed by the relevant data sources that they seek to be analysed. Next (1b) an HTTP request to the Analytics service is constructed by posting (in JSON format) the list techniques, parameters and SPARQL query of the data. After receiving the HTTP request, the Analytics tool will query the semantic data repository in order to retrieve the data, followed by the processing of the data given the methods the user has selected.
2. After processing the data set the Analytics tool will either, (2a) push the data set back the experimenter as a CSV file, or will request (2b) the semantic re-annotation of the processed data and thus storage in the semantic data repository via the message bus.

5.1.5 Data oriented use-cases

In this section we describe few data-oriented use-cases dealing first with the publishing of data by Raw-data Producers and then different ways of querying data from the FIESTA-IoT user's perspective.

5.1.5.1 Test-bed publishes semantically enhanced Data to the FIESTA-IoT Platform

As has been stated a number of times throughout this deliverable, Class-II & -III test-beds do not store semantically annotated data locally (in triple-stores for instance). As a consequence, they do not provide a local Data Endpoint, hence they will need to publish semantically enhanced data to the central repository so that it can be accessed by any third-parties like experimenters, Knowledge Producers or (added-value) Service Providers. To achieve this, the Message Bus will play an essential role, acting as the intermediate entity between test-bed and both FIESTA-IoT's Meta-Cloud Data Endpoint and experimenters, as hinted in Figure 32.

[10] By High-end we mean here experiments exploiting knowledge instead of annotated raw-data only.

Resource Managers need therefore to feed this Message Bus according to the publishing policies they are implementing, either their own or the one requested from experimenters during the Resource Reservation process described in Section 5.1.4.2 (if supported). In order to comply with the semantic descriptions addressed in FIESTA-IoT, the observations must be translated by the Semantic Annotator prior to send them through the IoT Service Endpoint.

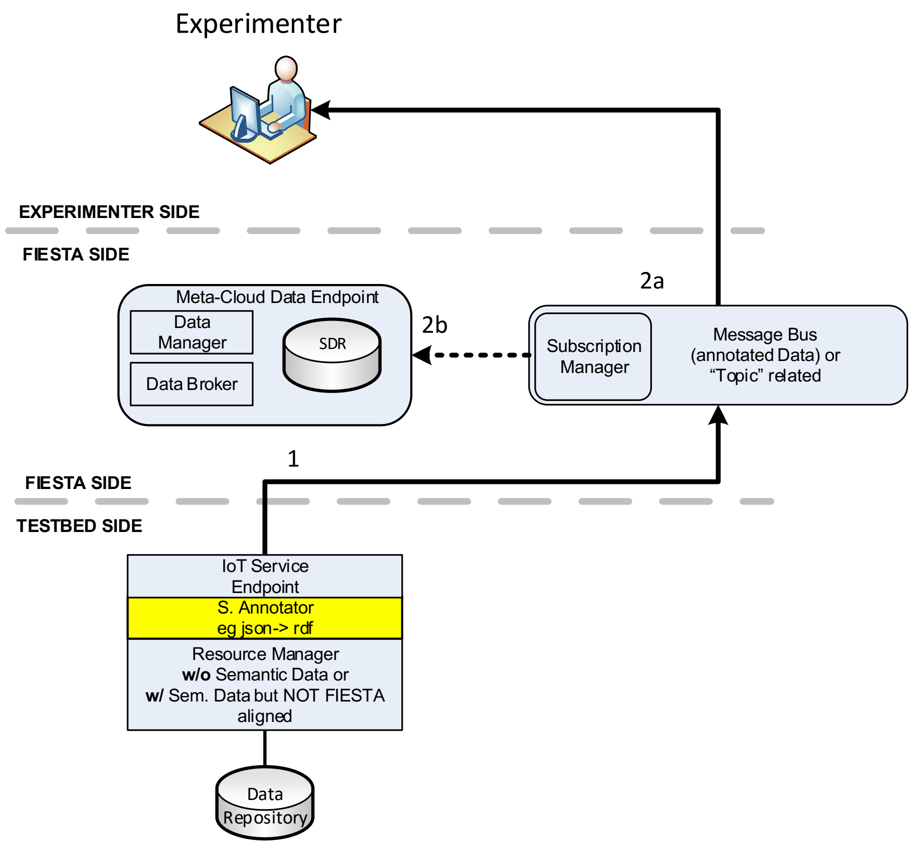


Figure 32: Data publication through the Message Bus sequence diagram

As a result:

- Class-II test-beds replicate data: locally it is not semantically annotated and stored, possibly in an either proprietary/standardised format e.g. JSON. Such test-beds are still able to provide access to information via IoT services. When answering IoT Service requests, they would still need to annotate the data before sending responses to the FIESTA-IoT Resource Broker;
- Class-III test-beds are most likely unable to provide IoT Service exploiting historical data. However, they will be only able to support IoT Services based on the delivery of the last/current values. Hence, it will be up to the test-beds the responsibility of publishing/advertising the IoT Services they can serve.

All in all, the sequence of messages followed in this use case is the one observed in Figure 32:

1. Every time a physical resource generates an (asynchronous) observation/measurement, a test-bed Resource Manager sends a (annotated) message towards FIESTA-IoT's Message Bus;

2. Once the Message Bus gets the information, it sends a copy to those subscribers that are registered to the concrete topic (or topics) to which the event (e.g. VE/phenomena/location) belongs to. As can be appreciated, either the Meta-Cloud Data Endpoint (message 2a) or experimenters (message 2b) might be the final destinations of the message.

5.1.5.2 Experiment queries/retrieves Data (Class-I test-beds)

Since Class-I test-beds can be seen as a sort of extension of the FIESTA-IoT platform, their joint operation will work like a distributed storage system. Unlike Class-II & -III test-beds, all the essential information will be saved locally, at test-bed level. All datasets (i.e. VEs, Data and Resources/IoT Services) will be fully compliant with the ontologies and annotation formats approved in the scope of the FIESTA-IoT project. Hence, the FIESTA-IoT architecture will play the role of a broker, abstracting the underlying stuff to end-users and forwarding the queries/responses between the real endpoints (Experimenters ↔ test-beds). With regards to the data acquisition over this “ideal” class of test-bed, Class-I platforms will provide their own Data Endpoint through which SPARQL (should we decide using SPARQL) queries can be sent by experimenters in order to retrieve the information directly from the source.

All in all, Figure 33 shows the sequence that will be followed to collect the data by the different test-bed categories. We list below the steps/messages that are to be generated by the different entities.

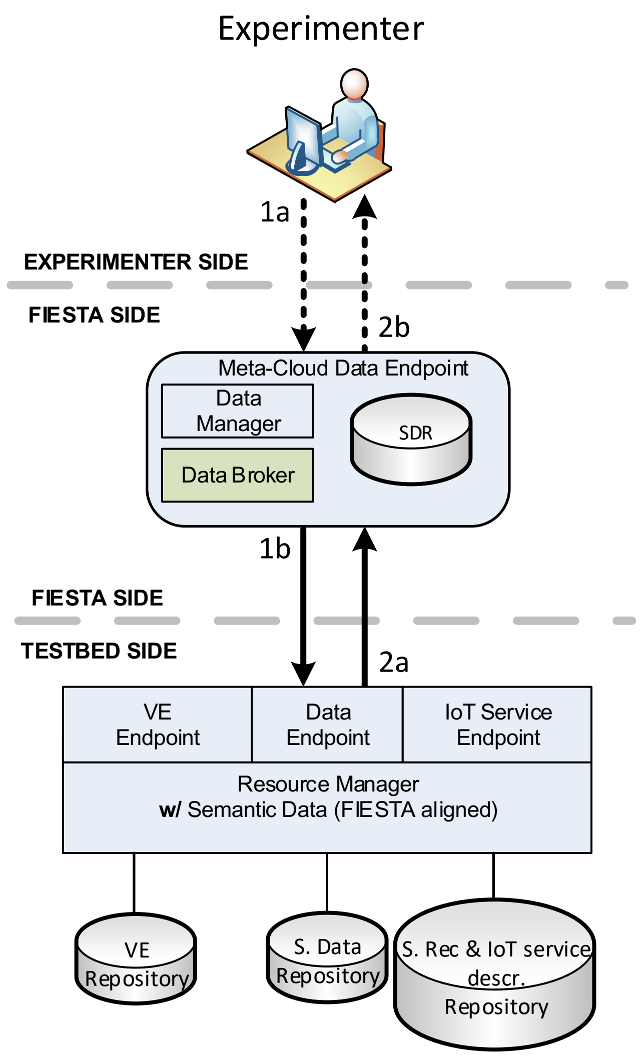


Figure 33: Data collection sequence diagram (Class-I test-bed)

1. A data query (i.e. typically, SPARQL-based) is sent by the experimenter (it is worth recalling that there are various entities that might be involved in the communication between the experimenter and the FIESTA-IoT Meta-Cloud Data Endpoint) towards the Meta-Cloud Data Endpoint's Resource Broker, corresponding to message 1a. Once it has detected that the query is addressed to a Class-I test-bed, the own Broker will just forward the message to the test-bed's Data Endpoint. Then, the query against the test-bed's Semantic Data Repository is performed (managed by the test-bed's Resource Manager);
2. If the information is there, the Resource Manager will gather the data and compose a response message, which will be addressed back to the experimenter, using again FIESTA-IoT as a relaying actor.

5.1.5.3 *Experiment queries/retrieves Data (Class-II & -III test-beds)*

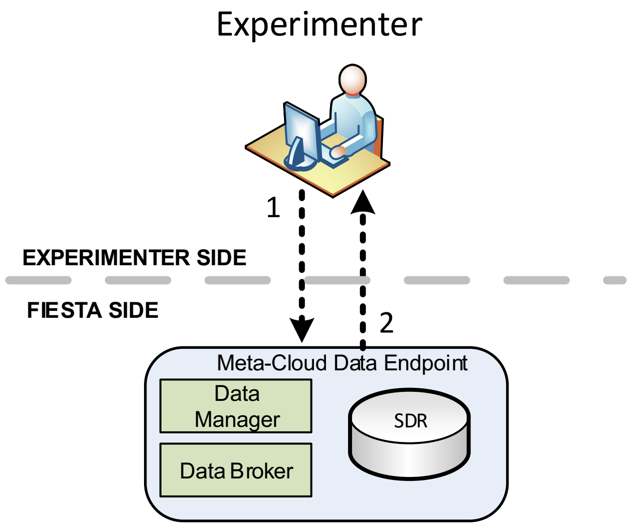


Figure 34: Data collection sequence diagram (Class-II & -III test-beds)

As already discussed in previous use-cases, all data generated by Class-II & -III test-beds is semantically annotated and replicated at the FIESTA-IoT level (namely in the Meta-Cloud Data Endpoint's Semantic Data Repository); thus storing it at the Semantic Data Repository. It goes without saying that this information accomplishes the ontologies defined for annotating data in FIESTA-IoT. As can be easily appreciated in Figure 34, the sequence diagram to retrieve data from those test-beds is rather straightforward:

1. The experimenter generates a data query (using e.g. SPARQL) and sends it to the Meta-Cloud Data Endpoint's Resource Broker. Then, the message will get the Data Manager, which extracts the raw query (e.g. SPARQL) snippet from the body of the message and the query is carried out in the Semantic Data Repository (SDR);
2. The response of this query will be sent back to the experimenter.

5.1.6 VE-oriented use-cases

5.1.6.1 *Virtualizer registers a VE*

The first thing to do in set an upper layer above the physical resources and their associated IoT Services involves the creation of VE instances, whose VE properties will be tightly linked to these thing-related stuff. In the same way as the rest of the use cases we have introduced so far, the category of the test-bed will have a huge relevance when it comes the time to interact with.

First, Class-I test-beds (already fully FIESTA-IoT compliant) manage locally the Virtual Entities it is dealing with and stores locally the VE descriptions. Those VE descriptions should contain at least:

- ObjectClass: the class of the object according to a domain ontology e.g. Bus;

- ObjectId: A unique ID used in order to name non-ambiguously the object when accessing to object properties;
- Address of the REST endpoint dealing with the Object (where REST get can be sent to e.g.);
- ObjectProperties: an object properties are qualified by a name, a class (e.g. temperature), a value (e.g. the current temperature);
- ObjectProperty binding: in addition to name, class and value, the property must be bound to a IoT service which exposes the resource that delivers the value to the VE property.

If a VE/IoT Service association has been set-up (see Section 5.1.6.3) observations made by the corresponding resource of test-bed (sensor readings) must refer to the VE/VE property they relate to, as part of the embedded metadata.

Moreover, a test-bed which is not fully FIESTA-IoT compliant (i.e. Class-II) has the possibility to register the VEs it is manipulating at the FIESTA-IoT level. The VE Wrapper functional component is then used and subsequent VE descriptions are sent to the VE Registry at the FIESTA-IoT side. Any request concerning VEs belonging to that test-bed will be answered by the VE Registry at FIESTA-IoT level via the VE endpoint (meaning that FIESTA-IoT is then responsible for maintaining values associated with the VE properties).

An explicit REST GET on a property of a given VE will result in a call (REST GET) from the VE Registry to the IoT Service endpoint of the test-bed so that the current value associated to the property can be received. For the sake of an easier visualization, the association/binding between VEs and Resources/IoT Services is handled in a separate use case (5.1.6.3)

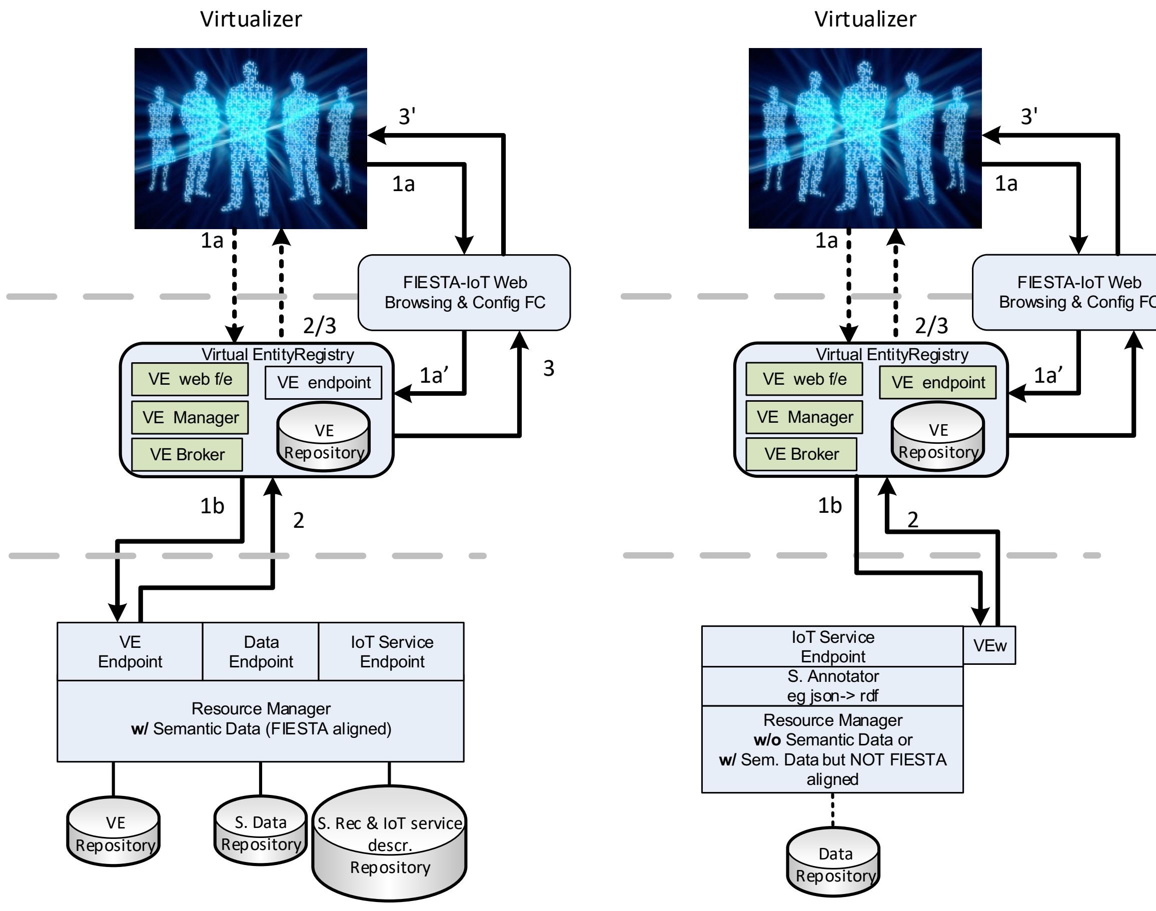
With regard to the last type, Class-III test-beds are not concerned with VEs and will only deal with requests for on-demand access to Resources/IoT Services and data queries.

After this thorough description, Figure 35 illustrates the sequence diagram (without including the VE/Resource association), whose messages meanings the outlined as follows:

1. Virtualizers build a VE layer upon an IoT service and generate a register request addressed to the VE manager, which is in charge of storing it into the VE repository. In this case, these requests can be directly handled by experimenters or, in a more straightforward manner, relying on the FIESTA-IoT Web Browsing & Configuration FC, which helps virtualizers to select among all the available resources to compose their own VEs. It is worth highlighting that this very initial message (1a) is addressed to either the VE Broker or the VE Web Front-end depending on whether the interaction is direct or through the browser, respectively. After that, the VE Browser forwards the message to either the Class-I test-bed VE endpoint (Figure 35a) or the Class-II test-bed VE Wrapper (Figure 35b);
2. For this response, there are two different ways to deal with: on the first hand, as seen in Figure 35a, Class-I test-bed will save locally all the VEs generated in the extent of FIESTA-IoT experimentation, so, through their VE endpoints (message 2), will share the VE descriptions with the FIESTA-IoT VE Registry

FC, which will use the VE Manager to record the information. On the other hand (Figure 35b), since Class-II test-beds do not directly "understand" VEs, they will compose the VE properties from the resource descriptions gathered by their Resource Managers. In this case, the VE Wrapper will be in charge of generating the VE description that will be stored in FIESTA-IoT, following the same steps seen for Class-I test-beds;

3. The FIESTA-IoT platform just sends back an ACK message, confirming the correct registration of the VE description. Alike the case for message 1, we have two ways to reach back the virtualizers (i.e. direct access or across the FIESTA-IoT Web Browser & Configuration FC).



a) Class-I test-bed registration b) Class-II test-bed registration

Figure 35: VE registration sequence diagram

Once the VE has been created/registered (i.e. empty state), the next and immediate state would be to select all those Resources/IoT Services that will define the VE properties of the entity. Last, but not least, it is worth highlighting two important issues to take into consideration regarding future VE issues: 1- Recall that the role of a Virtualizer can be played by either a test-bed or a third party. 2- It is important to highlight here the possibility of handling multi-level or nested VEs, giving rise to multidimensional VE registration/management.

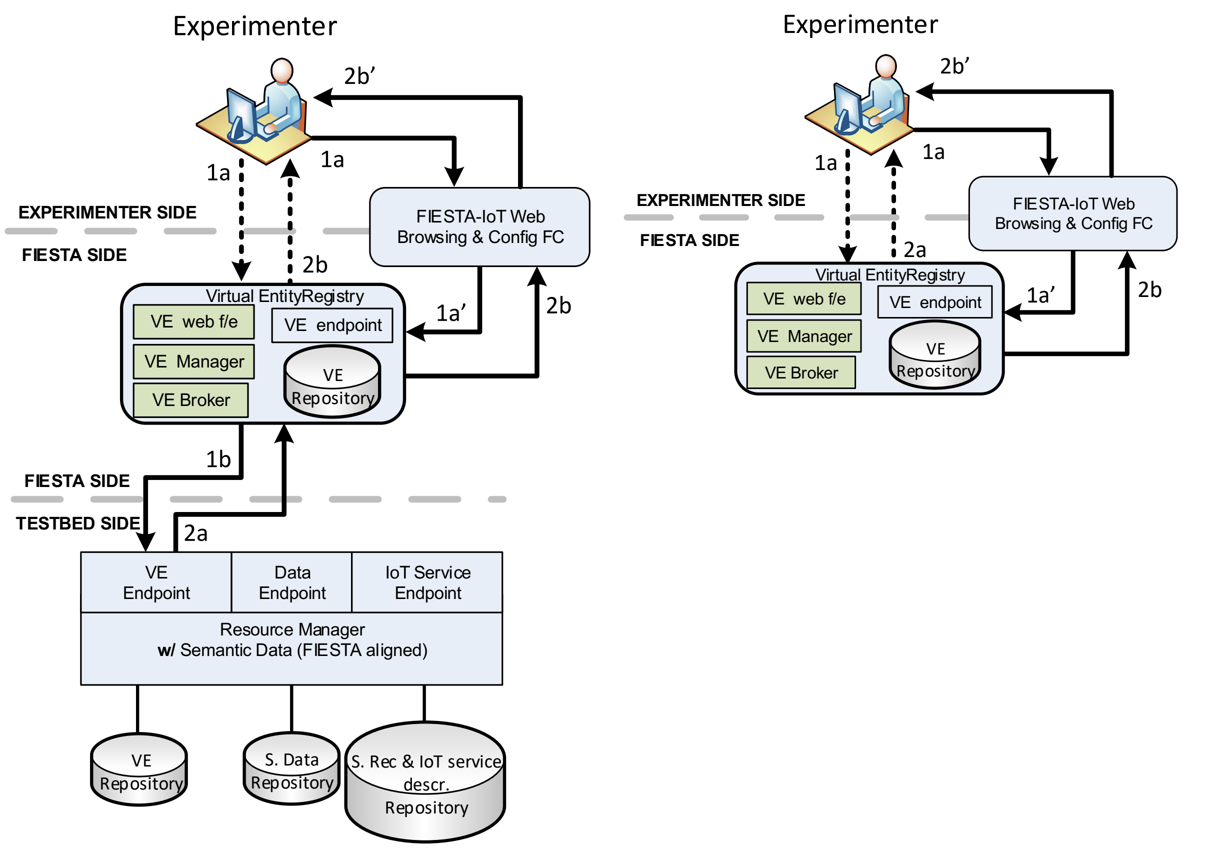
Resulting from this Association creation process, the VE registry will start managing the association, meaning polling values from the resource and maintaining VE properties values. As part of the result, it also starts exposing the VE at the FIESTA-IoT VE endpoint.

5.1.6.2 *Experiment looks up/browses VEs*

Figure 36 describes the messages exchanged from the moment the experimenter sends a first query, looking up a particular VE (or group of them) to the instant he/she receives the corresponding response. As can be seen in the different pictures, the access to the VE Entity Registry is twofold. End-users can either graphically browse among the different already-instanced VEs by means of the FIESTA-IoT Web Browsing & Configuration FC or, on the other hand, they can directly interact with the API that hooks to VE Broker, typically through explicit SPARQL requests.

Having into account that these two options are shared between the different types of test-beds, the sequence of message would look like this:

1. The first step in this process is to do a look-up on Virtual Entities that have been either declared to FIESTA-IoT (and then FIESTA-IoT is managing them) or which are managed locally at the test-bed side by Class-I test-beds. In both cases, the look-up request is managed at the FIESTA-IoT level and is either answered by the Virtual Entity Registry (for those VEs that are managed there, such as test-beds of Class-II & -III, as shown in Figure 36b) or by the VE endpoints at the test-bed side (i.e. Class-I test-beds, Figure 36a). In this later case, a SPARQL request (as VEs are described semantically) is forwarded to the VE endpoint. A typical request could be “get me all VEs of type Bus from the city of Santander” translated in the appropriate SPARQL query and according to a domain ontology used by the Santander test-bed;
2. The answer to such a request should be composed of VE descriptions that must also embed the VE endpoint address for further access to VE properties. As can be observed in both figures, it could either reach the end-user directly from the Virtual Entity Registry or through the FIESTA-IoT Web Browsing & Configuration FC.



a) Class-II: Through VE Broker (managed remotely by the federated test-beds)

b) Class-II: Through VE Manager (handled locally at FIESTA level)

Figure 36: VE search sequence diagram

5.1.6.3 Virtualizer creates an VE/IoT Service association

This use-case deals with the creation of associations at the FIESTA-IoT level using, as well as for the previous VE-related use-case, two ways to deal with: 1- through an specific API offered for that purpose by the VE Registry FC (via VE Browser), 2- Using the graphical interface (i.e. FIESTA-IoT Web & Configuration FC). In this stage, virtualizers (including the browsing of resources/IoT Services) need to select those resource descriptions which are to be bound to the VE, as shown in Figure 37. Said in other world, this particular use cases ties the concepts of two different realms, thus representing the physical entities in the virtual realm. Below we describe the sequence of messages followed in this process.

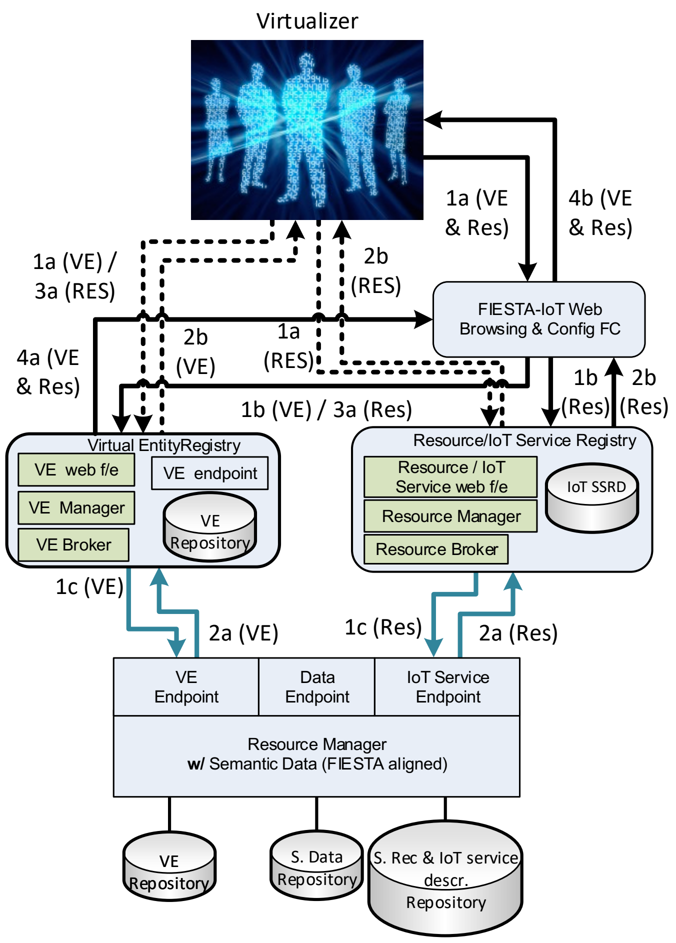


Figure 37: VE IoT Service Association Sequence Diagram

1. Virtualizers searches are divided in two parts: whilst on the one hand they browse/query the VE Entity Registry FC in order to find the VE instances they want to tweak, on the other side they do the same for Resources/IoT Service Registry FC. This latter one can be skipped if virtualizers have already the knowledge about all the resources that they want to bind to the VE(s). Besides that, it goes without saying once again that there are two options to establish a connection and get data from these functional components, heading the request to the Web Front-end or the Broker as a function of whether we are dealing with a graphical or API-based scheme, respectively;
2. Depending on the underlying test-bed(s) to which we are addressing the requests, they can either reach the test-bed VE/IoT Services endpoints (i.e. Class-I test-beds) and run the queries at FIESTA-IoT level, through the corresponding Managers (i.e. Class-II test-beds). Then, the responses are sent to a) the FIESTA-IoT Web Browsing & Configuration if the experimenters opted for a graphical interaction; otherwise, they will have to take care of these messages and “manually” establish the association;
3. The descriptions regarding the Resources/IoT Services are sent to the Virtual Entity Registry FC, where are appended to the VE properties;
4. If the process has been accomplished, an acknowledgement reaches the Virtualizer.

5.1.6.4 Experiment invoke VE-service (access to VE properties)

This use-case assumes that the use-case 5.1.2.3 has been performed, meaning that the “caller” knows the IDs of the VEs as well as the VE Services it wants to invoke. This following step therefore consists of sending REST requests to the FIESTA-IoT VE Registry, as shown in Figure 38. Depending on whether the VE is managed at FIESTA-IoT level or remotely (i.e. Class-I test-beds), the requests will be handled by the Class-I test-bed VE Endpoint, via the FIESTA-IoT VE Registry/VE Broker FC (Figure 38a), or the FIESTA-IoT VE Manager for Class-II test-beds (Figure 38b).

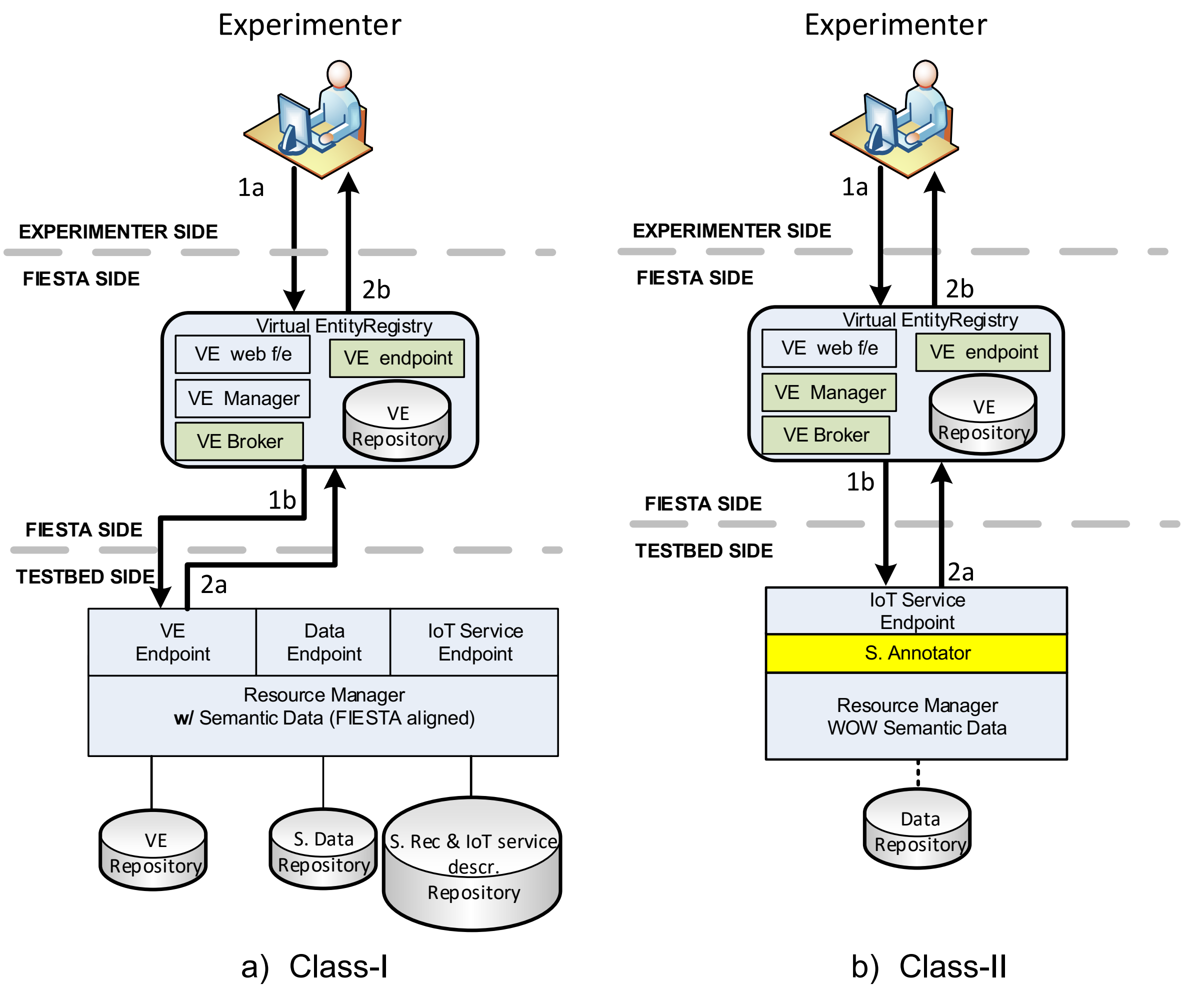


Figure 38: Service invocation use case sequence diagram

1. The experimenter sends the call addressed to the concrete VE Endpoint he/she wants to invoke, passing across the FIESTA-IoT architecture (message 1a). For Class-I test-beds (Figure 38a, message 1b), the VE Broker just has to forward the REST request to the test-bed’s VE Endpoint, which will be in charge of the gathering of all the info and the generation of the response. On the other hand, for Class-II test-beds, the process is rather different (Figure 38b, message 1b). After the request reaches the FIESTA-IoT VE Endpoint, since the information about the VE properties is held in the FIESTA-IoT VE repository, the VE Manager has to search this data there. This data includes the IoT Services that are associated to the VE, so the next step will consist in invoking (one by one) all the corresponding IoT Services. For that purpose,

the VE Broker will be in charge of bypassing the calls to the test-bed IoT Service Endpoint;

2. In this step, we assume that the IoT Service has been already run and the test-bed has to send the response back to the experimenter. Whereas Class-I test-beds have to directly deliver the response to the FIESTA-IoT VE Broker, as reflected by message 2a in Figure 38a, Class-II (recall that Class-III test-bed does not allow the use of VEs) test-beds need to rely on a Semantic Annotator to transform the information prior to its delivery (Figure 38b, message 2a). As can be easily inferred, messages 2b are the forwarded messages from FIESTA-IoT, headed to the experimenters.

6 FIESTA-IOT INSTANTIATION VIEW

6.1 Short description of actual components

6.1.1 Experiment related FCs

Creation, registration and execution of experiments are covered respectively by the *Experiment Management Console* (EMC), *Experiment Registry Module* (ERM), *Experiment Execution Engine* (EEE) and their dedicated APIs. An additional component, namely *Experiment Result Storage* (ERS), deals with the storage and retrieval of experiment results. Some of those components (ERM and EMC) are directly available from the platform portal. As far as the logical architecture is concerned those components span FCs located within the IoT Process Management, Service Organisation and Management functional groups.

6.1.2 General purpose FCs

Perfectly aligned to the logical architecture, the Reasoning and data Analytics (DAaaS) components provide added-value services to experimenters, who therefore can produce process data using some typical machine learning algorithms or reasoning functionalities without having to implement and deploy those. Of course additional in-house algorithms can be deployed and used as well, but they would reside at the experiment side and outside the platform. Both the Analytics and Reasoner concrete components span the IoT Service and Virtual Entity functional group, although the implemented components do not cover the virtual entity aspects.

6.1.3 Core-platform components

This second category concerns discovery functionalities and registries for historical data (observations) as well as messaging functionalities (Message Bus, MB Dispatcher) including also functionalities pertaining to establishing and controlling data flows from the testbed providers towards the platform in both push and pull mode (TPI DMS, TPS). The *Testbed Resource Registration* (TRR) offers registration and storing capabilities for testbed-hosted resource/IoT service (semantic descriptions) and testbed characteristics. An additional component, namely the Semantic & Syntactic Validator, aims at verifying that data pushed towards the registries follow FIESTA-IoT semantic guidelines (ontologies).

All those components span the IoT service, VE and Communication functional groups as well as some functionalities hosted at the testbed provider side (and therefore outside the platform).

6.2 Instantiation view

The instantiation view provides the developers' point of view of the FIESTA-IoT platform based on the logical architecture described in Section 4. It shows which concrete components have been developed and how they are interconnected with each other. This view described the status of the platform at the time this deliverable has been released. It is worth nothing that most of the concrete components names are different from the logical FCs names introduced in Section 4.3. This is due to the n-to-m correspondence between logical and concrete FCs. The mapping section (Section 6.3) elucidates how concrete components cover functionalities pertaining to the logical FCs.

Figure 39: Instantiation view

6.3 Mapping between logical components (raw) and concrete components (column)

This section provides a mapping table that summarises the correspondence between the logical functionalities as introduced in Section 4 and the concrete software components that implement those functionalities.

NOTE: Both the description of concrete components and the corresponding mapping reflect the status of FIESTA-IoT developments at the time this second and finale version of the FIESTA-IoT architecture has been released.

Concrete FCs→ Logical FCs ↓	TPI DMS	Message Bus	MB Dispatcher	IoT Registry	TPI Configurator	TRR	Semantic Annotator	Semantic & Syntactic Validator	Monitoring	DAaaS	ERS	EEE	EMC	ERM	Reasoning	TPS
Experiment Editor & ERM													X	X		
Experiment Execution Engine												X				
Experiment Result Storage											X					
FIESTA built-in Reasoner															X	
FIESTA Analytics (KAT)										X						
Experiment Management Console													X			
Testbed Web configurator					X											
Monitoring									X							
Web Browser & Config						X										
Meta-cloud VE Data Endpoint				x												
Meta-cloud Data				X												

Endpoint																
Message Bus		X														
Virtual Entity Registry						x										
IoT Service / Resource Registry						X										
Semantic Validator								X								
TPI DMS	X															
Message Bus Dispatcher			X													
Semantic Annotator							X									
IoT Service endpoint																X
VE Endpoint																X
VE Service						x										
VE Manager						x										
Data endpoint																X
Resource Manager						X										

Table 3: Mapping table between concrete and logical components

7 CONCLUSION

This Deliverable D2.4 (v2) presented the second and final release of the System Architecture for the FIESTA-IoT project. In a nutshell the improvements brought to D2.4 (v1) include:

- Updates and improvements in all sections (but Section1);
- Introduction to the Instantiation View in Section 3.3;
- Update of Test-bed taxonomy and definition of roles (in section 4.2 & 4.1 resp.) with UML use-cases
- Virtualizer example in 4.1.2.2
- Complete Functional View comprising definition of all experiment related FCs (top FG in the Functional View);
- Complete Information Views with additional System Use-cases –in particular-tackling the interactions between the Experiment plane and the already described FCs;
- Instantiation View: showing the mapping between logical components and implemented concrete software components.

ACKNOWLEDGMENT: The content of Section 6 has been elaborated on the basis of the material available from a FIESTA-IoT submitted paper (and therefore not yet public): "Experimentation as a Service over Semantically Interoperable Internet of Things Testbeds" by Jorge Lanza[1], Luis Sánchez[1], Juan Ramón Santana[1], Rachit Agarwal[2], Nikolaos Kefalakis[3], Paul Grace[4], Tarek Elsaleh[5], Mengxuan Zhao[6], Elias Tragos[7], Hung Nguyen[7], Flavio Cirillo[8,10], Ronald Steinke[9], John Soldatos[3] submitted to IEEE Access 15th of June 2018.

[1] Network Planning and Mobile Communications Lab. Universidad de Cantabria. Edificio Ingeniería de Telecomunicación. 39005 Santander, Spain.
[2] MiMove Team, Inria Paris, France.
[3]Athens Information Technology, 44 Kifisias Ave., 15125 Marousi, Athens, Greece.
[4] IT Innovation Centre, University of Southampton, UK.
[5] Institute for Communication Systems, University of Surrey, Guildford, GU2 7XH, United Kingdom.
[6] Easy Global Market, Espace Beethoven, 1200 Route des lucioles, 06560 Valbone, France.W
[7] Insight Centre for Data Analytics, NUI Galway, Ireland.
[8] NEC Laboratories Europe, Kurfuersten-Anlage 36, 69115 Heidelberg, Germany
[9] Fraunhofer Institute for Open Communication Systems FOKUS. Kaiserin-Augusta-Allee 31. 10589 Berlin, Germany.
[10] University of Naples "Federico II", Corso Umberto I, 40, 80138 Napoli NA, Italia

8 BIBLIOGRAPHY

(Cooper D., 2008) Cooper D., Santesson S., Farrell S. Boeyen S., Housley R., Polk W. (2008). *InternetX.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL)* (Project Management Institute, 2013)*Profile*. IETF RFC 5280, May.

(DigiMesh, 2015) DigiMesh Networking Protocol. (2015). http://www.digi.com/technology/digimesh/.

(FIESTA-IoT D2.2) FIESTA-IoT Deliverable D2.2 “Analysis of Platforms and Test-beds”

(FIESTA-IoT D2.3) FIESTA-IoT Deliverable D2.3 “Specification of Experiments, Tools and KPIs”

(FIESTA-IoT D3.1.1) FIESTA-IoT Deliverable D3.1.1 “Semantic Models for test-beds, interoperability and mobility support and best practices”

(FIESTA-IoT D5.4) FIESTA-IoT Deliverable D5.4: Best Practices for Experiments Design and Conduction", project FIESTA-IoT deliverable, 2018.

(Vandenberghe, 2013) Vandenberghe, W., Vermeulen, B., Demeester, P., Willner, A., Papavassiliou, S., Gavras, A., Boniface, M. (2013). *Architecture for the heterogeneous federation of future internet experimentation facilities*, In Future Network and Mobile Summit (FutureNetworkSummit), (pp. 1-11).

(initiative, 2015) FI-WARE initiative (Accessed 2015). https://www.fiware.org/.
FIWARE lab, the open innovation Lab (Accessed 2015). https://www.fiware.org/lab/.

(Jena) Carroll, Jeremy J., et al. *"Jena: implementing the semantic web recommendations." Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. ACM, 2004.

(Gluhak, 2011) Gluhak, A., et al. (2011). *A survey on facilities for experimental internet of things research*, IEEE Communications Magazine, vol.49, no.11, pp.58-67.

(project) Provisioning of urban / regional smart services and business models enabled by the Future Internet (OUTSMART) project (Accessed 2015). https://www.fi-ppp.eu/projects/outsmart/.

(Tonneau, 2015) Tonneau, A.S., Mitton, N., Vandaele, J. (July 2015) *How to choose an experimentation platform for wireless sensor networks? A survey on static and mobile wireless sensor network experimentation facilities*, Ad Hoc Networks, Volume 30, pp. 115-127.

(Gavras, 2010) Gavras, A. (2010). *Experimentally driven research white paper .* ICT-FIREWORKS .

(Haren, 2009) Haren, V. (2009). *TOGAF Verson 9.0.*

(IEEE, 2007) IEEE. (2007). *Guide for Monitoring, Information Exchange, and Control of Distributed Resources Interconnected with Electric Power Systems.* IEEE.

(IEEE, 1990) IEEE. (1990). *IEEE standard glossary of software engineering terminology.* IEEE.

(IoT-A, 2011) IoT-A. (2011, June 16). Retrieved May 29, 2015 from Project Deliverable D1.2 – Initial Architectural Reference Model for IoT: http://www.iot-a.eu/public/public-documents/d1.2/view

(IoT-A, 2013) IoT-A. (2013, July 15). Retrieved May 29, 2015 from Deliverable D1.5 – Final architectural reference model for the IoT v3.0: http://www.iot-a.eu/public/public-documents/d1.5/view

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

(Project Management Institute, 2013) Project Management Institute. (2013). *A Guide to the Project Management Body of Knowledge* (5th Edition ed.). USA.

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

(Volere) "Volere Requirement Specification Template" available at www.volere.co.uk (last accessed 31/07/2015)

(Haller *et al*., 2013) "A Domain Model for the Internet of Things", in iTHINGS'2013 proceeding, Beijing, China (see also IEEE eXplore)

(Rozanski&Woods, 2011) N. Rozanski and E. Woods, "Applying Viewpoints and Views to Software Architecture" , Addison Wesley, 2011

(IoT-A UNIs) "IoT-A Unified Requirements", available at http://www.iot-a.eu/public/requirements/copy_of_requirements (last accessed 31/07/2015)

FIESTA-IoT 2015