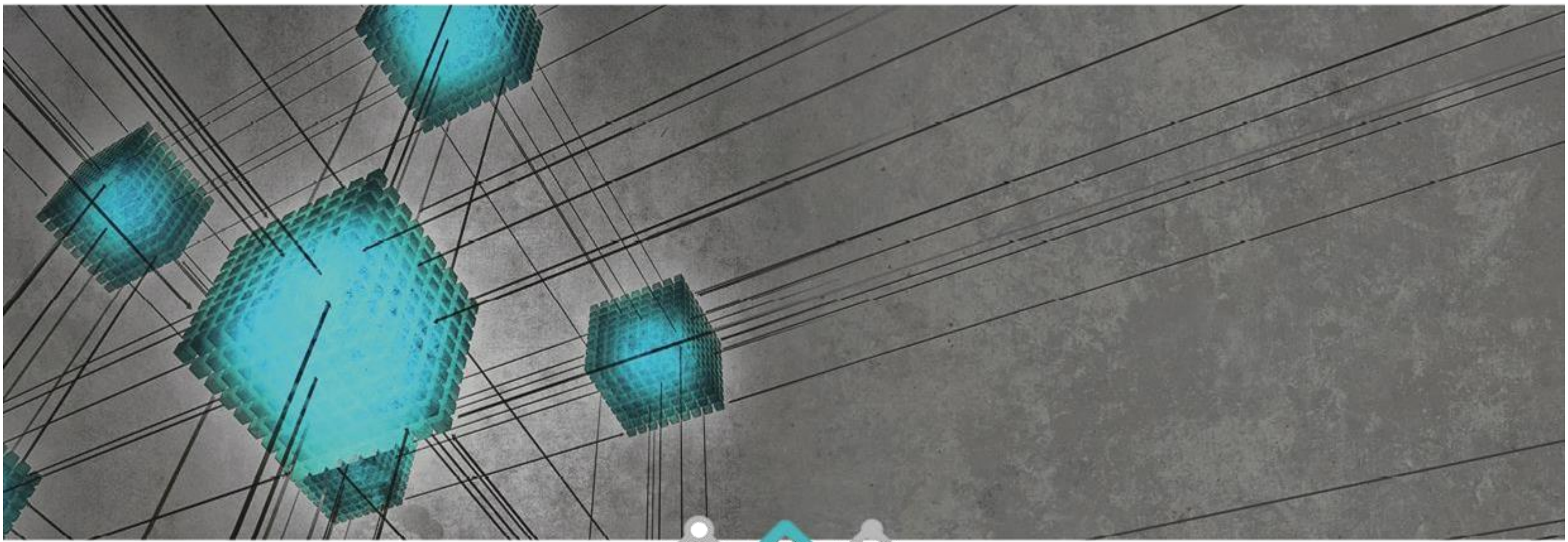


INTEROPERABILITY
OF HETEREOGENEUS
IOT PLATFORMS.

Deliverable D4.3

Initial Reference IoT Platform Meta-Architecture and Meta Data Model Interoperable IoT framework Model and Engine v1

7th October 2017

INTER-IoT

INTER-IoT aim is to design, implement and test a framework that will allow interoperability among different Internet of Things (IoT) platforms.

Most current existing IoT developments are based on “closed-loop” concepts, focusing on a specific purpose and being isolated from the rest of the world. Integration between heterogeneous elements is usually done at device or network level, and is just limited to data gathering. Our belief is that a multi-layered approach integrating different IoT devices, networks, platforms, services and applications will allow a global continuum of data, infrastructures and services that can will enable different IoT scenarios. As well, reuse and integration of existing and future IoT systems will be facilitated, creating a defacto global ecosystem of interoperable IoT platforms.

In the absence of global IoT standards, the INTER-IoT results will allow any company to design and develop new IoT devices or services, leveraging on the existing ecosystem, and bring get them to market quickly.

INTER-IoT has been financed by the Horizon 2020 initiative of the European Commission, contract 687283.

INTER-IoT

Initial Reference IoT Platform Meta-Architecture and Meta Data Model Interoperable IoT framework Model and Engine v1

Version: 1.0

Security: Public

30/09/2017

The INTER-IoT project has been financed by the Horizon 2020 initiative of the European Commission, contract 687283

Disclaimer

This document contains material, which is the copyright of certain INTER-IoT consortium parties, and may not be reproduced or copied without permission.

The information contained in this document is the proprietary confidential information of the INTER-IoT consortium (including the Commission Services) and may not be disclosed except in accordance with the consortium agreement.

The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the project consortium as a whole nor a certain party of the consortium warrant that the information contained in this document is capable of use, nor that use of the information is free from risk, and accepts no liability for loss or damage suffered by any person using this information.

The information in this document is subject to change without notice.

Executive Summary

The present document is the deliverable D4.3: Interoperable IoT framework Model and Engine v1 which reports about the results in the software design and implementation tasks (T4.4 - T4.5) of the WP4 in the project INTER-IoT. This document gathers a significant part of the outputs of these tasks, very specially the task T4.3 started in January 2017(M13), that aims to provide a proper design for a configuration and management framework and toolset, making easier the administration and use of the INTER-Layer mechanisms, and its twin task, T4.4, which takes care of the implementation of the outputs the design performed.

The report gives a full architectural view from the requirements until the communication between software components. It reflects the design work performed to univocally specify the software built to support the goals of the INTER-FW and INTER-API products of the project.

Structure

This document contains a report with the initial design carried out to define a full-fledged framework to enable administration, use and expansion of the multi-layered interoperability solution for heterogeneous platforms.

The report, which is intended to be a reference document for the technical implementation of the different modules that compose the INTER-Framework, is structured in a first initial glossary and study of the different technologies and concepts that are closely related to the design and implementation task, referring when appropriate some available alternatives for the technological implementation. Secondly, it addresses the analysis of the solution to be developed, separating concerns in functional aspects as commonly found in the modern software engineering analysis processes. After this, a complete design of all the different parts of the software is depicted, defining a general component architecture, security implementations and graphical design. Finally, the design of the specification of the single-uniform API to access the different interoperability layer features is specified, allowing to understand the capabilities of the INTER-FW solution and how it interfaces with each layer.

Relation with other deliverables

The document is the first version of a total of two reports. This version includes state of the art of related technologies, requirements analysis and software design, while the second version (D4.4) will be more focused in usability, deployment, third party developers contributions and features refinement. This second version will also report about all the updates and upgrades made to the software during the last period of development (M21-M30). Some of the design ideas and diagrams reported on the present document are developed and released in D4.5: Interoperable IoT Framework and tools, released also in M21.

This report also has a tight relation with D4.1 (Initial Reference IoT Platform Meta-Architecture and Meta Data Model), as far as the concepts proposed in that document are applied in this one, as it is explained in section 3.2.1. Due to this, this document is also related with D4.2, to be released in M24 and which will contain a refinement of the concepts proposed in D4.1.

Deliverables D3.1(M12) and D3.2(M22) describe the layer-oriented interoperability principles which are managed and used in the framework concepts and, thus, these deliverables represent a basis for understanding the analysis and design presented in this work.

Finally, the design and implementation of the INTER-FW is validated and tested in most of the activities planned in WP6 (pilots and open calls) and reported in D6.1, D6.2 and D6.3, so that D4.3 is an input for these activities.

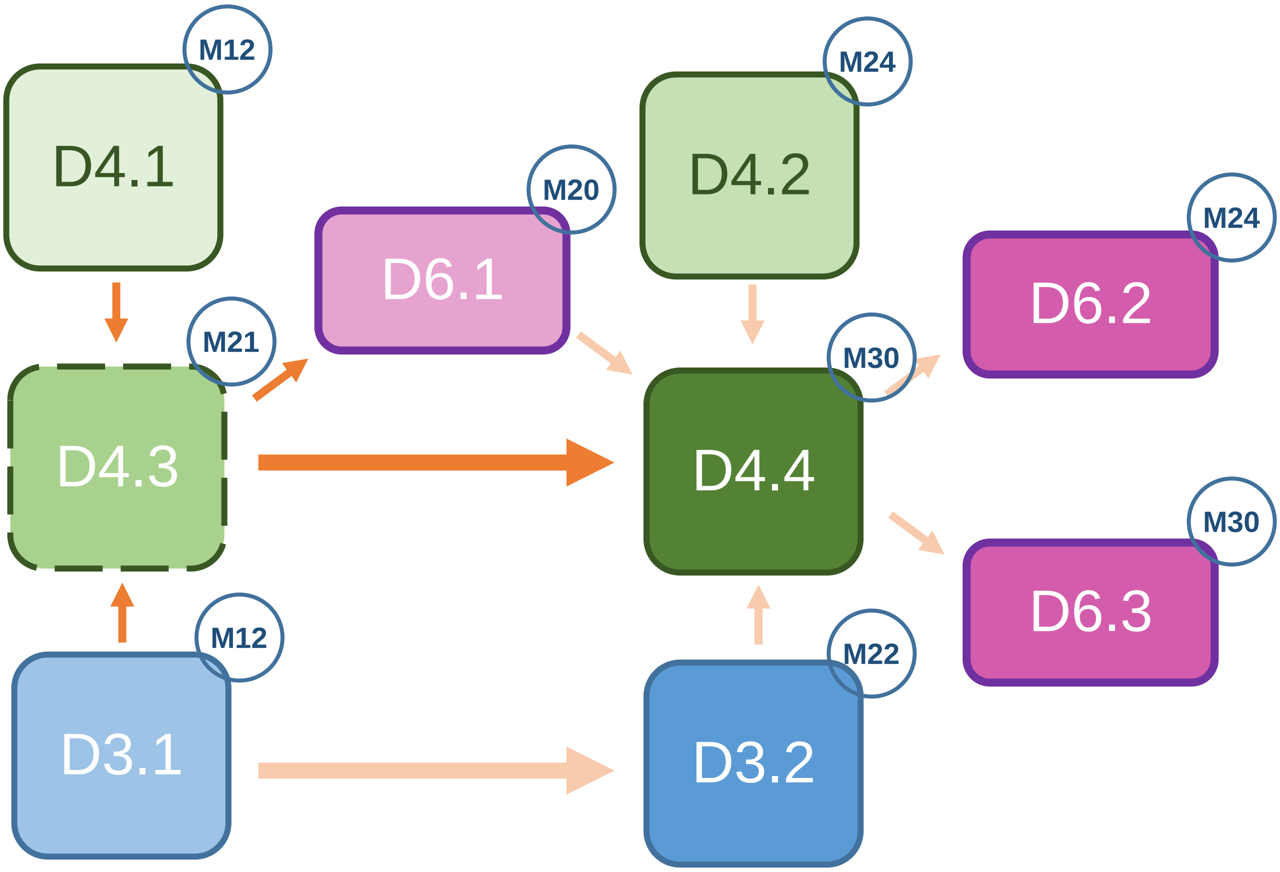


Figure 1 Main relations of this deliverable

Methodology followed

For the analysis of the INTER-FW and INTER-API, a curation of the user requirements has been performed. After this, use cases with user histories were identified, following an agile approach. A set of features and mockups was extracted from this analysis. Finally, a formal design of the components to be instantiated, data models, entities to be stored and main interactions has been made.

List of Authors

Organisation	Authors
Prodevelop	Miguel A. Llorente, Miguel Montesinos, Miguel Llàcer, Julián Martínez
ValenciaPort Foundation	Pablo Giménez
Systems Research Institute, Polish Academy of Sciences	Katarzyna Wasielewska-Michniewska, Paweł Szmeja
XLAB	Flavio Fuart, Matevž Markovič
Università Della Calabria	Raffaele Gravina
Rinicom	Eric Carlson
Universidad Politécnica de Valencia	Jara Suárez, Eneko Olivares, Andreu Belsa

Reviewed

Organisation	Name
Association pour le Développement de la Formation en Transports et Logisitique	Moncef Semichi
Universidad Politécnica de Valencia	Álvaro Fides

Edited

Organisation	Name
Prodevelop	Miguel A. Llorente
XLAB	Flavio Fuart

Approved

Universidad Politécnica de Valencia	Prof. Carlos E. Palau

Change control datasheet

Version	Date	Changes	Chapters	Pernters
0.1	07/06/17	ToC	All	PRO
0.2	14/06/17	Comments for writers and slight formatting for Google Drive sharing.	All	PRO
0.4	21/06/17	INTER-FW Overview		PRO, UPV, SRIPAS, VPF
0.3	28/06/17	Official template and formatting	All	PRO
0.5	31/07/17	Added sequence diagrams	2	PRO, UPV, UNICAL, SRIPAS, RINI, VPF
0.7	22/08/17	Complete review, introduction to some sections added. Added class diagrams for back-end analysis. Added mock-up screenshots.	All	PRO, VPF
0.8	28/08/17	Some pictures and references added	1	PRO
0.85	01/09/17	Improved sections 3.1, 3.3; Inputs from VPF (3.5).	3	PRO, VPF
0.92	04/09/17	Added entity relationship diagram. Added communication diagram.	3	PRO, VPF, UPV, XLAB
0.95	11/09/17	Security and relation with INTER-Layer; added tables to sequence diagrams, improved images, rearranged requirements, reviewed models.	2, 3, 4	PRO, VPF, UPV, RINI, SRIPAS
0.98	18/09/17	Minor changes, executive summary	0	PRO, VPF, SRIPAS, XLAB, UPV
1.02	03/09/17	Changes after internal review	All	PRO, UPV, AFT

Contents

Executive Summary ... 3
List of Authors ... 5
Change control datasheet ... 6
Contents ... 7
List of Figures ... 10
List of Tables ... 12
Acronyms ... 13
1 Background ... 15
1.1 INTER-Framework overview ... 15
1.2 Introduction to frameworks ... 16
1.3 Types of Frameworks and uses in INTER-IoT ... 19
1.3.1 Agent-based frameworks ... 20
1.3.2 Cloud based frameworks ... 21
1.3.3 REST-based frameworks ... 24
1.3.4 Reactive streams based frameworks ... 25
1.3.5 SOA-based frameworks ... 28
2 INTER-FW Analysis ... 32
2.1 Analysis overview ... 32
2.2 Use cases and sequence diagrams ... 32
2.2.1 Log In ... 32
2.2.2 API/Token generation ... 34
2.2.3 Add a new platform ... 34
2.2.4 Update platform configuration ... 36
2.2.5 Remove a platform ... 37
2.2.6 Add or register a virtual instance of a gateway ... 38
2.2.7 Update configuration of a virtual instance of a gateway ... 38
2.2.8 Remove a virtual instance of a gateway ... 39
2.2.9 Connect services with AS2AS tool ... 40
2.2.10 List permissions ... 42
2.2.11 Manage platform permissions ... 43
2.2.12 List IPSM instances ... 44
2.2.13 Register IPSM instance ... 45
2.2.14 Unregister IPSM instance ... 46
2.2.15 List alignments in IPSM ... 47
2.2.16 List alignments in Semantic Repository ... 48
2.2.17 Add a new alignment to Semantic Repository ... 49
2.2.18 Add a new alignment to IPSM ... 51
2.2.19 Delete an alignment from IPSM ... 52
2.2.20 Delete an alignment from Semantic Repository ... 53
2.2.21 Delete an ontology from Semantic Repository ... 54
2.2.22 Get an alignment from IPSM ... 55
2.2.23 View an alignment from IPSM ... 56
2.2.24 List active IPSM translation channels ... 57
2.2.25 Create a new translation channel in IPSM ... 58
2.2.26 Delete a translation channel in IPSM ... 59
2.2.27 List ontologies in Semantic Repository ... 60
2.2.28 View an ontology in Semantic Repository ... 61
2.2.29 View an alignment in Semantic Repository ... 62

2.2.30 View an alignment history in Semantic Repository ... 63
2.2.31 Get an ontology from Semantic Repository ... 64
2.2.32 Get an alignment from Semantic Repository ... 65
2.2.33 Add a new ontology to Semantic Repository ... 65
2.2.34 Add a new alignment to Semantic Repository ... 67
2.2.35 Add a new alignment to Semantic Repository ... 68
2.2.36 Change user data ... 68
2.2.37 Set up QoS requirements ... 69
2.2.38 Visualize SDN topology changes ... 70
2.3 Back-end analysis ... 71
2.3.1 Requirements ... 71
2.3.2 UML class diagrams ... 72
2.3.3 Technologies ... 78
2.4 Front-end analysis ... 79
2.4.1 Requirements ... 79
2.5 API analysis ... 80
2.5.1 API Requirements Analysis ... 80
2.5.2 First iteration: Inter-Layer exposed API ... 81
2.5.3 Second iteration: Security and configuration management ... 82
3 INTER-FW Design ... 83
3.1 Overview ... 83
3.2 INTER-IoT RA Instantiation ... 83
3.2.1 INTER-IoT RA instantiation ... 85
3.2.2 INTER-FW Functional Components ... 87
3.2.3 Functional Components traceability ... 99
3.3 INTER-FW Components ... 102
3.3.1 Communication diagrams ... 103
3.3.1 Entity relationship diagram ... 108
3.4 INTER-FW security management ... 109
3.5 INTER-FW GUI design ... 111
3.5.1 Main screen (landing page) ... 111
3.5.2 Platforms view ... 112
3.5.3 Gateway view ... 114
3.5.4 Network view ... 117
3.5.5 Service view ... 121
3.5.6 Semantic view ... 122
3.5.7 API management view ... 125
3.5.8 Configuration view ... 126
3.6 INTER-FW relation with INTER-Layer ... 127
4 INTER-API ... 129
4.1 API Management ... 129
4.2 API description ... 131
4.2.1 API Specification tools ... 131
4.2.2 Comparison ... 132
4.2.3 Other API description languages and solutions ... 132
4.2.4 Conclusions ... 133
4.3 API design ... 133
4.3.1 Inter-Layer REST API Endpoints ... 134
5 Ethical issues ... 138
5.1 Introduction ... 138
5.2 Ethics and INTER-FRAMEWORK ... 138

5.2.1 Data types 138
5.2.2 Requirements for ethical data processing 139
6 Conclusions and future work 141
7 References 142
8 Annex I User requirements 144
9 Annex II Requirements – use cases mapping 160

List of Figures

Figure 1 Main relations of this deliverable 4
Figure 2 Multi-layered approach of INTER-IoT 15
Figure 3 INTER-FW overview 16
Figure 4 Diagram of libraries, framework and application relationship. 17
Figure 5 A classical representation of the difference between library and framework 18
Figure 6 A representation of the relationship between libraries and APIs 19
Figure 7 Cloud-based frameworks classification 22
Figure 8 IoT online survey Source: IEEE IoT 22
Figure 9 The four attributes of a reactive system 25
Figure 10 Main interfaces in reactive streams 26
Figure 11 Example of service oriented architecture based framework 29
Figure 12 SOA Aspects in INTER-IOT 31
Figure 13 D2D UML class diagram 73
Figure 14 N2N UML class diagram 74
Figure 15 MW2MW UML class diagram 75
Figure 16 AS2AS UML class diagram 76
Figure 17 DS2DS UML class diagram 77
Figure 18 User management UML class diagram 78
Figure 19 High-level functionalities architectural approach in INTER-FW 83
Figure 20 Process followed for generating the INTER-FW architecture. 84
Figure 21 Functional-decomposition viewpoint of the INTER-IoT Reference Architecture 85
Figure 22: Functional Groups of the INTER-IoT Reference Architecture involved in the INTER-FW 85
Figure 23: Functional Components of the INTER-FW architecture according to the INTER-IoT Reference Architecture. 88
Figure 24: Functional Components of the Management FG instantiated for INTER-FW concrete architecture. 90
Figure 25: Functional Components of the Security FG instantiated for INTER-FW concrete architecture. 92
Figure 26: Functional Components of the Device Access FG instantiated for INTER-FW concrete architecture. 95
Figure 27: Functional Components of the Platform Interoperability FG instantiated for INTER-FW concrete architecture. 96
Figure 28: Functional Components of the Service Interoperability FG instantiated for INTER-FW concrete architecture. 97
Figure 29: Functional Components of the Semantics FG instantiated for INTER-FW concrete architecture. 98
Figure 30 UML Component diagram of INTER-FW administration and configuration framework 102
Figure 31 Login use cases family: communication diagram 104
Figure 32 Manage platforms use cases family: communication diagram 105
Figure 33 Manage gateways use cases family: communication diagram 105
Figure 34 Manage permissions use cases family: communication diagram 106
Figure 35 Manage IPSM use cases family: communication diagram 107
Figure 36 ERD of INTER-FW (SQL style) 108
Figure 37 Security agents in INTER-IoT 110
Figure 38. INTER-FW web application login page 111
Figure 39. INTER-FW web application main view 112
Figure 40. INTER-FW web application platforms view 113

Figure 41. INTER-FW web application add platform view 113
Figure 42. INTER-FW web application platform's device list view 114
Figure 43. INTER-FW web application platform's device details view 114
Figure 44. INTER-FW web application gateway view 115
Figure 45. INTER-FW web application add gateway view 115
Figure 46. INTER-FW web application manage gateway devices view...................................... 116
Figure 47. INTER-FW web application show gateway device details view................................. 116
Figure 48. INTER-FW web application network view.. 117
Figure 49. INTER-FW web application manage network elements view 118
Figure 50. INTER-FW web application add network elements view 118
Figure 51. INTER-FW web application network element configuration view 119
Figure 52 QoS queue configuration 119
Figure 53 QoS virtual switch configuration 120
Figure 54 QoS rules configuration view 120
Figure 55. INTER-FW web application service view.. 121
Figure 56. INTER-FW web application add node view 121
Figure 57. INTER-FW web application add flow view.. 122
Figure 58. IPSM general configuration view 123
Figure 59. Ontologies repository view... 123
Figure 60. IPSM Translation channels view... 124
Figure 61. IPSM Configuration view 124
Figure 62.IPSM Alignment configuration view 125
Figure 63. Alignments repository view 125
Figure 64. INTER-FW web application API management view 126
Figure 65. INTER-FW web application configuration view.. 126
Figure 66 Relation with INTER-Layer diagram 127
Figure 67. INTER-API manager structure.. 130
Figure 68. Swagger editor IPDM component.. 133

List of Tables

Table 1 Backend related user requirements 72
Table 2 Frontend related requirements 79
Table 3 API Requirements analysis 81
Table 4 INTER-API Frontend in the INTER-IoT RA 89
Table 5 Web Console Frontend in the INTER-IoT RA 90
Table 6 Web Configuration FC use cases and analysis 91
Table 7 Fault FC use cases and analysis 91
Table 8 Member FC use cases and analysis 92
Table 9 State FC use cases and analysis 92
Table 10 Authorisation FC use cases and analysis 93
Table 11 Authentication FC use cases and analysis 94
Table 12 Key Exange and Management FC use cases and analysis 94
Table 13 Identity Management FC use cases and analysis 94
Table 14 Virtual Entity FC use cases and analysis 95
Table 15 Platform Resolution FC use cases and analysis 97
Table 16 Service resolution FC use cases and analysis 98
Table 17 Ontology resolution FC use cases and analysis 99
Table 18 Traceability matrix of Functional Components of INTER-FW vs. use cases 100
Table 19 Traceability matrix of Functional Components of INTER-FW vs. INTER-FW architectural components 101
Table 20 Manage networks use cases family: communication diagram 107
Table 21 Ethical issues related requirements 140
Table 22 Requirements - use cases mapping part I 160
Table 23 Requirements - use cases mapping part II 161
Table 24 Requirements - use cases mapping part III 162
Table 25 Requirements - use cases mapping part IV 163

Acronyms

API	Application Programming Interface
AS2AS	Application and Services to Application and Services
AWS	Amazon Web Services
CL	Cross Layer
COAP	Constrained Application Protocol
CSS	Cascade Style Sheets
D2D	Device to Device
DS2DS	Data and Semantics to Data and Semantics
DSL	Domain Specific Language
ERD	Entity Relationship Diagram
FC	Functional Component
FG	Functional Group
GUI	Graphic User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol – Secure
IEEE	Institute of Electrical and Electronics Engineers
ILI	Interoperability Layer Infrastructure
INTER-FW	Inter Framework
IoC	Inversion of control
IoT	Internet of Things
IOT-A	Internet of Things – Architecture
IPSM	IoT Platforms Semantic Mediator
JADE	Java Agent DEvelopment Framework
JIMAF	Java-based Interoperable Mobile Agent Framework
JMS	Java Messaging Service
JS	Javascript
JSON	Javascript Object Notation
JSON-LD	Javascript Object Notation for Linked Data
JVM	Java Virtual Machine
KVP	Key Value Pairs
MAPS	Mobile Agent Platform for SunSPOT
MAS	Multi-agent System
MD	Markdown
MFC	Microsoft Foundation Classes
MQTT	Message Queue Telemetry Transport
MS	Microsoft
MW2MW	Middleware to Middleware
N2N	Network to Network
OASIS	Organization for Advancement of Structured Information Standards
OPC-UA	Object Linking and Embedding for Process Control – Unified Architecture
OS	Operative System
OSGi	Open Services Gateway initiative
QoS	Quality of Service
RA	Reference Architecture
RAML	RESTful API Modeling Language
REST	Representational State Transfer architectural style
RPC	Remote Procedure Call
SCA	Service Component Architecture

SDK	Software Development Kit
SOA	Service Oriented Architecture
SOAIF	Service Oriented Architecture Integration Framework
SOAP	Simple Object Access Protocol
SORCER	Service-oriented computing environment
SQL	Structured Query Language
UML	Unified Modelling Language
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
WCF	Windows Communication Foundation
WSDL	Web Services Description Language
WSN	Wireless Sensor Networks
YAML	YAML Ain’t Markup Language

1 Background

1.1 INTER-Framework overview

The modular layered approach followed in INTER-IoT has multiple advantages such as strong flexibility, adaptability to very different scenarios, separation of concerns, decoupling between functional components, scalability across physical tiers, etc. However, the approach also presents shortcomings in terms of usability or tooling unification, especially in the case of INTER-IoT, where each Interoperability Layer Infrastructure (ILI) is designed to be completely decoupled, being able to work standalone.

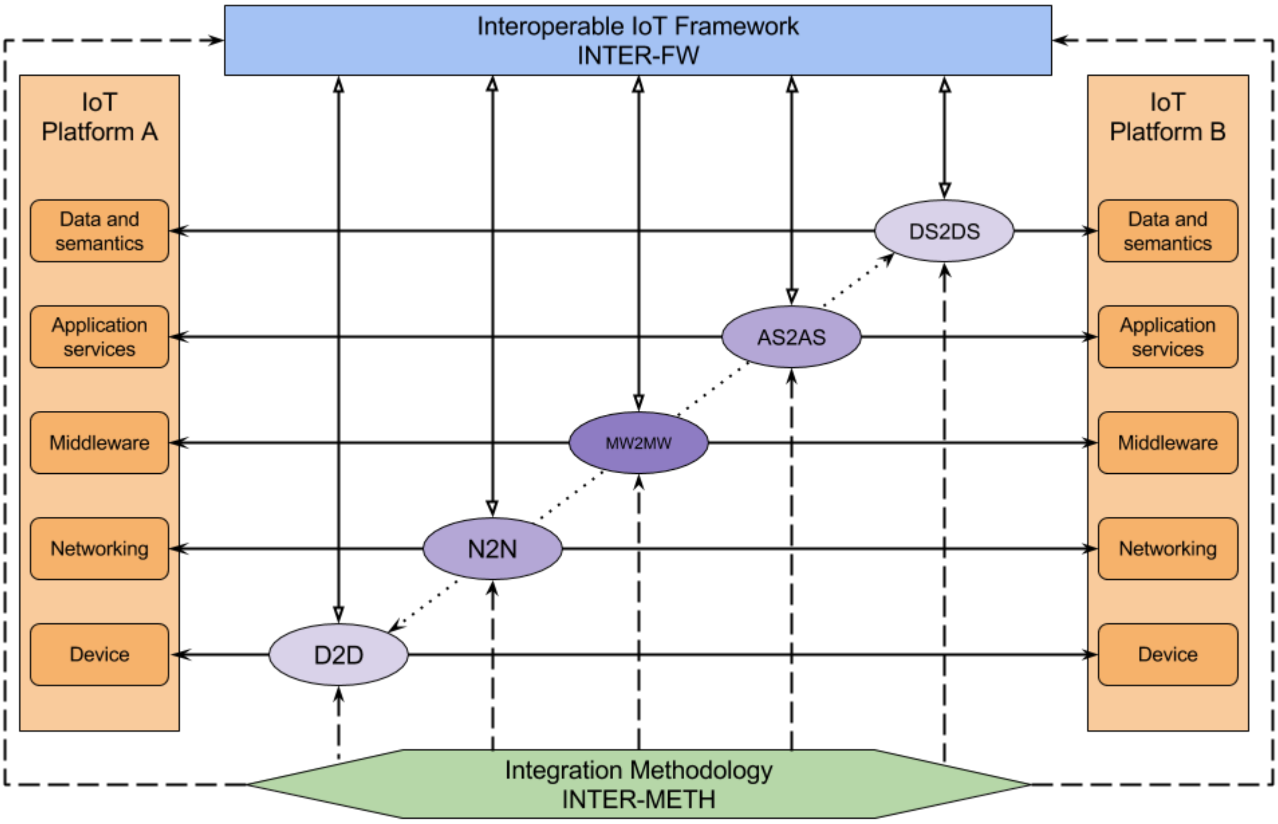


Figure 2 Multi-layered approach of INTER-IoT

Other important challenge of the INTER-IoT global design, affecting each layer separately is the future-proof design. One of the fundamental problems solved in INTER-IoT, the isolation of IoT data in information domains and the inability to automatically share these data across these domains, is a consequence of the excessive short-term sight in the design or instantiation of IoT systems.

INTER-IoT addresses these described drawbacks by the introduction of a framework based on the six ILIs (D2D, N2N, MW2MW, AS2AS, DS2DS and CL) exposed APIs. This framework, presented as the INTER-FRAMEWORK or INTER-FW has three main goals identified:

1. Enable extension and scalability of the INTER-IoT solution to support present and future applications or more demanding scenarios.
2. Configure, monitor and manage the different layers from a unique view.

3. Provide and manage a REST-like API to enable the use of the interoperable platforms and INTER-IoT features.

These goals are addressed in the following functional components developed in INTER-FW:

1. Software development framework: a combination of software results, documentation, templates and examples allowing extensibility through the application of the INTER-IoT interoperability patterns in new platforms, devices, services, etc.
2. Instantiation, configuration and management framework and toolset: a web based application which unifies in a single environment the monitoring, management configuration of the different IoT components (sensors, network elements, gateways, platforms...) at each interoperability layer. This element also includes key cross-layer elements management, such as authentication or authorization configurations.
3. A global, unified API to offer a single-entry point to application and services developers, either platform owners or third parties.

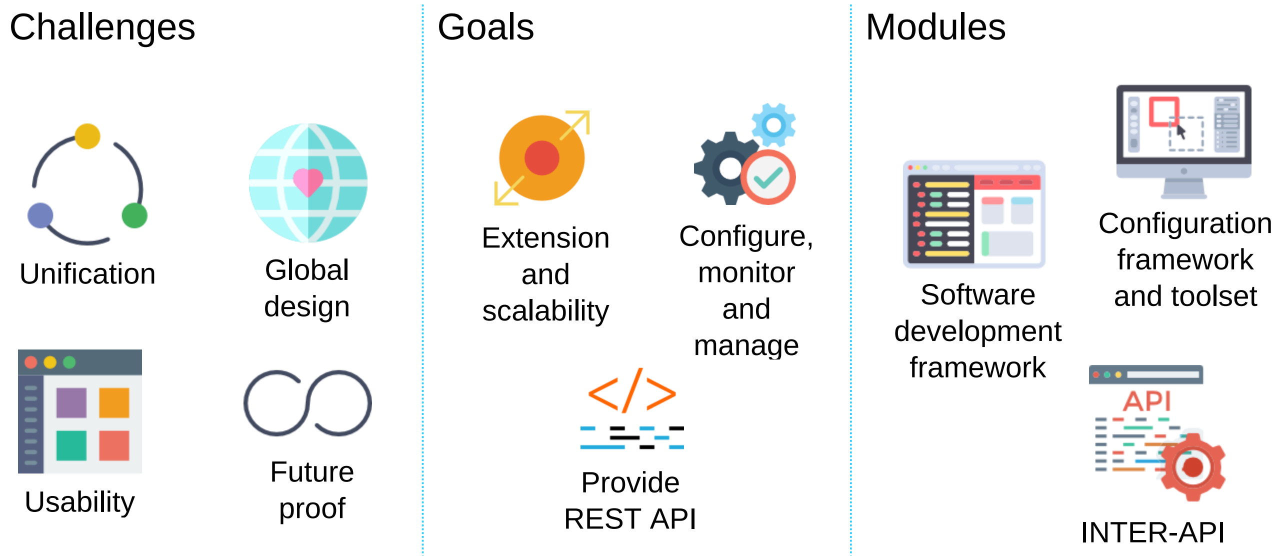

Challenges
Unification
Global design
Usability
Future proof
Goals
Extension and scalability
Configure, monitor and manage
Provide REST API
Modules
Software development framework
Configuration framework and toolset
API
INTER-API

Figure 3 INTER-FW overview

1.2 Introduction to frameworks

In software engineering, the concept o ‘framework’ is wide and can be applied in very different disciplines. The idea of what a framework is depends strongly in the approach and the goal of the software artefact resulting. However, there are some related concepts that very frequently are related to the software framework. Some of them are quickly discussed in this section.

Library

A library is a collection of code relating to a specific task, or set of closely related tasks which operate in each domain. Libraries are developed to be called by external agents, avoiding any control over the applications.

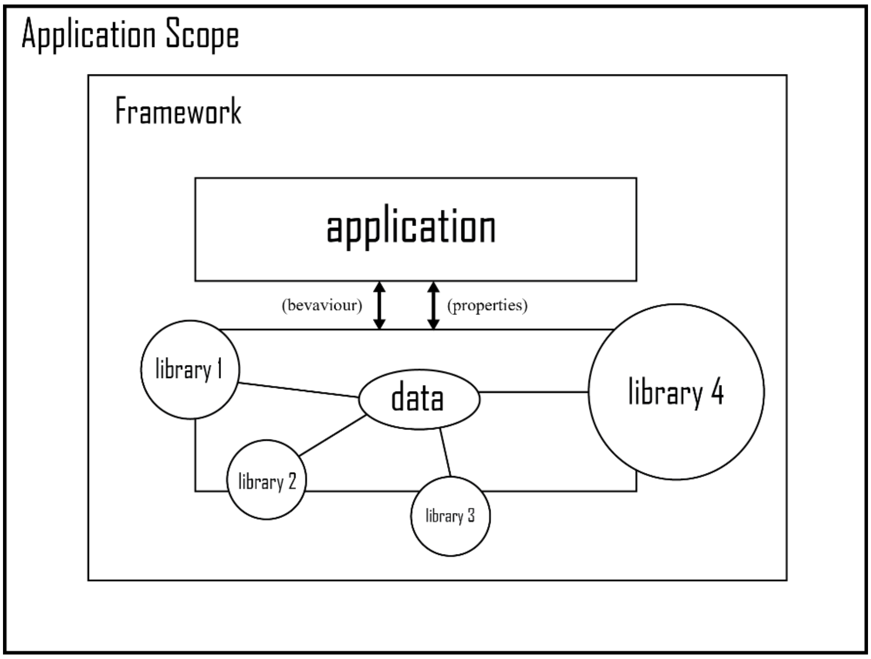


Figure 4 Diagram of libraries, framework and application relationship.

Toolset

Historically, a toolset is a more focussed library, with a defined and specific purpose and includes advanced interfaces solutions such as graphical widgets or GUI elements either in desktop or web environments. Toolsets usually operate in a higher level of abstraction of a library, being very often library consumers (it is very common to find libraries distributions with an associated toolset e.g. mosquitto[1], jmeter[2], docker[3], etc.).

Framework

A framework is a suite of interrelated libraries and software modules aimed to solve a problem or a set of coupled problems. The definition of a framework is usually linked to one of its key properties, the Inversion of Control (IoC) principle, which means that the program flow is taken over by the framework itself (a pre-existing and external element to the program), instead of the common case in imperative programming, where it is controlled by the developed program.

However, frameworks are also targeted either towards a specific output; an application for a specific OS for example (MFC for MS Windows for example), or for more general-purpose work (Spring framework for example).

[1] https://mosquitto.org/

[2] http://jmeter.apache.org/

[3] https://www.docker.com/products/docker-toolbox

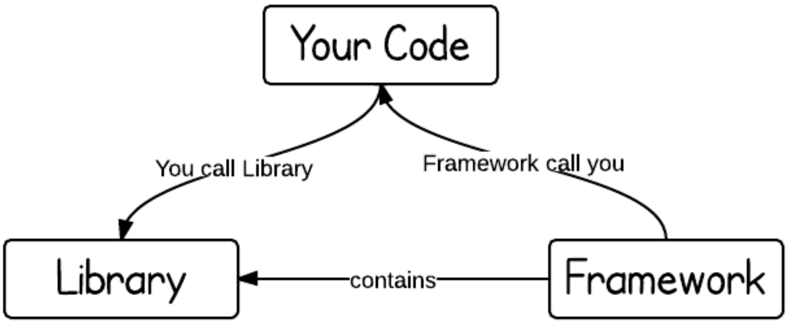


Figure 5 A classical representation of the difference between library and framework

Software Development Kit (SDK)

An SDK is a collection of tools to assist the programmer to create and deploy code/content which is very specifically targeted to either run on a very particular platform or in a very particular manner. An SDK can consist of simply a set of libraries which must be used in a specific way only by the client code and which can be compiled as normal, up to a set of binary tools which create or adapt binary assets to produce its (the SDK's) output.

Engine

An Engine (in computer science terms) is usually referred to a central, core part of a software system or subsystem. An engine also may be defined as software that facilitates automated processes, in which different software elements work interactively to minimize human intervention. Examples of software engines include relational database engines, workflow engines, inference engines and search engines. A common characteristic of software engines is metadata that provides models of the real data that the engine processes. Software modules pass data to the engine, and the engine uses its metadata models to transform the data into a different state.

Application Programming Interface (API)

An API offers up a contracted external interface to another piece of software (Client code to the API). More formally, an API is a set of rules ('code') and specifications that software programs can follow to communicate with each other. It serves as an interface between different software programs and facilitates their interaction, like the way the user interface facilitates interaction between humans and computers. APIs are extremely popular in the recent times as a pattern to overtake the interoperability limitations of the continuous creation and expansion of IT services. Other important driver of the API popularity is the creation of application-centric ecosystems, allowing third parties to create clients, use data and add value to the features of existing systems.

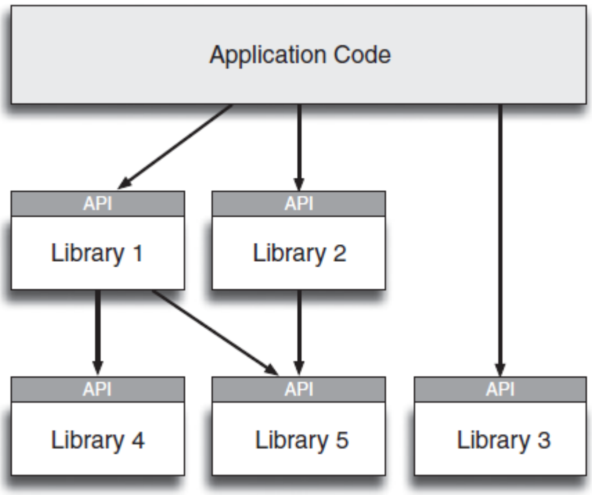


Figure 6 A representation of the relationship between libraries and APIs

In general, APIs are language dependent, since each interface is a contract specified in a given protocol. However, APIs can make use of communication standards or protocols (e.g. HTTP, SOAP, COAP...) to overtake this limitation and be programming language independent (though maintaining the syntax dependency with the selected protocol). Thus, the classification of the APIs can be done as follows:

- Language APIs: APIs that usually are part of a SDK and are composed by libraries and annotations to be invoked in a concrete programming language such as Java, .NET, Ruby, Javascript, Python, etc.
- Protocol APIs: These APIs, typically accessible within a network (mainly the Internet) and run on the hosting systems. They can also be referred as Web Services, since they are usually exposed through the Web to offer services to integrators and third parties. The W3C consortium defines Web Services as software systems designed to support an interoperable machine-to-machine interaction over a network. The three most common architectural styles on Web Services are:
- RPC (Remote Procedure Calls): RPC-based Web Services presents a call interface to distributed procedures and functions. Typically, the basic unit of this type of services is the WSDL operation (Service descriptor).
- SOAP (Simple Object Access Protocol): Web Services can be implemented following the concepts of the SOA (Service-oriented Architecture) architecture, where the basic unit of communication is the message, rather than the operation. This is typically referred to as message-oriented services.
- REST (REpresentation State Transfer): REST-based Web Services attempts to emulate the HTTP protocol or similar protocols by restricting the interface to a known set of standard operations (e.g. GET, PUT, etc.). Therefore, this style is more focused on interacting with resources with state, than with messages and operations.

1.3 Types of Frameworks and uses in INTER-IoT

For the sake of completeness of this document, a short study of different popular types of frameworks has been performed. The results are gathered in this section.

1.3.1 Agent-based frameworks

A software agent is a computer program that acts on behalf of a user or other program in a relationship of agency. Such "action on behalf of" implies a number of abilities that the agent need to possess: autonomous behaviour, social ability, responsiveness, proactiveness, and mobility.

Agents may be autonomous or, often, they interact with other agents. In the latter case the environment in which the agents operate is called Multi-agent system (MAS).

Software agent represent a very expressive paradigm for modelling dynamic distributed systems, and perfectly fit the generic and specific requirements of IoT systems. Indeed, agent-based modelling could provide even more benefits in the IoT domain than in conventional distributed environments.

A MAS is a middleware that basically provide an API for developing agent-based applications, and an agent server able to execute agents by providing them with basic services such as agent creation, execution, migration, communication, and access to resources.

A great variety of agent platforms have been to date developed, mostly Java-based. In the following three popular agent platforms will be briefly introduced: JADE, MAPS, and JIMAF. JADE is probably the most representative MAS; it is very powerful and general-purpose. JIMAF focus on enhanced agent interoperability support. Finally, MAPS is an agent platform specifically designed for supporting wireless sensor networks.

- JADE (Java Agent DEvelopment Framework) is an open-source software Framework fully implemented in the Java language. It simplifies the implementation of multi-agent systems through a middleware that complies with the FIPA specifications and through a set of graphical tools that support the debugging and deployment phases. A JADE-based system can be distributed across machines (which not even need to share the same OS) and the configuration can be controlled via a remote GUI. The configuration can be even changed at run-time by moving agents from one machine to another, as and when required. Besides the agent abstraction, JADE provides a simple yet powerful task execution and composition model, peer to peer agent communication based on the asynchronous message passing paradigm, a yellow pages service supporting publish subscribe discovery mechanism and many other advanced features that facilitates the development of a distributed system.

Thanks to the contribution of the LEAP project, ad hoc versions of JADE exist designed to deploy JADE agents transparently on different Java-oriented environments such as Android and J2ME devices.

- MAPS (Mobile Agent Platform for SunSPOT) is an innovative Java-based framework specifically developed on SunSPOT technology for enabling agent-oriented programming of WSN applications. It has been defined according to the following requirements:
    - Component-based lightweight agent server architecture to avoid heavy concurrency and agents cooperation models.
    - Lightweight agent architecture to efficiently execute and migrate agents.
    - Minimal core services involving agent migration, agent naming, agent communication, timing and sensor node resources access (sensors, actuators, flash memory, switches and battery).
    - Plug-in-based architecture extensions through which any other service can be defined in terms of one or more dynamically installable components implemented as single or cooperating (mobile) agent/s.
    - Java language for programming mobile agents.

The MAPS architecture is based on components that interact through events and offer a set of services to mobile agents including message transmission, agent creation, agent cloning, agent migration, timer handling, and easy access to the sensor node resources.

- JIMAF (Java-based Interoperable Mobile Agent Framework) focus on interoperability as a key issue for a wider adoption of Mobile Agent Systems (MASs) in heterogeneous and open distributed environments where agents, in order to fulfill their tasks, must interact with other non homogeneous agents and traverse different agent platforms to access remote resources.

  JIMAF relies on an event-driven, proxy-based mobile agent model. It supports interoperable mobile agents which can be easily coded and adapted to existing MASs without any modifications of the MAS infrastructures. An interoperable mobile agent consists of a MAS-neutral high-level part (programmed as an event-driven lightweight agent) which does not change throughout the mobile agent lifecycle, and a low-level part which depends on the specific MAS in which the mobile agent is operating and therefore changes after a heterogeneous migration, assuming the "shape" of the mobile agent of the new hosting MAS.

Communications among interoperable mobile agents are based on asynchronous messages, whereas communications between an interoperable mobile agent and a MAS-specific mobile agent are mediated by a wrapper agent. While execution and communication interoperability is directly furnished by the framework, migration of interoperable mobile agents between two heterogeneous MASs is enabled by MAS Bridges which are software equivalents of network bridges.

In the context of the Inter-IoT project, there are many “advantages” enabled (or favoured) using software agents. Agents can allow for:

- protocol encapsulation. In addition, in case of protocol upgrading, a new set of mobile agents can easily replace the old one at run-time,
- agents have natural orientation to heterogeneity. Mobile agents can act as wrappers among systems based on different hardware and software. This ability can well fit the need for integrating heterogeneous IoT devices based on different platforms. An agent may be able to translate requests coming from a system into specific suitable requests to submit to another different system,
- robustness and fault-tolerance. The ability of mobile agents to dynamically react to adverse situations and events (e.g. low battery level) can lead to more robust and fault tolerant IoT systems; e.g. the reaction to the low battery level event can trigger the migration of all executing agents to an equivalent device to continue their activity,
- devices and platforms configuration and management, enabled by agents migrating to the target device/platform or communicating with the local agent.

The use of agents and associated development frameworks can be found in the lower layers of INTER-IoT, such as the D2D level and in some parts of MW2MW. However, given the functional relation between INTER-FW and INTER-LAYER components (the former uses and aggregates functions of the latter), an extensive use of agent-based frameworks is not well suited for the INTER-FW, since it performs tasks of orchestration and encapsulation, while IoT interoperability-related business logic is frequently delegated to the lower layers.

1.3.2 Cloud based frameworks

The term "cloud based framework" is used to loosely describe collections of anything from development tools to middleware to database services that ease the creation, deployment, and management of cloud applications. Those that work at a software licensing and delivery model in which the software is hosted centrally are referred to as Software as a service (SaaS). Those that work at the level of servers, storage and networks are labelled as infrastructure-as-a-service (IaaS) frameworks. Platform as a service (PaaS) is a category between these 2 in which cloud computing services provide a platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app. PaaS can be delivered in two ways: as a public cloud service from a provider, where the consumer controls software deployment with minimal configuration options, and the provider provides the networks, servers, storage, OS, 'middleware' (i.e.; java runtime, .net runtime, integration, etc.), database and other services to host the consumer's application; or as a private service (software or appliance) inside the firewall, or as software deployed on a public infrastructure as a service.

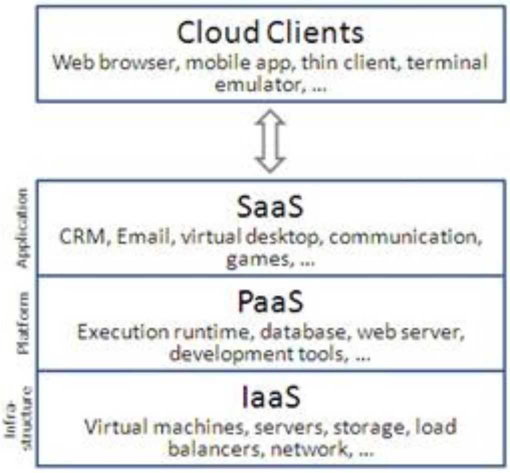

Cloud Clients
Web browser, mobile app, thin client, terminal emulator, ...
Application
SaaS
CRM, Email, virtual desktop, communication, games, ...
Platform
PaaS
Execution runtime, database, web server, development tools, ...
Infra-structure
IaaS
Virtual machines, servers, storage, load balancers, network, ...

Figure 7 Cloud-based frameworks classification

For the purposes of INTER-IoT, the platforms and infrastructure levels are relevant.

Representative implementations

The number of cloud based frameworks is extensive and in continuous growth[4]. The Eclipse IoT working group, IEEE IoT and AGILE IoT co-sponsored an online survey to understand how developers build IoT solutions[5]. 528 individuals participated between February 11 and March 25 2016.

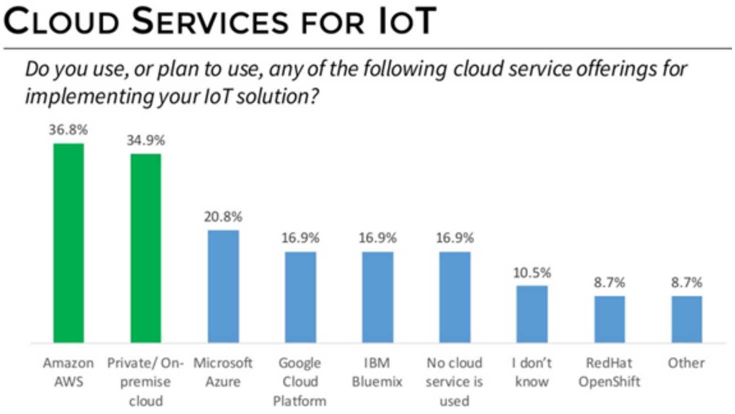


Figure 8 IoT online survey Source: IEEE IoT

[4] https://en.wikipedia.org/wiki/Category:Cloud_platforms

[5] http://iot.ieee.org/images/files/pdf/iot-developer-survey-2016-report-final.pdf

In the following lines, some examples of popular cloud based frameworks are briefly commented.

Amazon Web Services IoT[6]

Launched in late 2015, AWS IoT allows companies to use Amazon Web Services to meet IoT needs by combining device communications, security measures, and a number of versatile application program interfaces (APIs) with Amazon's robust AWS cloud. Essentially, AWS IoT acts as a device gateway that facilitates messaging between IoT devices and the cloud. Amazon supplies clients with a device SDK that they can use to connect their devices to the gateway and communicate with the cloud using MQTT or HTTPS. The platform also creates what Amazon dubs a "device shadow," a persistent virtual representation of each individual device connected to the platform that takes note of metadata and the kinds of information the device can generate. Users are able to create rules that will send messages to other AWS services and prompt automated actions. AWS IoT's main strength is the scale at which it operates within Amazon's overall cloud portfolio and the additional services that it can leverage from that existing stable.

Predix[7]

GE developed the open-source platform for both internal and external industrial applications. Users can connect industrial machinery to data-collection devices, which then process and transmit data at the edge or transmit data directly to a cloud server for processing and analysis. Predix includes authorization and authentication protocols at both the device and application level for security, which is integral in the industrial settings where it is used. Users are able to run analysis algorithms in order to parse and read the data. They can also develop, deploy, and manage applications and services based around data collected through the industrial platform, allowing for highly customizable uses and various forms of automation. Additionally, users can create mobile apps to serve as dashboards, with which they can tap into the data outputs in real time and view data based upon their particular IoT setup and needs — like monitoring equipment or output levels, for example.

Azure IoT[8]

With a number of premade solutions that can be specifically altered to meet a company's needs, Microsoft's Azure IoT Suite enables companies to implement a cloud-based IoT solution with minimal development. The suite offers device SDKs to connect devices to the cloud and transmit operational data and statistics. The platform builds a database and allows users to engage in real-time analytic observations through SQL queries. Users can access an existing algorithm library to draw on others' experiences and perform more informed predictive analysis. Azure also offers extensive device authentication security measures.

Bluemix[9]

A versatile and modular platform, IBM's Bluemix allows companies interested in IoT deployments to employ a fully cloud-hosted, managed IoT platform. Bluemix can connect to existing devices or gateways through either the MQTT or HTTP protocols and transmit data in real time from a remote site to the cloud. It creates independent, scalable runtimes for each application on its cloud servers that can be accessed through a number of APIs, and leverages IBM's Watson AI computing system to engage in real-time analytics. IBM also allows users to create their own applications within Bluemix to communicate information from the cloud to relevant users via a dashboard and display the analytic findings.

[6] https://aws.amazon.com/iot

[7] https://www.ge.com/digital/predix

[8] https://azure.microsoft.com/suites/iot-suite/

[9] https://www.ibm.com/cloud-computing/bluemix/internet-of-things

The knowledge and use of cloud-based frameworks in INTER-IoT has a twofold purpose:

- Integrate systems based on these cloud-based frameworks with the INTER-LAYER infrastructure.
- Build a cloud-based framework to manage the different layers and the platform nodes connected to them.

1.3.3 REST-based frameworks

The philosophy of REST[10] is based on the concept of resources and idea of using the inherent power of HTTP to manipulate and retrieve representations of these resources in varying states. Every REST resource is identified uniquely by an URL, which is operated upon by a subset of HTTP commands: get, post, put and delete. This is a major advantage of REST, as HTTP is the universal protocol of communication over the internet. Another advantage of REST is that it does not bloat messages sent over the internet -- that is, it does not add additional headers apart from standard HTML header already contained in the message. For example, to execute a delete command upon a resource, one must only embed the ID of the resource to the HTTP header, containing the HTTP delete command.

REST architectural approach is scalable, while in performance benchmarks it is shown to have linear time complexity regarding running more and more concurrent requests from clients. Its architecture is based on the client-server model, it is stateless and cacheable.

REST approach is most suitable for cases when one needs to manage hierarchically organized resources, uniquely identified with resource identifiers. This is commonly the case, when one's problem naturally translates to trees and it can be naturally and easily presented with URI paths, while majority of system's operations can be expressed with simple query parameters. For problems that don't conform well to this requirement, performing even simple operations can necessitate usage of many query parameters, while also including the body of the query into the HTTP[11] command body.

The difference between protocols such as SOAP and REST, which is an architectural style, is that in SOAP, changes are made by accessing server objects remotely, while REST focuses on representing server objects on the client side. From this fact REST derives its scalability – when we need to make changes to the server side, we simply enhance representations, which will only be a concern of the client side, while the API itself (control methods) are left untouched. Pure REST never manipulates objects on the server side.

Regarding security, we can secure REST by using HTTPS instead of HTTP.

REST implementation examples

A representative implementation of a REST-based framework is the Portal framework [1], which serves as a bridge between abstract program actions and concrete device inputs, thus creating kind of a middleware layer. It naturally maps into the client-server paradigm, employing the server side to handle all registered interaction techniques, application, device and user profiles, while enabling the client to deal primarily with applications that utilize the framework. Its architecture of interconnected components (plug-in manager, controller, data transmission component, data model component and data flow component) also neatly integrates into REST's preferred hierarchical

[10] http://restfulapi.net/

[11] http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

world-view. It was implemented both using REST and using SOAP. The REST implementation was found to be more scalable, as well as less complex.

Second representative implementation of a REST-based framework is Livy[12], which is a service that enables easy interaction with an Apache Spark cluster over a REST interface. With it one can submit Spark jobs or snippets of Spark code, one can retrieve results synchronously or asynchronously, and also manage SparkContext. A session is emulated through usage of a session REST object, which contains application's session id, owner id, state of the session and so forth. There are also REST objects for a statement (result of an execution statement) and a batch object.

Django REST framework[13] is a third representative of REST-based frameworks. It is a toolkit for building web-based APIs, which are self-describing, list API endpoints, describes allowable operations on each and presents them as hypermedia controls that it sends in responses.

REST in INTER-IoT

As described above, this approach is suitable for the INTER-IoT project and is the main mechanism to expose the layers APIs and the global API of the framework.

As REST philosophy builds upon representations being completely at the mercy of the client, INTER-FW would not become bloated with addition of new platform types, new devices, new brokers, security features, etc. Existing nodes, such as /platforms/1/devices/5 could easily be not only accessed, but also managed.

1.3.4 Reactive streams based frameworks

The concept of a reactive stream refers to handling of (possibly unbounded) sequence of data in a "reactive way". According to the Reactive Manifesto[14], a reactive system has to be:

- responsive: responds in a timely manner to requests,
- resilient: stays responsive even in the face of failure,
- elastic: stays responsive even under heavy load, and
- message-driven: ensures loose coupling between components by use of asynchronous message-passing.

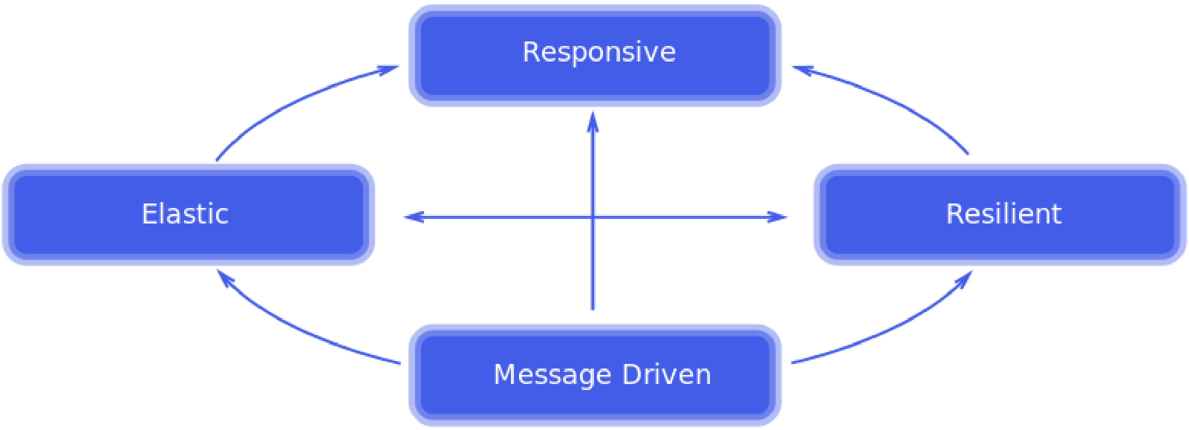


Figure 9 The four attributes of a reactive system

[12] http://livy.incubator.apache.org./

[13] http://www.django-rest-framework.org/

[14] http://www.reactivemanifesto.org/

In the context of data-stream handling/processing the reactivity principle essentially boils down to the ability to establish and maintain a non-blocking data flow between (possibly distributed) system components or systems. The Reactive Streams initiative standard defines a set of interfaces for handling streams in a reactive manner and gives a detailed specification of their intended behaviour. The main idea behind is the “back-pressure” mechanism, based on the Publish-Subscribe model. The Reactive Streams (RS) standard offers the following interfaces: Publisher, Subscriber, Subscription, and Processor.

- Publisher<T> is the source of data for Subscriber(s). It offers a possibly unbounded sequence of data elements of type T.
- Subscriber<T> is an entity responsible for managing the subscription to a Publisher and handling the incoming data of type T. It also copes with error conditions and subscription cancellation.
- Subscription represents the lifecycle of a Subscriber subscribing to a Publisher.
- Processor<T,R> represents a reactive stream processing stage which is a Subscriber<T> and, at the same time a Publisher<R>.

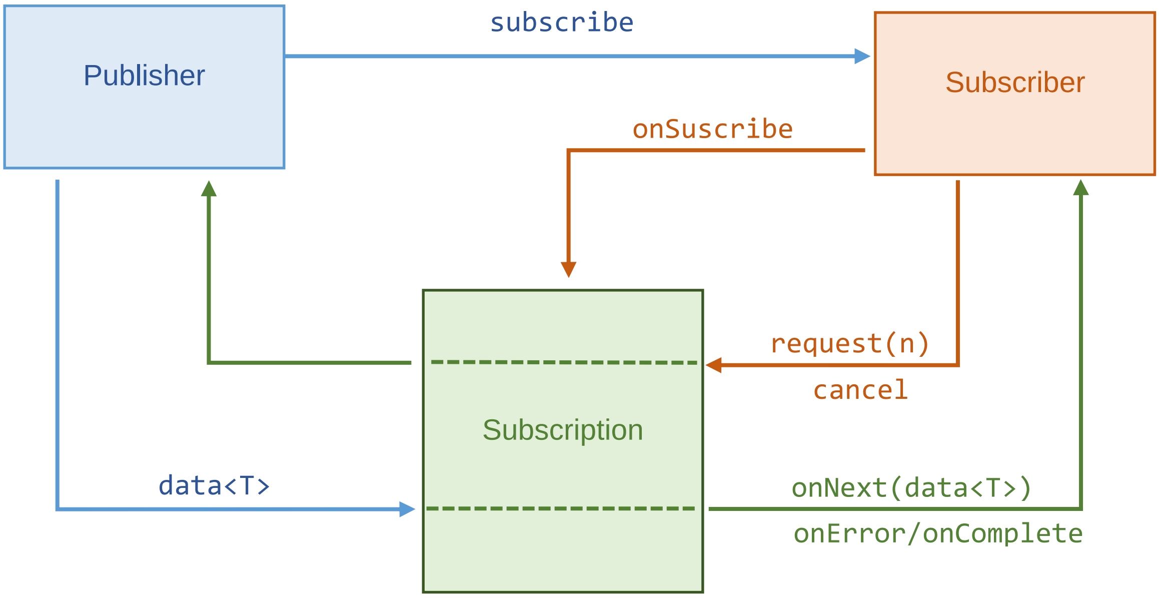


Figure 10 Main interfaces in reactive streams

The RS standard specification, although presented in the form of just four simple interfaces turns out to be quite complex and demanding at the implementation level. The treatment of asynchronous communication combined with back-pressure and proper error handling mechanisms is not as simple as it may seem at first. Fortunately, the existing libraries and frameworks that support the RS standard offer DSLs which make creating and transforming reactive streams much easier and intuitive.

Some examples of RS frameworks are listed in the following lines:

RxJava

RxJava[15] is a lightweight (single-jar) library supporting reactive programming in Java. It offers and advocates functional programming techniques, including in-mutability and statelessness. RxJava is essentially a JVM implementation of the Reactive Extensions (ReactiveX), which builds upon the standard Observer pattern.

Even though, Netflix participated in the Reactive Streams initiative from the very beginning, the RxJava 1.* implementation of streams does not directly comply with the RS standard. Interoperability between RS implementations and RxJava 1.* can be achieved with the help of RxJavaReactiveStreams module, though. The development of RxJava 2.* took a long time, but the second generation of the project has recently been published, offering full implementation of the RS standard.

At the moment, the development of the library seems to concentrate mainly on the Android platform.

Reactor

Reactor Core 3[16], similarly to RxJava, is a small (single-jar), highly focused library directly implementing the Reactive Streams standard, adding many useful stream transformation-building mechanisms. There are two types of Publisher(s) in Reactor Core 3: a (0..N) Flux<T>, and (0|1) Mono<T>. The library requires Java 8 to operate. Prior to the version 3.0, Reactor was using RxJava for the stream processing, and wasn't RS standard compliant.

An interesting feature of version 3 is that it also offers support for the native implementation of Reactive Streams coming in Java 9 (Flow API).

The Reactor Core library is a core dependency for the coming Spring Framework 5, in particular, offering RS capabilities through the Spring Data 5 module.

Akka streams

According to the Akka streams project webpage[17] "Akka is a toolkit and runtime for building highly concurrent, distributed, and resilient message-driven applications on the JVM". Akka utilizes the Actor Model of concurrency.

An actor is a lightweight primitive, able to communicate asynchronously with other actors in the system. Upon receiving a message an actor can react by sending a finite number of messages, spawning a finite number of new actors, and changing its own internal state. The model assumes no shared global state.

With actors (both local and remote) as basic building blocks Akka offers a wide range of modules supporting construction of distributed systems. In particular the module akka-stream provides a large set of high-level tools for defining, constructing and managing reactive streams. The types Source<T,U>, Flow<S,T>, and Sink<T,U> are built on top of the Reactive Streams standard types Publisher<T>, Processor<S,T>, and Subscriber<T>. The "extra type parameter" U provides a way of handling any auxiliary information that might be generated in addition to the data flow within the stream.

Akka offers high-level DSLs for defining blueprints for streams, and a rich set of predefined transformations on streams.

[15] https://github.com/ReactiveX/RxJava

[16] https://github.com/reactor/reactor-core

[17] http://doc.akka.io/docs/akka/2.5.3/scala/stream/index.html

Use in INTER-IoT

Concepts of “stream of data” and “stream processing” seem very natural and fundamental for any Internet of Things application. Also, the principles of “reactivity” undeniably apply to the realm of IoT. The popularity/acceptance of the Reactive Streams standard increases, and will probably continue to do so in the future. Solutions for many popular data sources/stores are available at the moment and new ones are being created. Hence, a framework offering Reactive Streams as a tool/concept is well worth considering in the case of INTER-IoT.

For a successful application within the INTER-FW, however, a framework definitely needs to provide more than just plain Reactive Streams support. Any INTER-IoT deployment will surely be distributed and will run on a (cloud) cluster. Therefore, the INTER-FW should also provide some flexible, high-level mechanisms for defining/composing such deployments. A carefully designed DSL would probably be very helpful in this respect.

Both Reactor and Akka Streams constitute a part of comprehensive frameworks – Spring Framework and Akka respectively. RxJava 2.*, on the other hand, is a targeted library offering “just” an implementation of Reactive Streams standard, together with some tooling.

Taking the above into account, we should probably narrow our choice to Reactor vs Akka Streams, or rather Spring vs Akka. Both frameworks provide comprehensive sets of tools for building distributed applications, and support microservice architecture. Spring has an undeniably longer history, but Akka is definitely a mature project as well. Both frameworks have vibrant user communities, and are actively maintained. Out of the two Akka seems to be more coherent and compact in design. It also offers high-level DSLs which INTER-FW might successfully utilize/extend.

Some of the principles of reactive programming are successfully used in the design and implementation of MW2MW layer (Intermw), while not following a specific framework.

1.3.5 SOA-based frameworks

The concept of Service-Oriented Architecture is a software design where services can be accessed by other software components using a communication protocol over the network. These services implement protocols that describe how to send and parse message using description metadata. A service can be defined as a software unit piece with a functionality that can be accesses remotely and acted upon and updated independently, these services have four main properties:

1. It logically represents a business activity with a specified outcome.
2. It is self-contained.
3. It is a black box for its consumers
4. It may consist of other underlying services.

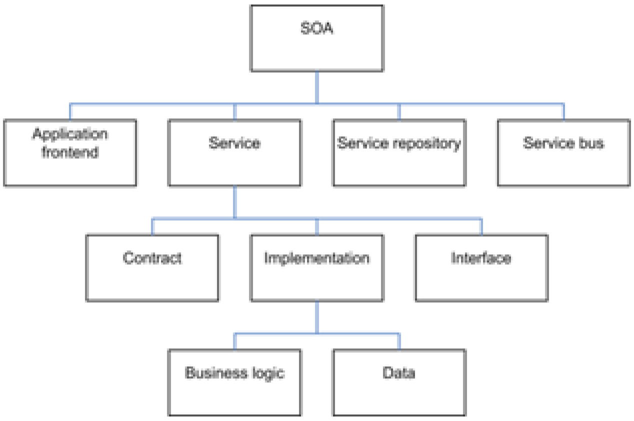


Figure 11 Example of service oriented architecture based framework

The SO-Architecture is more related on how to compose an application by integration of distributed, separately-maintained and deployed software components. It is enabled by technologies and standards that make it easier for components to communicate and cooperate over a network, especially an IP network. These technologies include:

- Web services (based on WSDL[18] and SOAP[19])
- Messaging, (e.g. ActiveMQ, JMS, RabbitMQ)
- RESTful HTTP, with REST[20].
- OPC-UA[21]
- WCF (Microsoft's implementation of Web services, part of WCF[22])
- Apache Thrift[23]
- SORCER[24]

But if we want to apply this software design into the implementation of a framework, we can use this architecture as a base and cover it with a different tools and run-time infrastructure software. Hence, we obtain a SOA architecture implementation framework or SOAIF. This is composed by all the technologies that an enterprise might need to build and run a SOA, including design-time and run-time capabilities as well as the software functionality with: tools, management, integration, modelling, security, and processes.

This frameworks act as an infrastructure platform that lets developers and business analyst create, deploy, manage and change processes within and across the enterprise. But these frameworks have very specific requirements as:

- distributed event-enabled architecture,
- flexibility via service-enabled processes,
- enterprise standards support (fault tolerance, reliability, and scalability),
- security in distributed environment,
- visual process composition and monitoring,

[18] https://www.w3.org/TR/ws-arch/#WSDL
[19] https://www.w3.org/TR/ws-arch/#SOAP
[20] https://www.w3.org/2001/sw/wiki/REST
[21] https://opcfoundation.org/developer-tools/specifications-unified-architecture
[22] https://msdn.microsoft.com/en-us/library/hh128109.aspx
[23] https://thrift.apache.org/
[24] http://sorcersoft.org/project/site/

- and rapid process changes.

Representative Implementations

Java OSGi[25] technology is the most representative specification to implement software frameworks and application in a service-oriented architecture. Some of the most representative implementation examples are:

- Equinox[26], Concierge[27], Apache Felix[28] or Karaf[29] as runtime framework and service platforms OSGi-based on a module design that allows developers to implement an application in a bundles structure using the common services infrastructure they provide.
- Swordfish[30], is an open source SOA framework intended for applications ranging from enterprise environments to embedded systems. Its features as a SOA runtime platform that leverages three popular projects: Service Component Architecture (SCA) as common programming model and assembly description format, Java Business Integration (JBI) as a common messaging model, and Open Services Gateway initiative (OSGi) as the basis of the runtime platform.
- WSO2 Carbon[31] the core platform of the middleware, also based in OSGi, that allows components to be dynamically installed, started, stopped, updated and uninstalled, and it eliminates component version conflicts, creating a service oriented structure for an enterprise middleware.

SOA in INTER-IoT

INTER-FW and some interoperability layers (gateway, intermw) implementations are partially or completely distributed using services for each functionality of the framework and communicating each other by communication protocols. This is useful in the scalability and extendibility of the framework due to the modularity that this architecture provides.

It also presents some caveats, as for instance that the components need to rely on the underlying network, so this has to be strongly configured for resilience.

As INTER-FW has to access all the components of Inter layer, INTER-FW can implement different services to be consumed for each one of the clients on each interoperability solution. So that each tier of INTER-LAYER expose an API (e.g REST API) to connect with specific services within the framework where an added value is also implemented.

[25] https://www.osgi.org

[26] http://www.eclipse.org/equinox

[27] http://www.eclipse.org/concierge

[28] http://www.felix.apache.org

[29] http://www.karaf.apache.org

[30] https://wiki.eclipse.org/Swordfish_Documentation:_Architecture:_Interceptor_Framework

[31] http://wso2.com/products/carbon

This added value is an environment where future developer can implement a new service included within INTER-FW to be used by one or more interoperability solutions. So that, the modularity that Service Oriented Architecture provides makes easier this extension.

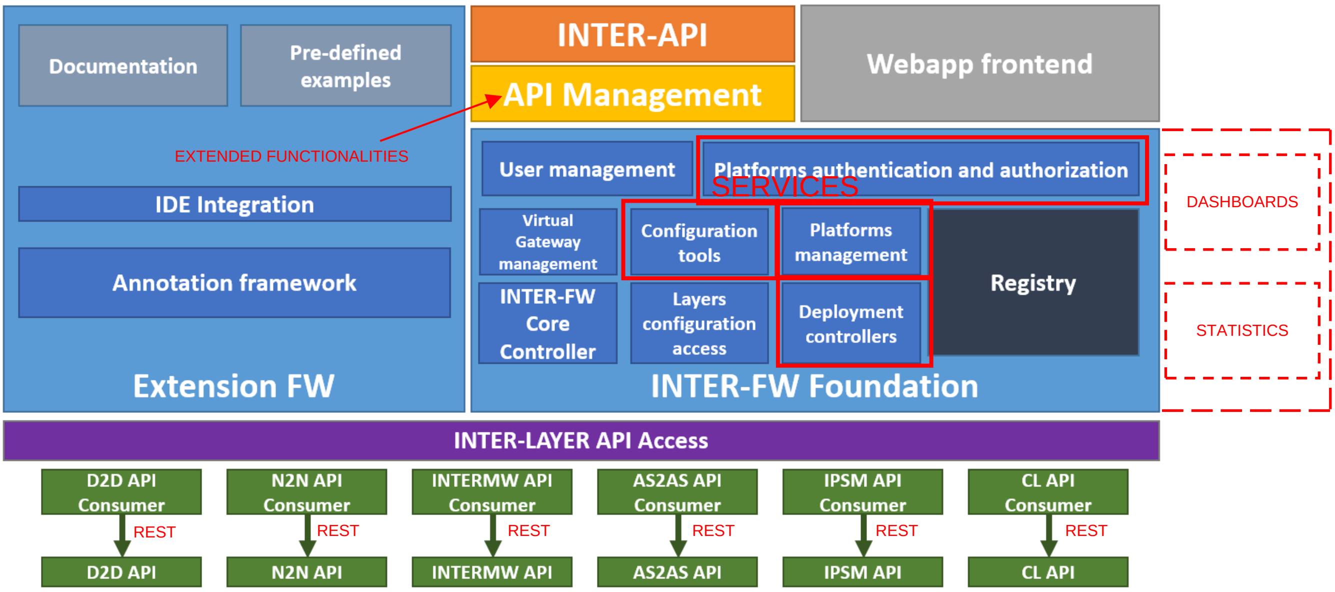


Figure 12 SOA Aspects in INTER-IOT

2 INTER-FW Analysis

2.1 Analysis overview

The analysis of INTER-FW has been a process to transform user and technical requirements gathered mainly in WP2 activities and consolidated with WP3/WP4 activities into technical specifications to enable the implementation of all the components composing the framework.

Initially, all requirements affecting INTER-FW and INTER-API were extracted and analysed, considering the initial version of the meta-model and meta-architecture (D4.1) and the intial results of the INTER-LAYER (D3.1). After this, all the use cases were extracted according to the features identified, and analyzed in a structured document by all the partners participating in the task. Inputs from WP3/WP5 task leaders were also received to allow a good integration of INTER-FW with other ongoing developments and to design the set of tools coherently to the rest of technical tasks.

After this, a mock-up of the front-end was also developed in collaboration of all the partners. The mock-up is available online in the INTER-IoT repository (https://git.inter-iot.eu/Inter-IoT/framework/src/master/design). Once the mock-up was validated by partners and stakeholders, an analysis of the backend was made, including the security aspects of the web application and the credentials brokerage needed to manage the security and privacy of the connected platforms. UML Class diagrams have been used as supporting tool to depict the different models in the INTER-FW web application and tooling.

These design documents are available in the project git, currently accessible only to project partners and open call winners. The following sections gather the results of these activities.

2.2 Use cases and sequence diagrams

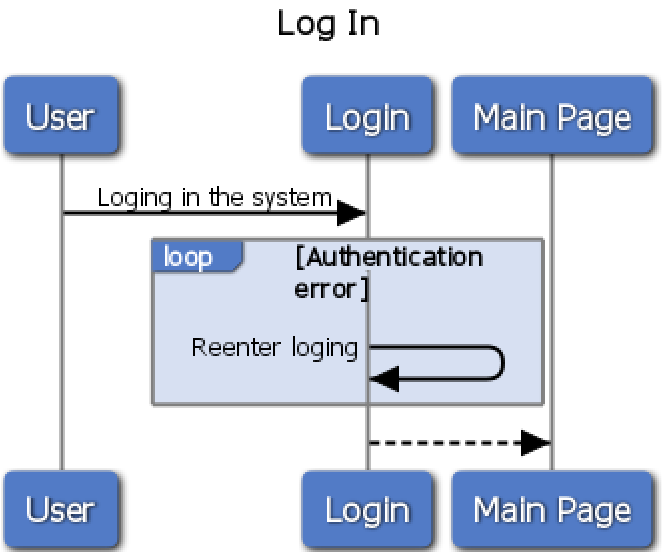
In the analysis of the INTER-FW, all the functionalities and/or use cases extracted from the requirements have been identified and analysed in depth before beginning any development. Use cases are described as a story that facilitates the identification of the users, their motivation, and the action they take to solve a problem. The template used to describe this history is the following [2] (agile user story template):

As a <role of the user, e.g. INTER-IoT Third Party Developer>
I want to <user motivation, e.g. add a device to an existing gateway>
So that <goal/benefit/value e.g. I can interoperate my device with other existing in my office>

Finally, a sequence diagram has been made for each of the use cases of identified, in which you can see the different actions that must be carried out among the proposed menus.

2.2.1 Log In

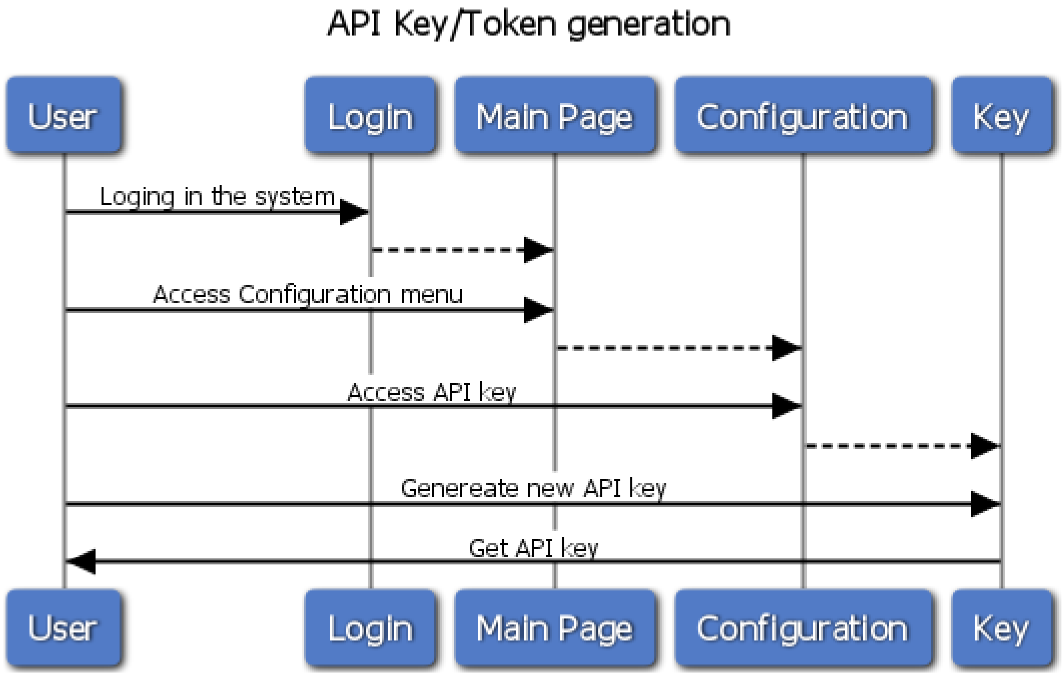
As an IoT platform administrator, I want to authenticate myself on the INTER-IoT environment, so that I'm able to exploit the available platform functionalities according to my permissions.



Preconditions	User is already registered, so he has INTER-IoT credentials (user & password).
Steps	1. Go to the page containing the authentication module 2. Insert user credentials (a link is available for unregistered users, which are re-direct to a registration form) 3. Confirm the operation
Verification	Credentials match and user is redirected to a customized page (in case of unsuccessful login, user is invited to re-insert credentials or to start the password recovery procedure -- a new password is sent to his/her registered email address. After the fifth unsuccessful attempt, login functionality is temporarily disabled for security reasons, user have to wait 30 minutes and portal administrator is notified through an email).
Postconditions	User can operate accordingly to his/her permissions.

2.2.2 API/Token generation

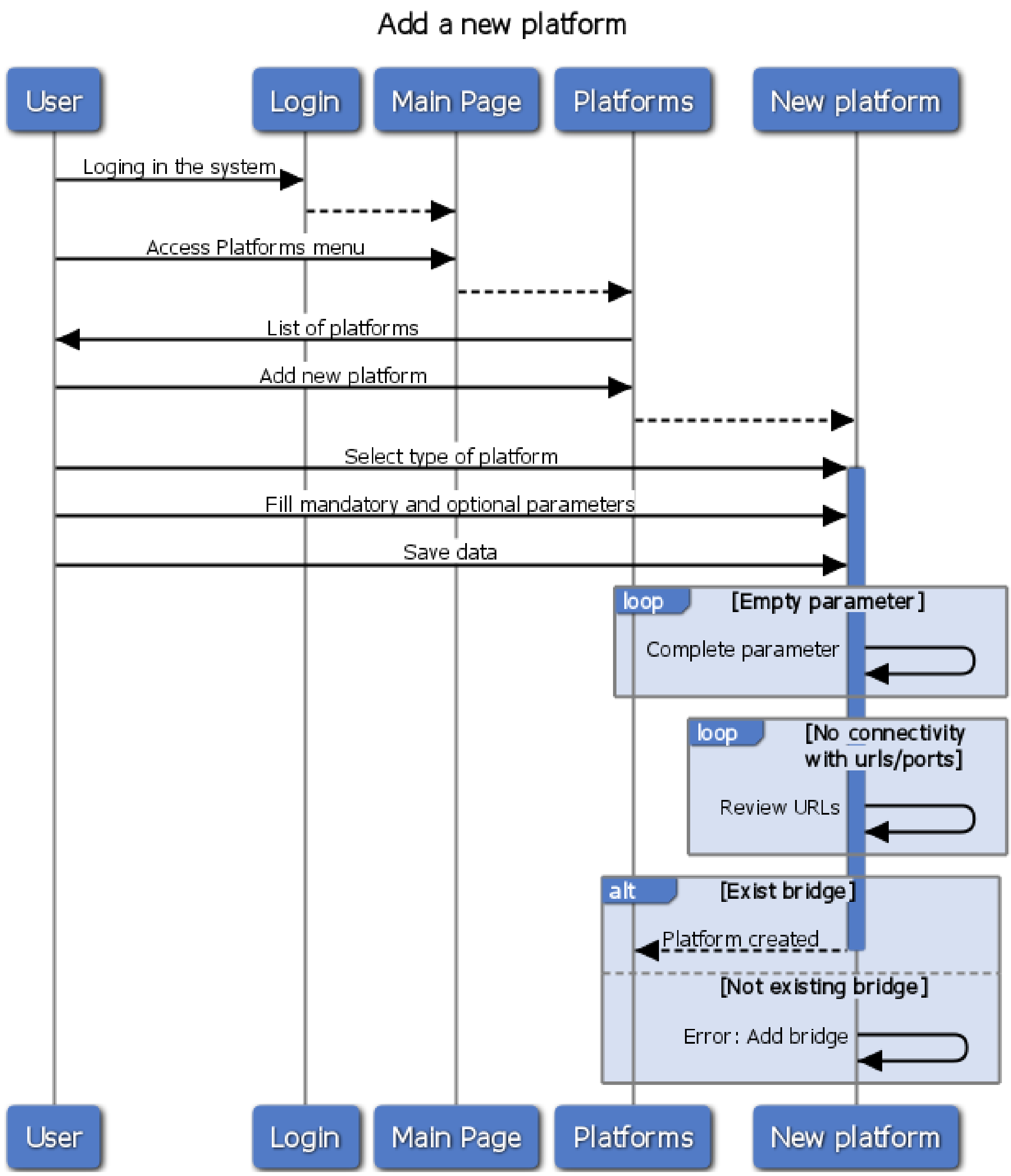
As an IoT integrator, I want a Rest API to integrate my services in the INTER-IoT environment so that the services can interoperate with other IoT platforms.



Preconditions	-
Steps	1. Login in the system 2. Access to the configuration menu. Use the old key/token to confirm your privileges 3. Ask for a new key/token 4. Copy the new token
Verification	-
Postconditions	The new key/token can be used to access the APIs.

2.2.3 Add a new platform

As an IoT platform administrator, I want to add my platform to the INTER-IoT environment, so that the platform can interoperate with other platforms.

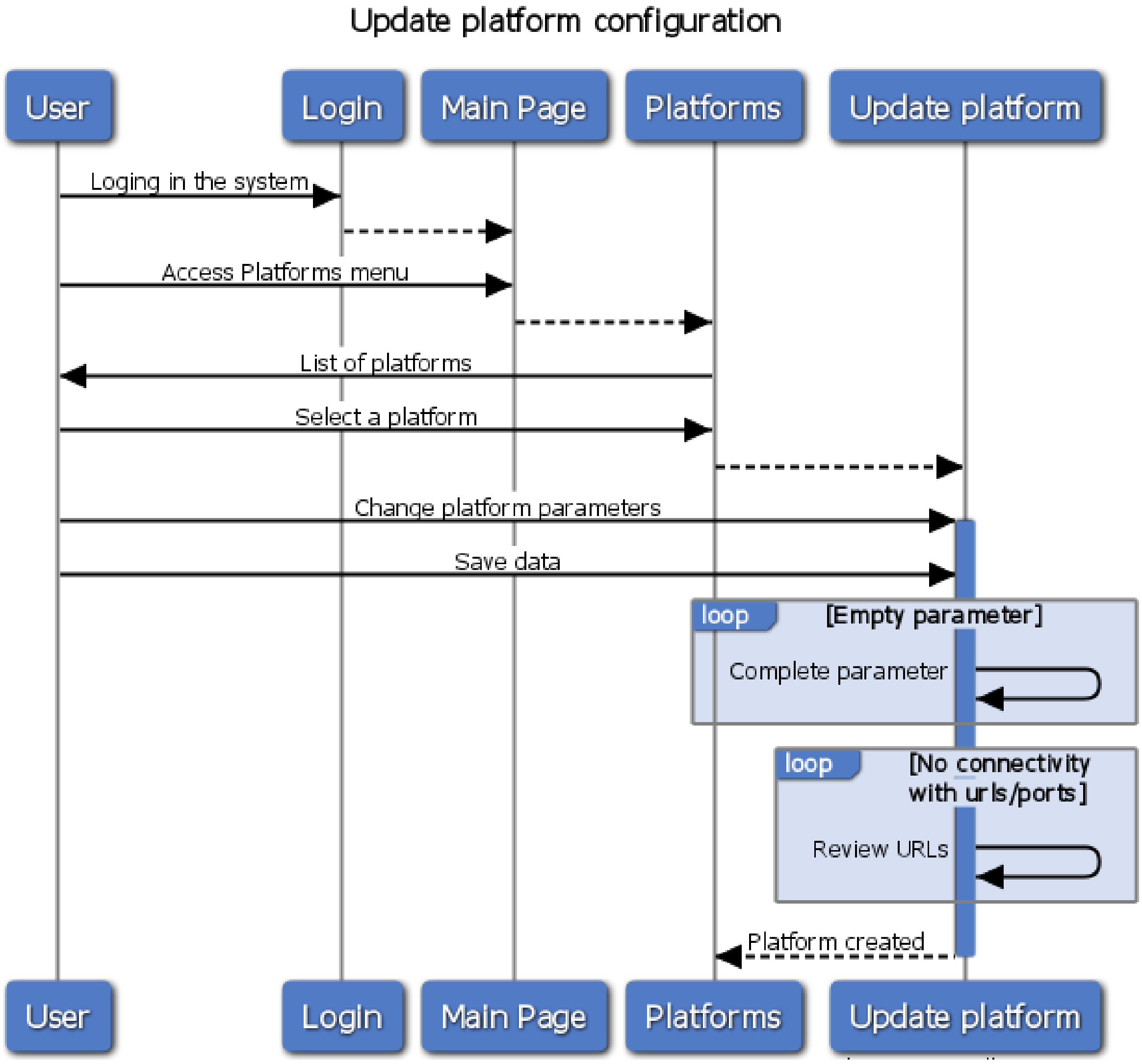


Preconditions	-
Steps	1. Login in the system 2. Access to the platform menu with the list of all the registered platforms 3. Select the button to create a new platform 4. Complete the mandatory and optional fields 5. Save the data
Verification	If any of the mandatory fields are not complete, the system requires to fill it in. The urls and ports are checked when the platform is created. If any of them has not connectivity, the system requires changing it. After the platform is created, it need a proper bridge to interoperate. If it does not exist a bridge for this kind of platform, it is necessary to include it.

Postconditions	The new platform can be used from the webapp

2.2.4 Update platform configuration

As an IoT platform administrator, I want to change my platform configuration, so that the new parameters are updated.

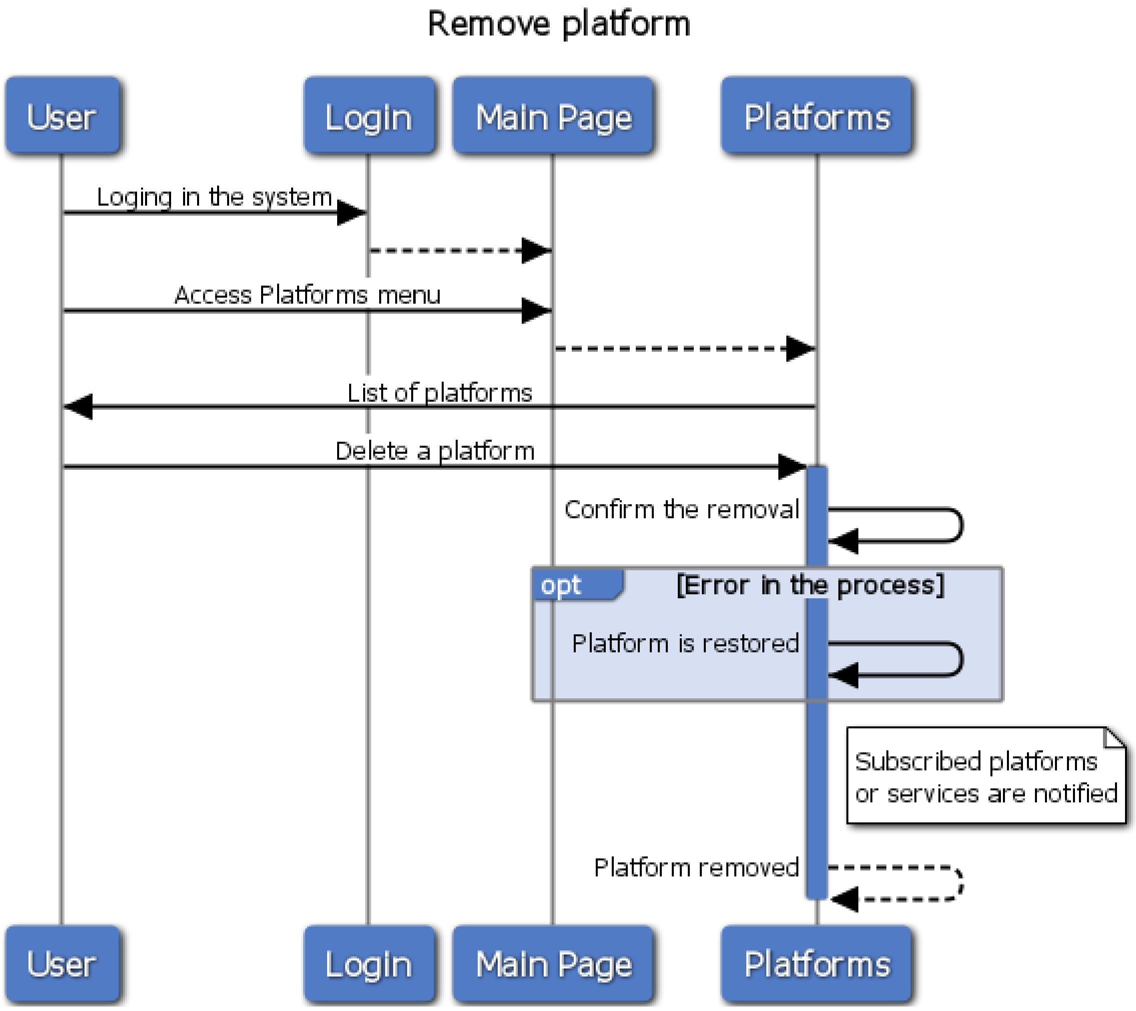


Preconditions	The platform is already registered in the system
Steps	1. Login in the system 2. Access to the platform menu with the list of all the registered platforms 3. Search the platform you want to update in the list and select the update button 4. Update all the needed fields 5. Save the data
Verification	If any of the mandatory fields are not complete, the system requires to fill it in. The urls and ports are checked when the platform is created. If any of them has not connectivity, the system requires changing it.

Postconditions	The updated platform can be used from the webapp

2.2.5 Remove a platform

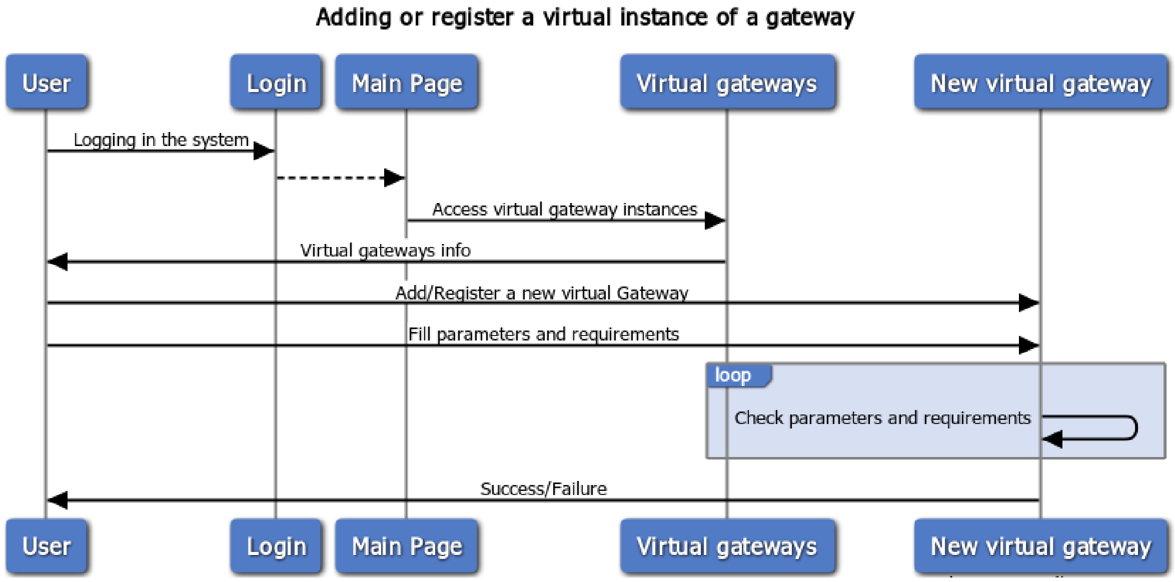
As an IoT platform administrator, I want to remove my platform from the INTER-IoT environment, so that the data is now confidential.



Preconditions	The platform is already registered in the system
Steps	1. Login in the system 2. Access to the platform menu with the list of all the registered platforms 3. Search the platform you want to update in the list and select the remove button 4. Confirm that you are sure to remove the platform
Verification	If there is any problem during the removal process, the platform is restored. All the platform or devices subscribed to data of the platform are notified of its removal
Postconditions	The platform cannot be used from the webapp anymore

2.2.6 Add or register a virtual instance of a gateway

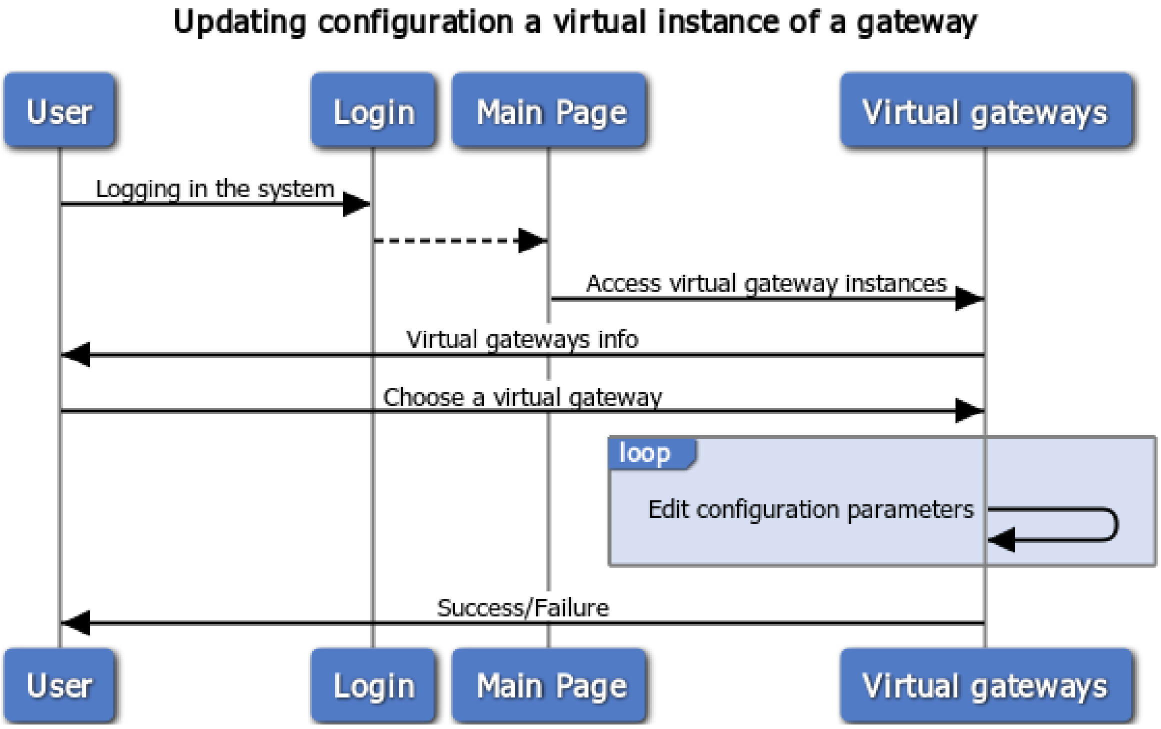
As a user who desires to install the INTER-IoT Gateway solution in my small-scale deployment (e.g. devices from my smart home) to access its information from the same User Interface. In such case, the data from my Bluetooth devices and data from Wi-Fi devices can be stored in the same database and accessed from the same UI (that could be a GUI).


Adding or register a virtual instance of a gateway
User
Login
Main Page
Virtual gateways
New virtual gateway
Logging in the system
Access virtual gateway instances
Virtual gateways info
Add/Register a new virtual Gateway
Fill parameters and requirements
loop
Check parameters and requirements
Success/Failure
User
Login
Main Page
Virtual gateways
New virtual gateway

Preconditions	-
Steps	1. User logs in and access virtual gateway page 2. User adds virtual gateway and fills requirements
Verification	User is notified upon gateway registration, whether it is successful of not.
Postconditions	If the operation was successful, the user can use the newly added gateway. Else, the state of the system remains unchanged.

2.2.7 Update configuration of a virtual instance of a gateway

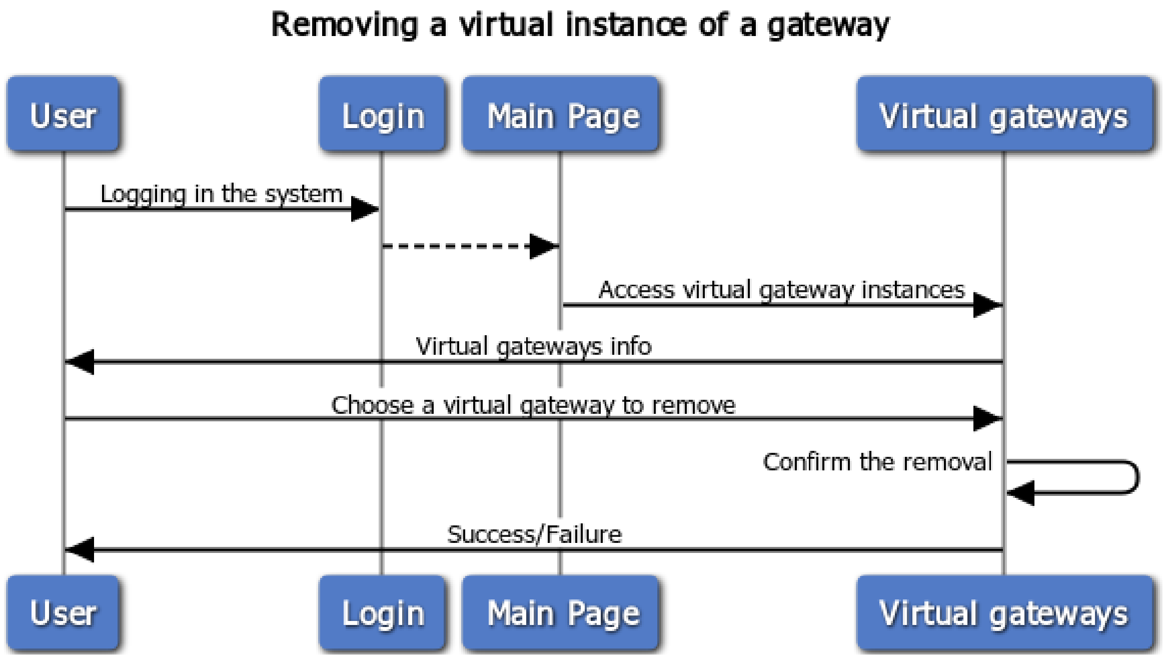
As a user who desires to configure the INTER-IoT Gateway solution in my small scale deployment (e.g. devices from my smart home) to access its information from the same User Interface. I want to custom configure the parameters of my gateway (virtual instance). So that I can change some parameters such as the size of the storage buffer of data, the time slot for pooling information, choose the platform to configure, etc.



Preconditions	Virtual instance of the gateway is in the system
Steps	1. User logs in and access virtual gateway page 2. User chooses a virtual gateway instance 3. User modifies virtual gateway parameters
Verification	User is notified upon gateway modification, whether it is successful of not.
Postconditions	If the operation was successful, the user can use the changed gateway. Else, the state of the system remains unchanged.

2.2.8 Remove a virtual instance of a gateway

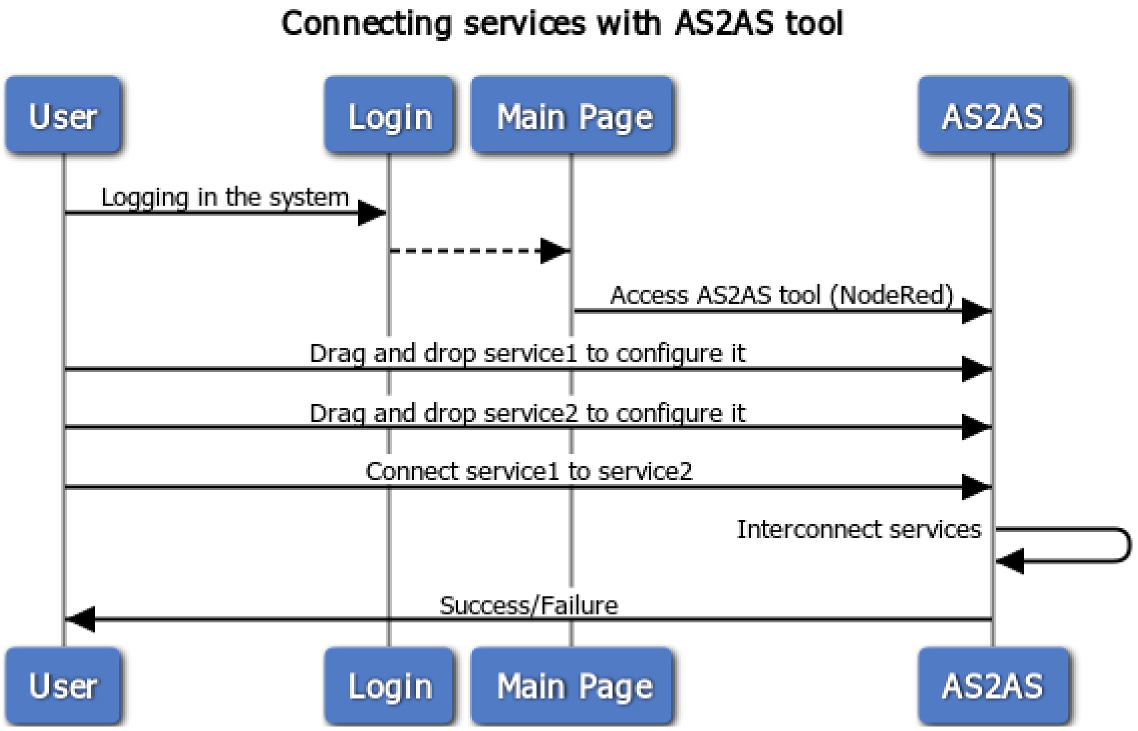
As a User with a deployed INTER-IoT GW solution I want to remove an instance of the virtual GW to create another one or to eliminate the deployment of my system. So that, I can reinstall it in somewhere else or just remove it to create a fresh new instance of the virtual GW.



Preconditions	Virtual instance of the gateway is in the system, and is not in use.
Steps	1. User logs in and access virtual gateway page 2. User chooses a virtual gateway instance 3. User removes virtual gateway
Verification	User is notified upon gateway removal, whether it is successful of not.
Postconditions	If the operation was successful, the user cannot use the removed gateway. Else, the state of the system remains unchanged.

2.2.9 Connect services with AS2AS tool

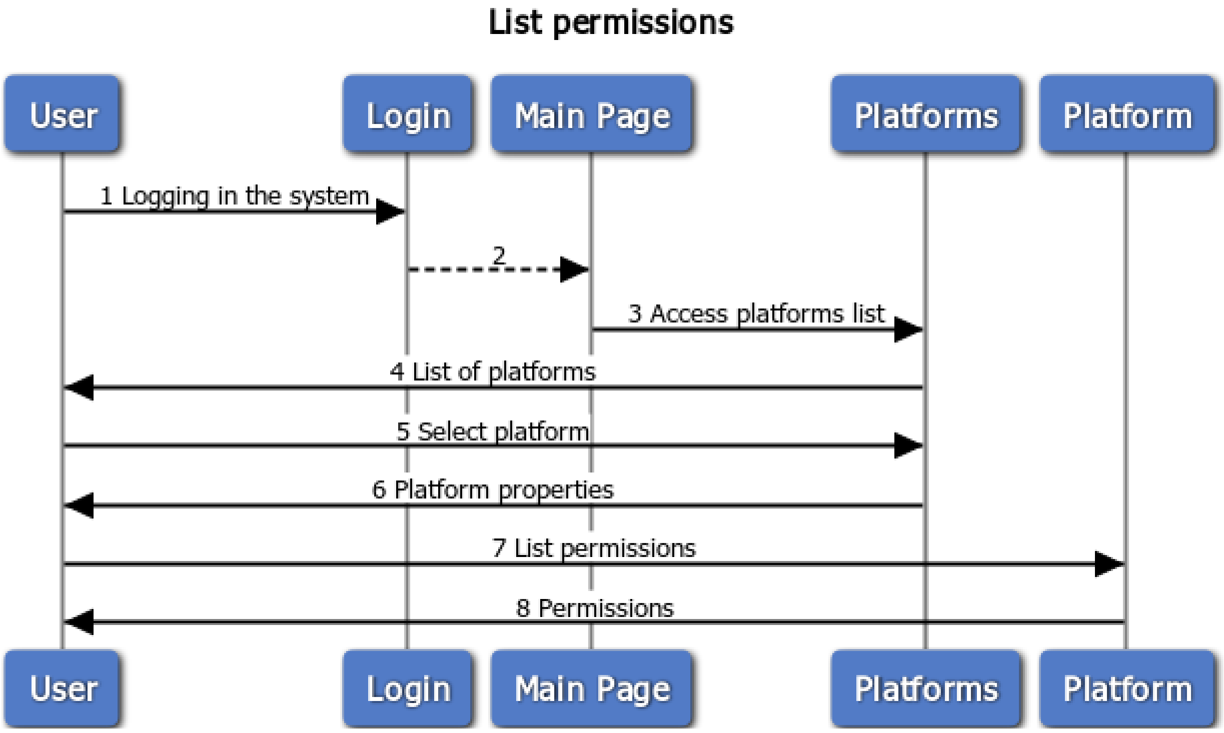
As a user who has two or more IoT platforms deployment, I want to connect two services from these different platforms, so that I can create a composite service using the output of one service as the input for the other.


Connecting services with AS2AS tool
User
Login
Main Page
AS2AS
Logging in the system
Access AS2AS tool (NodeRed)
Drag and drop service1 to configure it
Drag and drop service2 to configure it
Connect service1 to service2
Interconnect services
Success/Failure
User
Login
Main Page
AS2AS

Preconditions	The deployed environment with the AS2AS solution installed and running the INTER-IoT Node-RED version. To have installed the nodes that provide the access to the desired services. To have installed the nodes that facilitate the conversion of formats/protocols from the output of one of them to the input of the other. To have IoT Platforms configured
Steps	1. As INTER-FW portal has embedded the INTER-IoT AS2AS solution, you must access this portal, with the authentication access rights. 2. Then you drag and drop the first node (service) and complete the configuration parameters needed to access this service. 3. Later, you drag and drop the second service and configure it as well. 4. Then, you can drag and drop a function node to connect the first service to the second. a. If the code of the connector is not already implemented, the user should write a short function to extract the values from the output of the first service to use it as the input of the later one. 5. Deploy the flow created. 6. If necessary, start the running of the initial service.
Verification	The deploy does not return error. The correct execution of the flow that imply the data coming from the first service is used correctly as an input for the second one to carry out a specific task. Use a debug node to verify the correct operation.
Postconditions	A composite service has been created that can be used isolated or to compose new composed services. The user can provide different inputs to the first service to see how affects the second one. The second service is fed with the information of the first one.

2.2.10 List permissions

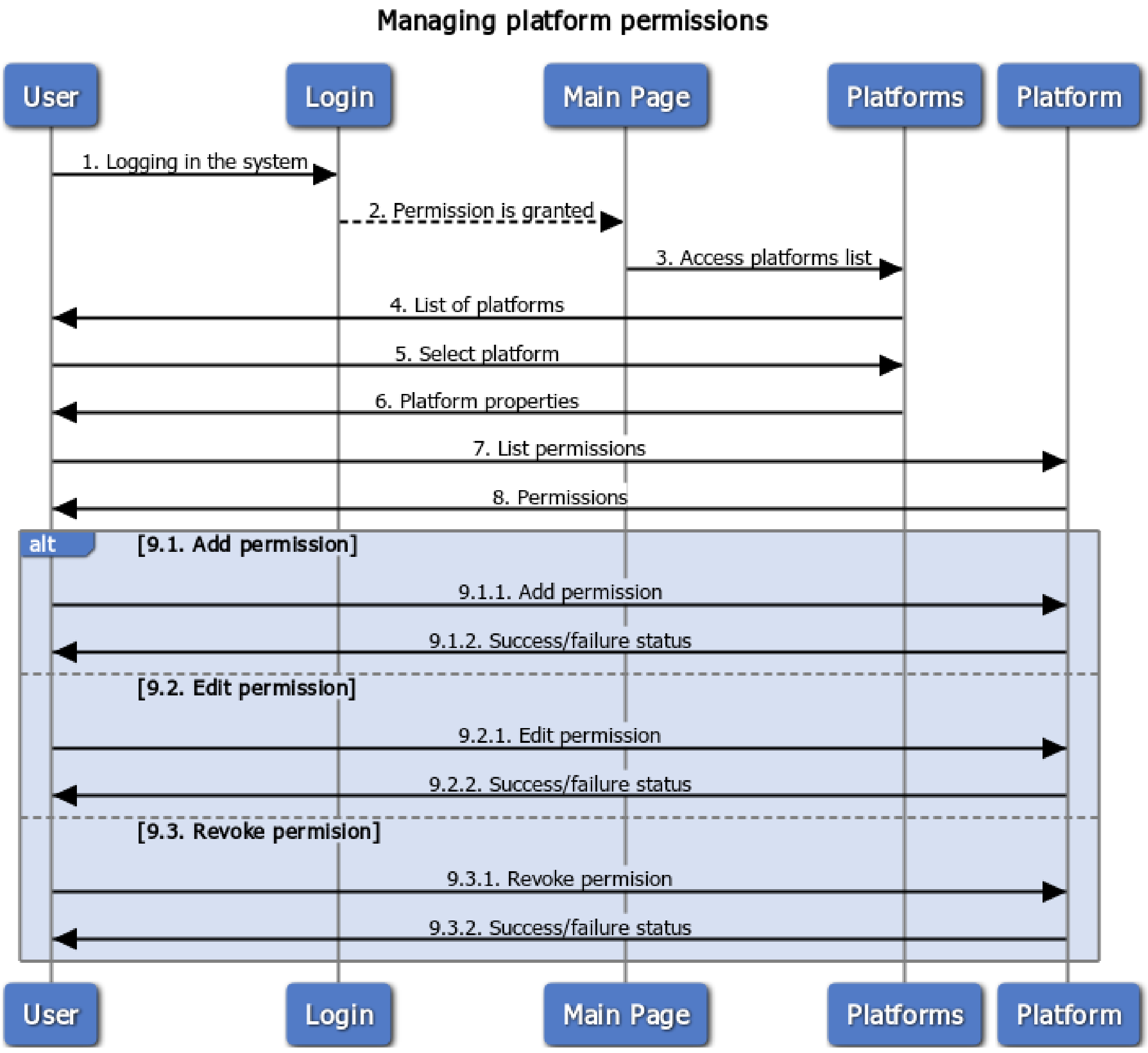
As an INTER-IoT Administrator, I want to list permissions previously granted, so that a summary of the granted permissions is displayed.



Preconditions	User is an INTER-IoT Administrator.
Steps	0. User browses to the INTER-FW admin. 1. User logs in (see 2.2.1) 2. Permission is granted and browser shows the Main Page 3. User clicks on the platforms tab. 4. The list of platforms is shown on screen. 5. User select the platform to list its permissions. 6. The list of platform details is returned. 7. User selects the option “List permissions” 8. List of permission assignment is returned and shown on screen.
Verification	Permissions for the selected user(s) are displayed
Postconditions	none

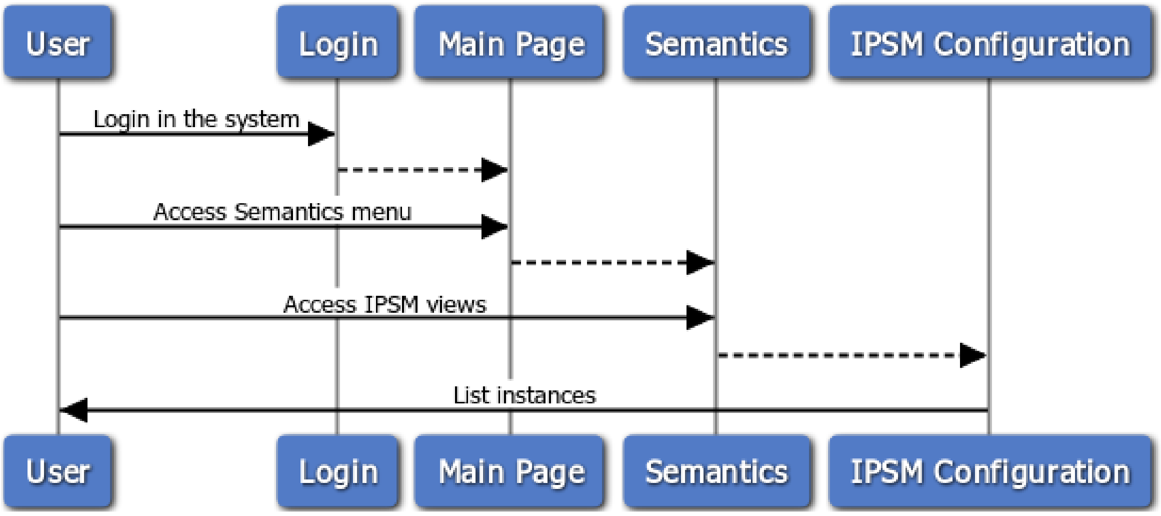
2.2.11 Manage platform permissions

Preconditions	User is an INTER-IoT Administrator.
Steps	0. User browses to the INTER-FW admin <url> 1. User logs in (see 2.2.1) 2. Permission is granted and browser shows the Main Page 3. User clicks on the platforms tab. 4. The list of platforms is shown on screen. 5. User selects the platform to manage permissions. 6. The list of platform details is returned. 7. User selects the option “List permissions” 8. List of permission assignment is returned and shown on screen. 9. Permission management actions **1. Add permission action** 1. User selects “Add” permission action and enters the data for the new permission (User, Rights, Actions) 2. After a successful action the screen is updated with the new permission. In case of error, the error is displayed. **2. Edit permission action** 1. User selects “Edit” permission action and changes the data for an existing permission (Rights, Actions) 2. After a successful action the screen is updated. In case of error, the error is displayed. **3. Revoke permission action** 1. User selects “Revoke (delete)” permission action. 2. After a successful action the screen is updated with the new permission. In case of error, the error is displayed.
Verification	Permissions for the selected user(s) updated
Postconditions	none



2.2.12 List IPSM instances

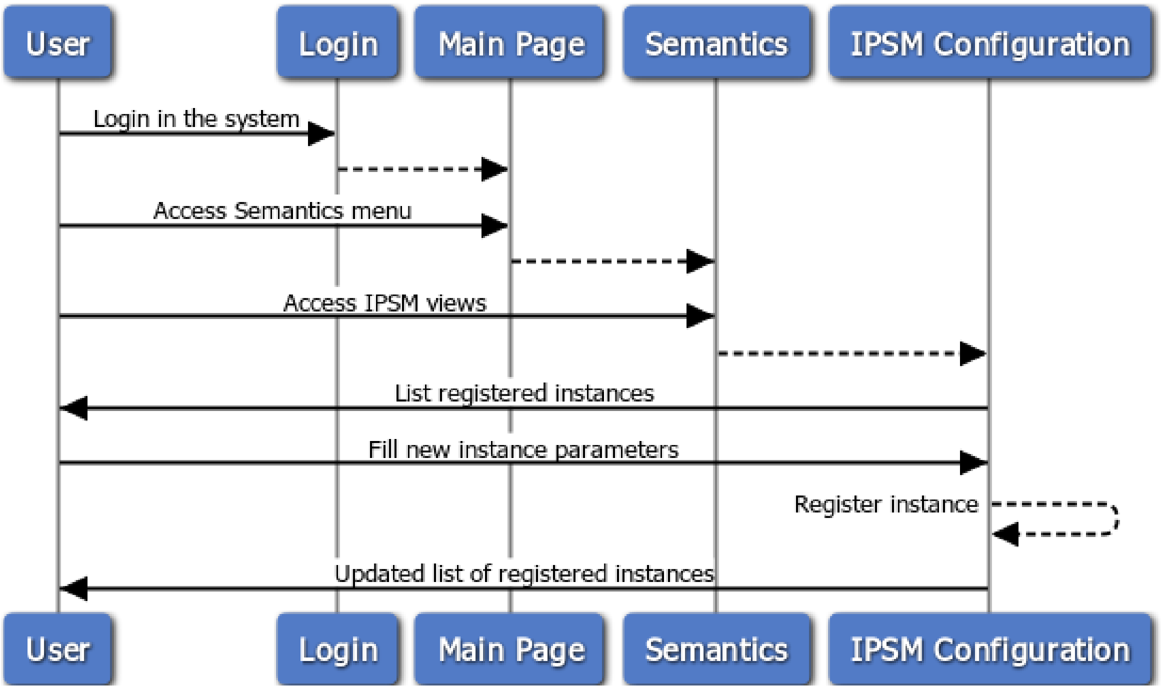
As an INTER-IoT administrator or a developer working for integrated IoT Platform owner I want to get information on what alignments are active (used) in the IPSM deployment, so that I can verify the configuration.



Preconditions	-
Steps	1. Log into INTER-IoT FW Portal 2. Go to *Semantics* page to access IPSM configuration 3. Go to *IPSM Configuration* view 4. Registered IPSM instances are displayed in the table with actions available for each row
Verification	The table with registered instances is successfully populated.
Postconditions	Selected registered instance provides context for operation related to alignments and translation channels within IPSM.

2.2.13 Register IPSM instance

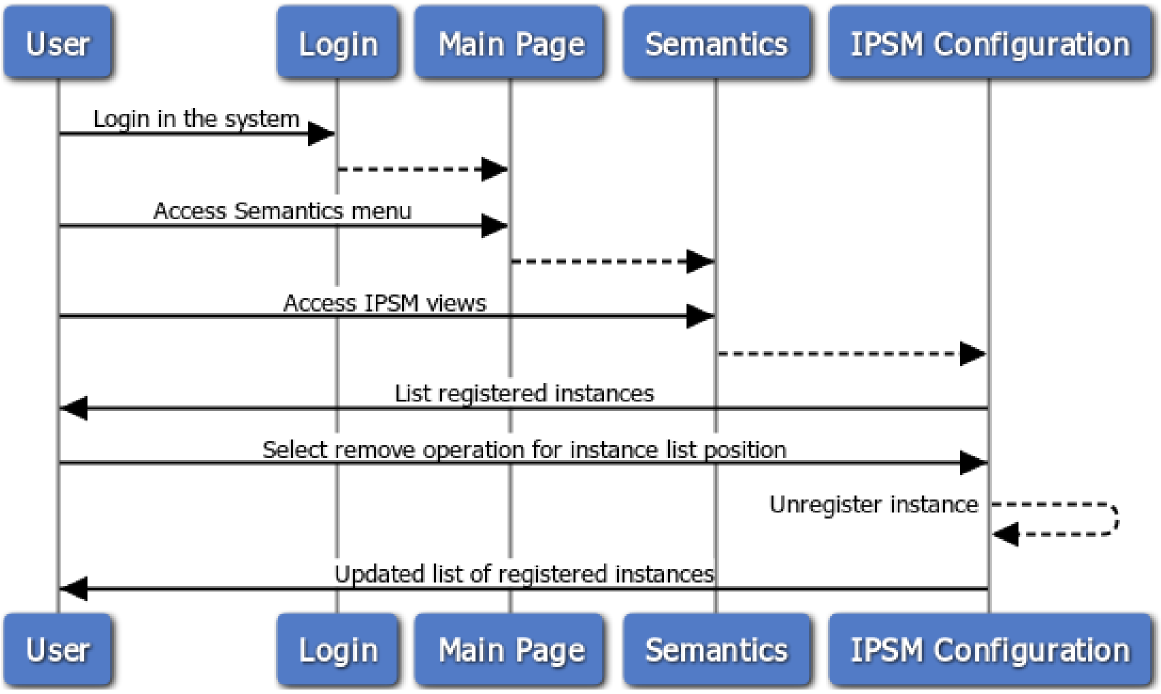
As an INTER-IoT administrator and I want to register existing IPSM installation to be able to interact with it via INTER-FW Portal.



Preconditions	1. Have a host and port of IPSM REST API.
Steps	1. Log into INTER-IoT FW Portal 2. Go to *Semantics* page to access IPSM configuration 3. Select *IPSM Configuration* view 4. Fill in parameters describing IPSM instance to be registered 5. Select *Register IPSM instance* operation
Verification	Newly registered IPSM instance is listed in the registered IPSM instances table.
Postconditions	Selected registered instance provides context for operation related to alignments and translation channels within IPSM.

2.2.14 Unregister IPSM instance

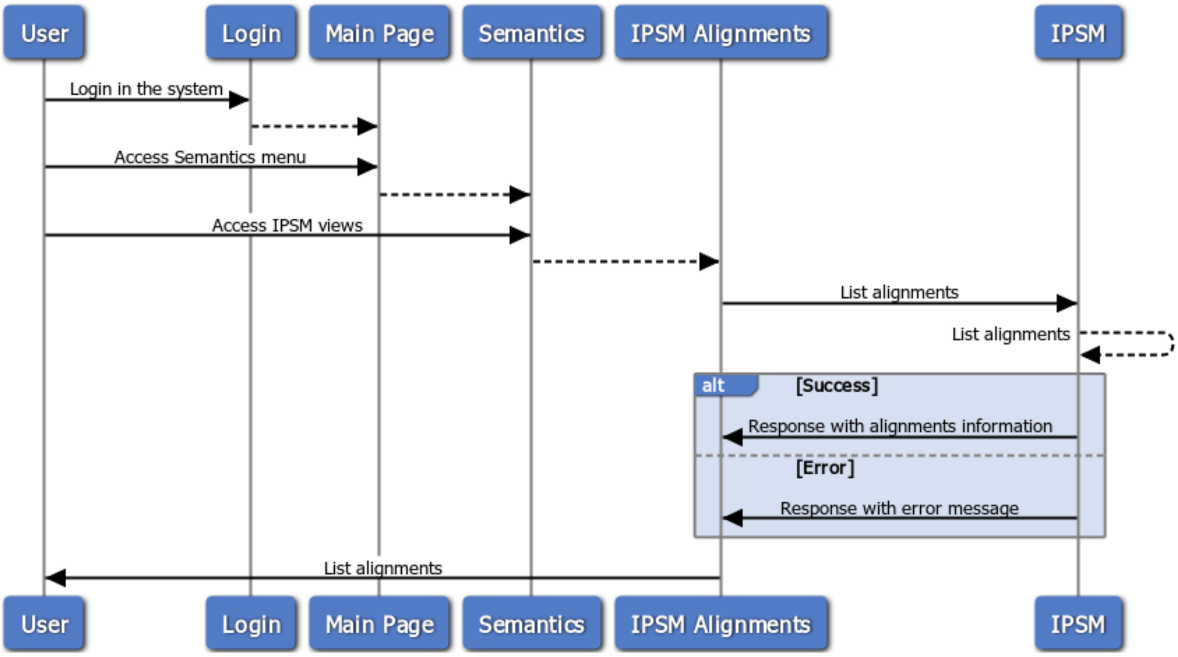
As an INTER-IoT administrator I want to unregister existing IPSM installation since I no longer want to interact with it via INTER-IoT FW Portal.



Preconditions	1. Have a host and port of IPSM REST API.
Steps	1. Log into INTER-IoT FW Portal 2. Go to *Semantics* page to access IPSM configuration 3. Select *IPSM Configuration* view 4. Select *Remove* operation next to the row with an entry representing IPSM instance that is to be unregistered
Verification	Removed IPSM instance is not listed in the registered IPSM instances table.
Postconditions	IPSM instance is no longer attached to the INTER-IoT deployment.

2.2.15 List alignments in IPSM

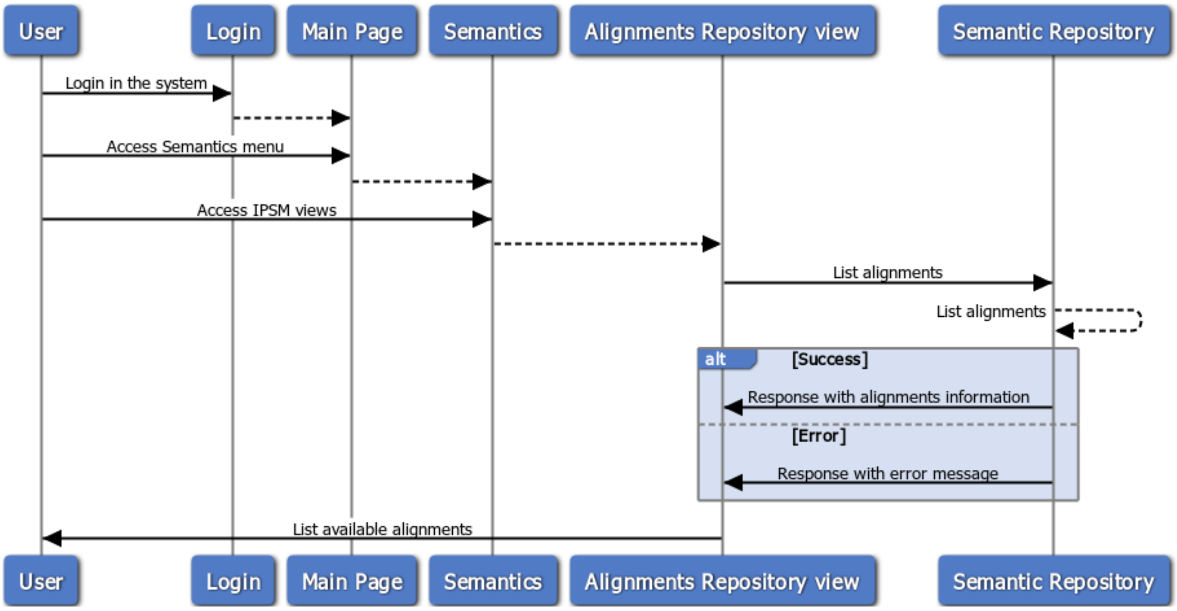
AS an INTER-IoT administrator or a developer working for integrated IoT Platform owner I want to get information on what alignments are active (used) in the IPSM deployment, so that I can manage this alignments or prepare a configuration for a new translation channel.



Preconditions	1. Know the host and port of IPSM instance for which alignments are to be listed
Steps	1. Log into INTER-IoT Portal 2. Go to *Semantics* page to access IPSM configuration 3. On *IPSM Configuration* view mark IPSM instance for which alignments will be listed 4. Select *IPSM Alignments* view 5. The GET operation of IPSM REST API is invoked 6. The response with json content is returned. a. If status 200 is returned then table is successfully populated with available alignments information b. Otherwise, error message returned in response is displayed to the user 7. Available alignments are displayed in the table with actions available for each row
Verification	The table with active alignments containing information about identifier, source ontology, target ontology, author, version, comment is successfully populated.
Postconditions	Returned alignments can be used to create channels or get/view/delete selected alignment.

2.2.16 List alignments in Semantic Repository

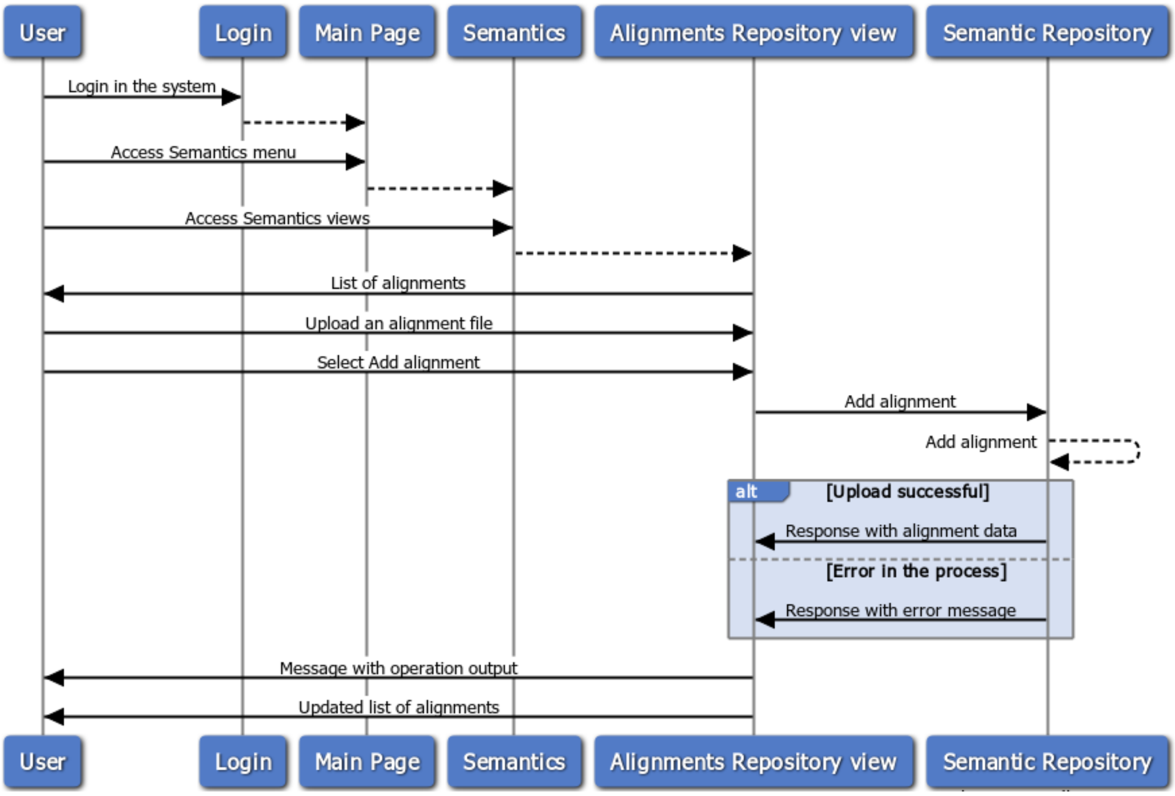
As an INTER-IoT administrator or a developer working for integrated IoT Platform owner and I want to get information on what alignments are available in the Semantic Repository, so that I can manage them (download, view, delete, add a new version) or verify available alignments to be added to a specific IPSM instance.



Preconditions	-
Steps	1. Log into INTER-IoT FW Portal 2. Go to *Semantics* page to access IPSM configuration 3. Select *IPSM Alignments* view 4. The GET operation of IPSM Semantic Repository REST API is invoked 5. The response with json content is returned. a. If status 200 is returned then table is successfully populated b. Otherwise, error message returned in response is displayed to the user 6. Available alignments are displayed in the table with actions available for each row
Verification	The table of available alignments containing information about identifier, source ontology, target ontology, author, version, description, upload date is successfully populated.
Postconditions	Returned alignments can be used to create channels or get/view/delete selected alignment.

2.2.17 Add a new alignment to Semantic Repository

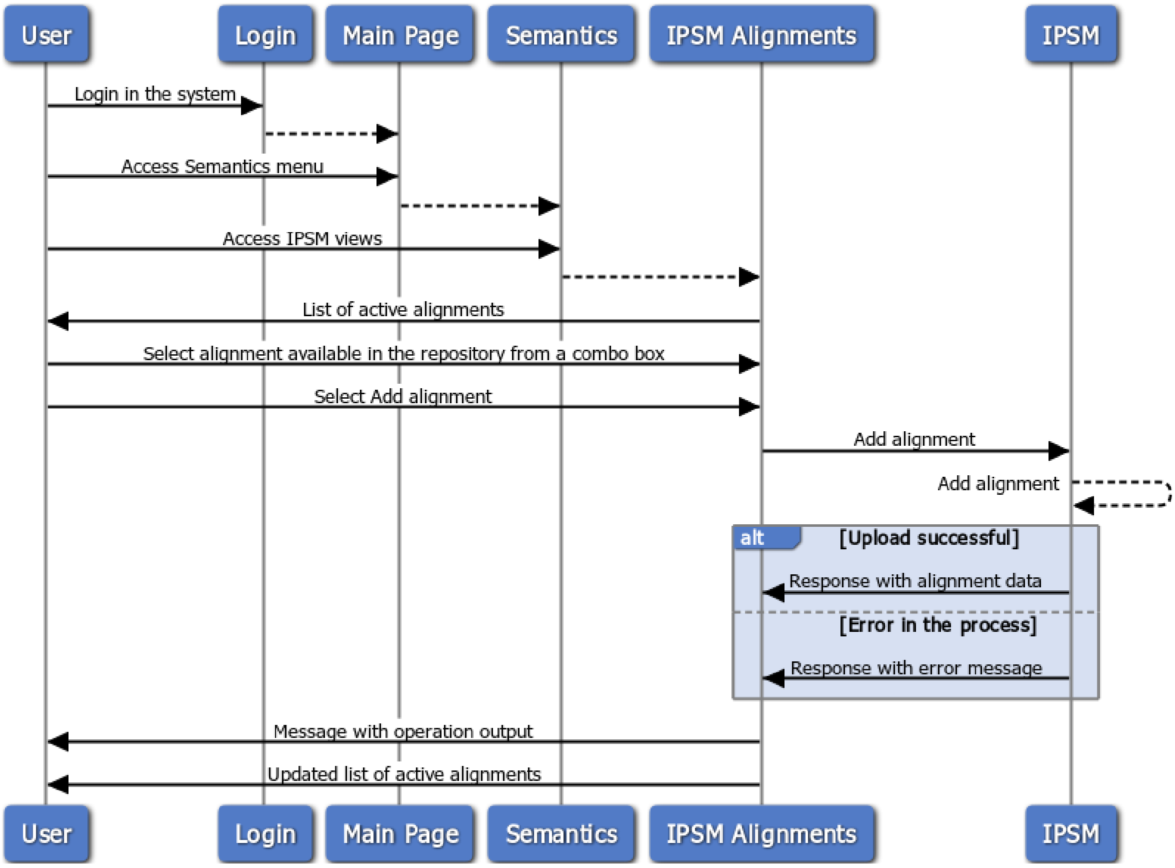
As a semantic engineer working for a IoT Platform owner that has integrated a platform into an INTER-IoT ecosystem, or as an INTER-IoT administrator. I want to publish an alignment to be available within INTER-IoT deployment.



Preconditions	1. Have a new platform to integrate with INTER-IoT (platform is not *natively* supported) 2. NP has an API supporting the basic integration features 3. Have an alignment file persisted in IPSM alignment format. Preconditions for creating alignment: have the platform ontology (existing or created based on INTER-IoT methodology) and the central ontology used in INTER-IoT deployment.
Steps	1. Log into INTER-IoT FW Portal 2. Go to *Semantics* page to access IPSM configuration 3. Select *Alignments Repository* view 4. Upload or drag&drop from local computer an alignment file 5. Select *Add alignment* operation 6. The PUT operation of Semantic Repository REST API is invoked 7. The response with json content is returned a. If status 200 is returned then message that alignment has been uploaded successfully is displayed b. Otherwise, error message return in the response is displayed to the user
Verification	The alignment file is successfully uploaded to Semantic Repository and confirmation of success is returned by Semantic Repository API and displayed to the user. Newly uploaded alignment is listed in the available alignments table.
Postconditions	Alignment is available in Semantic Repository and can be added to specific IPSM instance.

2.2.18 Add a new alignment to IPSM

As a semantic engineer working for a IoT Platform owner and I want to integrate my platform with INTER-IoT ecosystem at the semantic level, so that I can translate data between my platform's semantics and semantics of the other platform that is available in the INTER-IoT ecosystem.

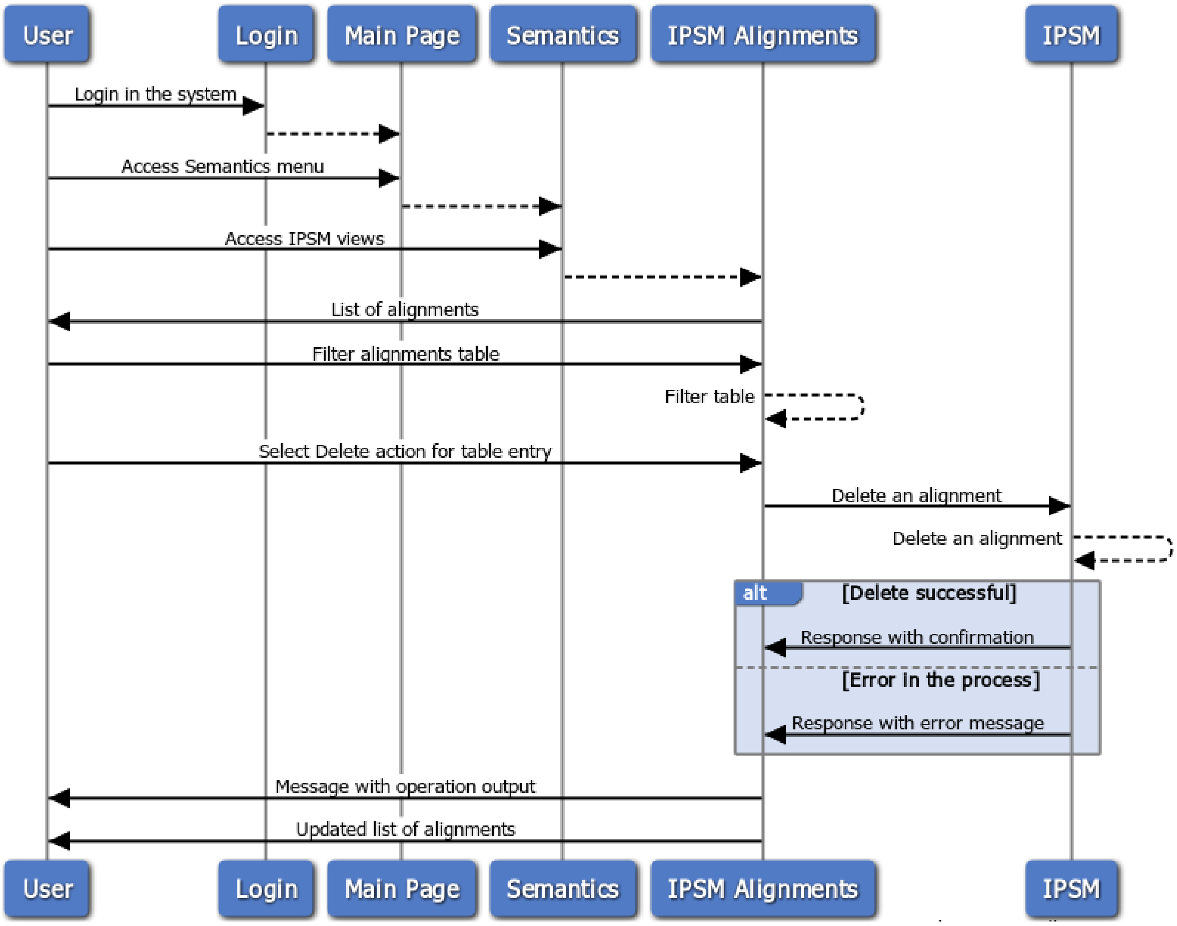

User
Login
Main Page
Semantics
IPSM Alignments
IPSM
Login in the system
Access Semantics menu
Access IPSM views
List of active alignments
Select alignment available in the repository from a combo box
Select Add alignment
Add alignment
Add alignment
alt
[Upload successful]
Response with alignment data
[Error in the process]
Response with error message
Message with operation output
Updated list of active alignments

Preconditions	1. Know which alignment from Semantic Repository should be added to an IPSM instance
Steps	1. Log into INTER-IoT FW Portal 2. Go to *Semantics* page to access IPSM configuration 3. On *IPSM Configuration* view mark IPSM instance for which alignment will be added 4. Select *IPSM Alignments* view 5. From a list of alignments available in the repository select the one that is to be added to IPSM instance 6. Select *Add alignment* operation 7. The PUT operation of IPSM REST API is invoked 8. The response with json content is returned a. If status 200 is returned then message that alignment has been uploaded successfully is displayed b. Otherwise, error message return in the response is displayed to the user
Verification	The alignment file is successfully uploaded to IPSM and confirmation of success is returned by IPSM API and displayed to the user. Newly uploaded alignment is listed in the available alignments table.

Postconditions	Alignment is available in IPSM alignments repository and can be used to create translation channels.

2.2.19 Delete an alignment from IPSM

As an INTER-IoT administrator or a developer working for integrated IoT Platform owner I want to remove an alignment between ontologies that is no longer used in any translation process.

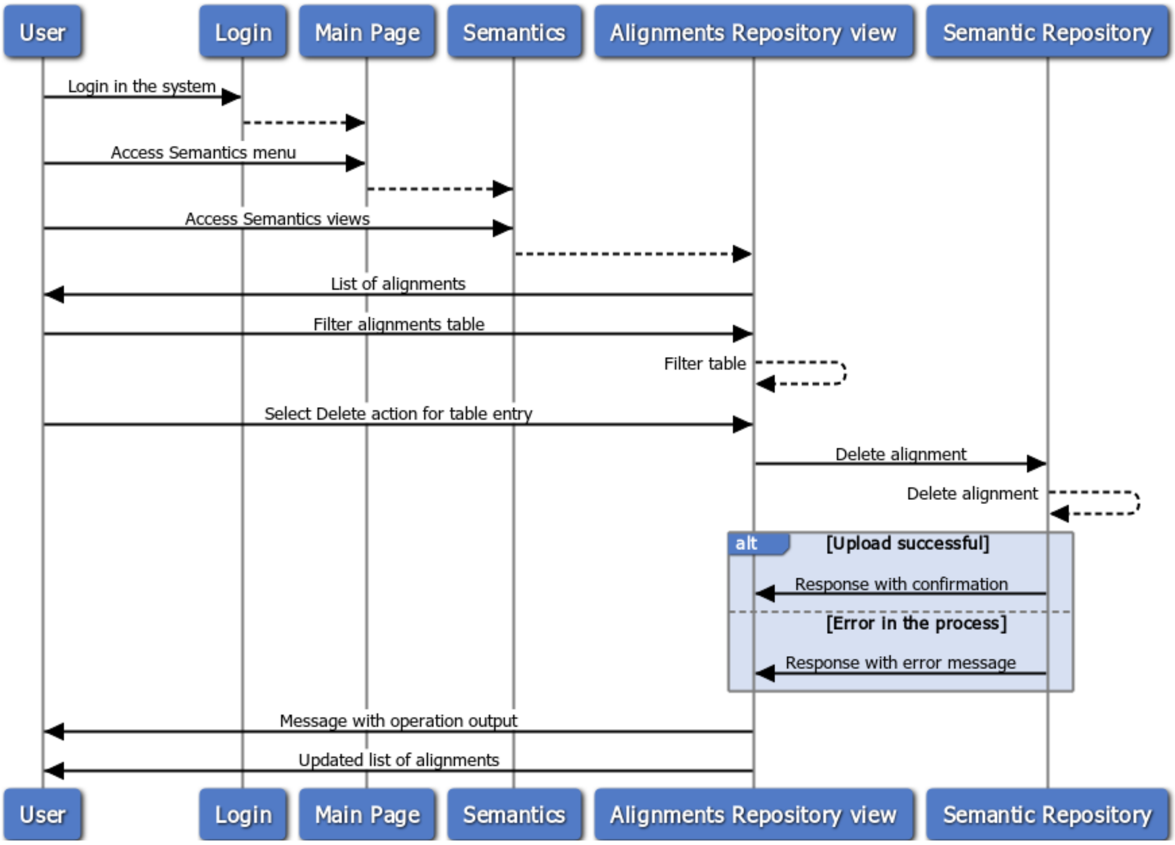


Preconditions	1. Know details of an alignment that is to be deleted (identifier, or source and target ontology)
Steps	1. Log into INTER-IoT FW Portal 2. Go to *Semantics* page to access IPSM configuration 3. On *IPSM Configuration* view mark IPSM instance for which alignment will be deleted 4. Select *IPSM Alignments* view 5. Filter the table of available alignments by identifier, source or target ontology, author, version, or comment 6. Select *Delete* action for a specific alignment entry 7. Confirm the operation 8. The DELETE operation of IPSM REST API is invoked 9. The response with json content is returned a. If status 200 is returned then message confirming than an alignment has been deleted is returned

	b. Otherwise, error message returned in response is displayed to the user (e.g. when alignment is used in any active translation channel)
Verification	Confirmation of successful deletion of an alignment with a given identifier is returned by IPSM API.
Postconditions	List of available alignments does not contain deleted alignments and it is no longer available to be used in channel creation.

2.2.20 Delete an alignment from Semantic Repository

As an INTER-IoT administrator I want to remove an alignment between ontologies that is no longer used in any translation process in any IPSM instance.

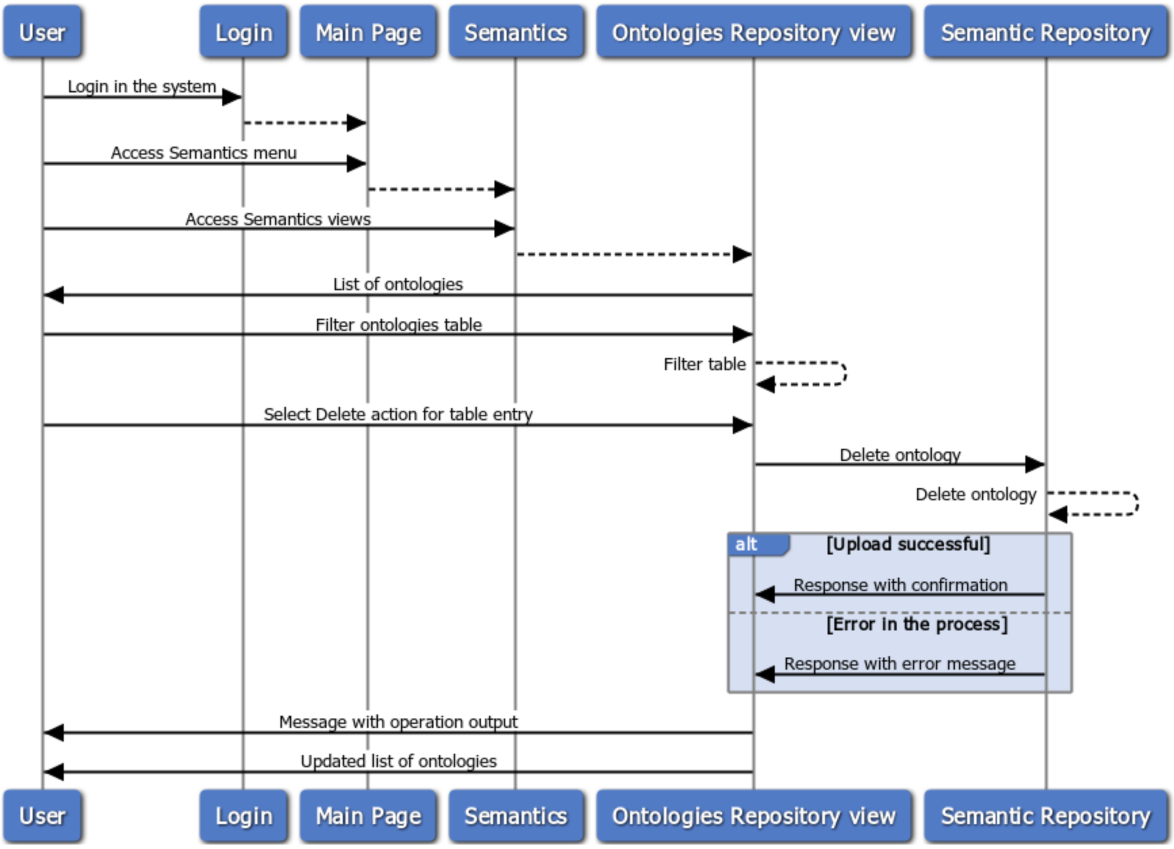


Preconditions	1. Know details of an alignment that is to be deleted (identifier, or source and target ontology)
Steps	1. Log into INTER-IoT FW Portal 2. Go to *Semantics* page to access IPSM configuration 3. Select *Alignments Repository* view 4. Filter the table of available alignments by identifier, source or target ontology, author, version, or comment 5. Select *Delete* action for a specific alignment entry 6. Confirm the operation 7. The DELETE operation of Semantic Repository REST API is invoked

	8. The response with json content is returned a. If status 200 is returned then message confirming than an alignment has been deleted is returned b. Otherwise, error message returned in response is displayed to the user
Verification	Confirmation of successful deletion of an alignment with a given identifier is returned by the API.
Postconditions	List of available alignments does not contain a deleted alignment and it is no longer available to be added to IPSM instance.

2.2.21 Delete an ontology from Semantic Repository

As an INTER-IoT administrator I want to remove an ontology from the Semantic Repository so that it cannot be used as a support in alignment creation. This can happen with a platform's ontology e.g. when a platform is removed from the INTER-IoT ecosystem.

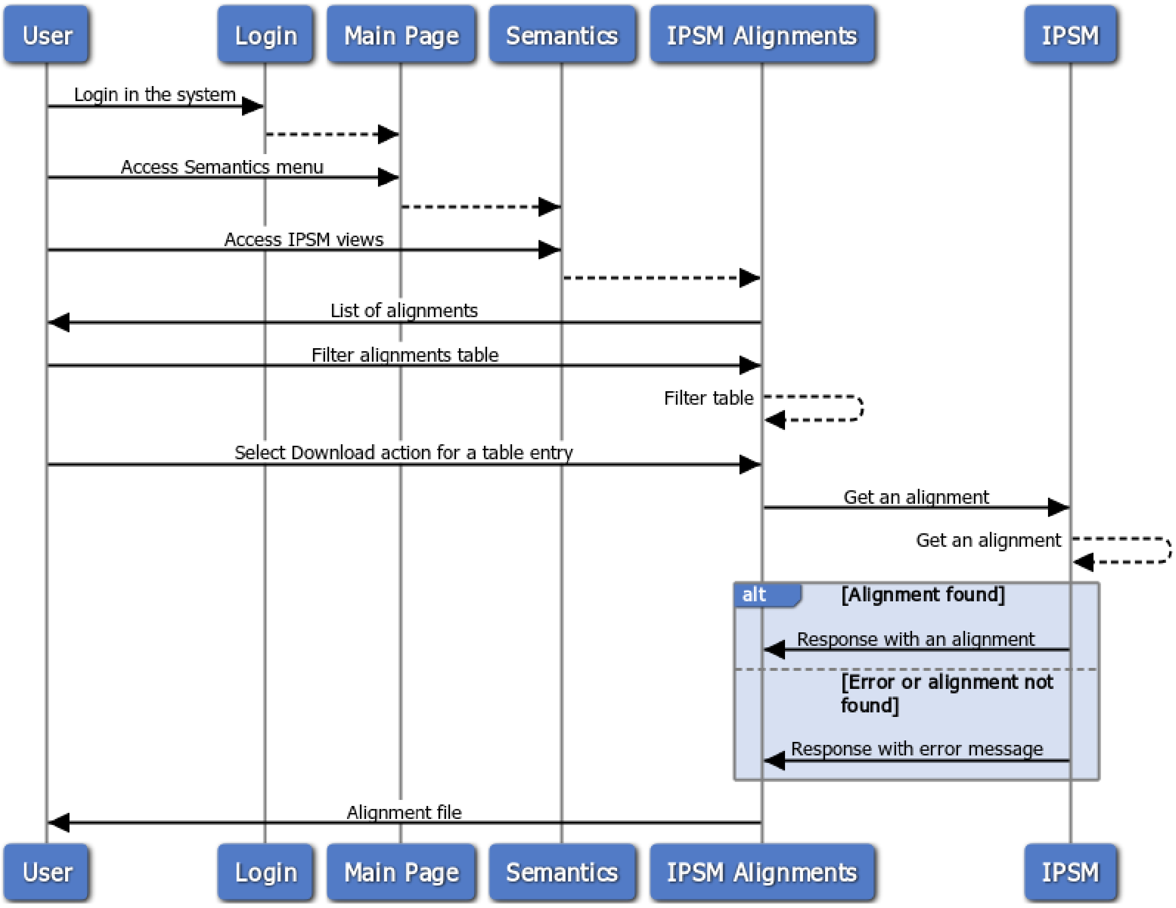


Preconditions	1. Know details of an ontology that is to be deleted (identifier, or base URI)
Steps	1. Log into INTER-IoT FW Portal 2. Go to *Semantics* page to access IPSM configuration 3. Select *Ontology Repository* view 4. Filter the table of available ontologies 5. Select *Delete* action for a specific ontology entry 6. Confirm the operation

	7. The DELETE operation of Semantic Repository REST API is invoked 8. The response with json content is returned a. If status 200 is returned then message confirming than an alignment has been deleted is returned b. Otherwise, error message returned in response is displayed to the user
Verification	Confirmation of successful deletion of an ontology with a given identifier is returned by the API.
Postconditions	List of available ontologies does not contain a deleted ontology.

2.2.22 Get an alignment from IPSM

As an INTER-IoT administrator or a developer working for integrated IoT Platform owner I want to retrieve an alignment between ontologies that has been uploaded to the IPSM instance so that I can verify it or create a new version/alignment based on it.

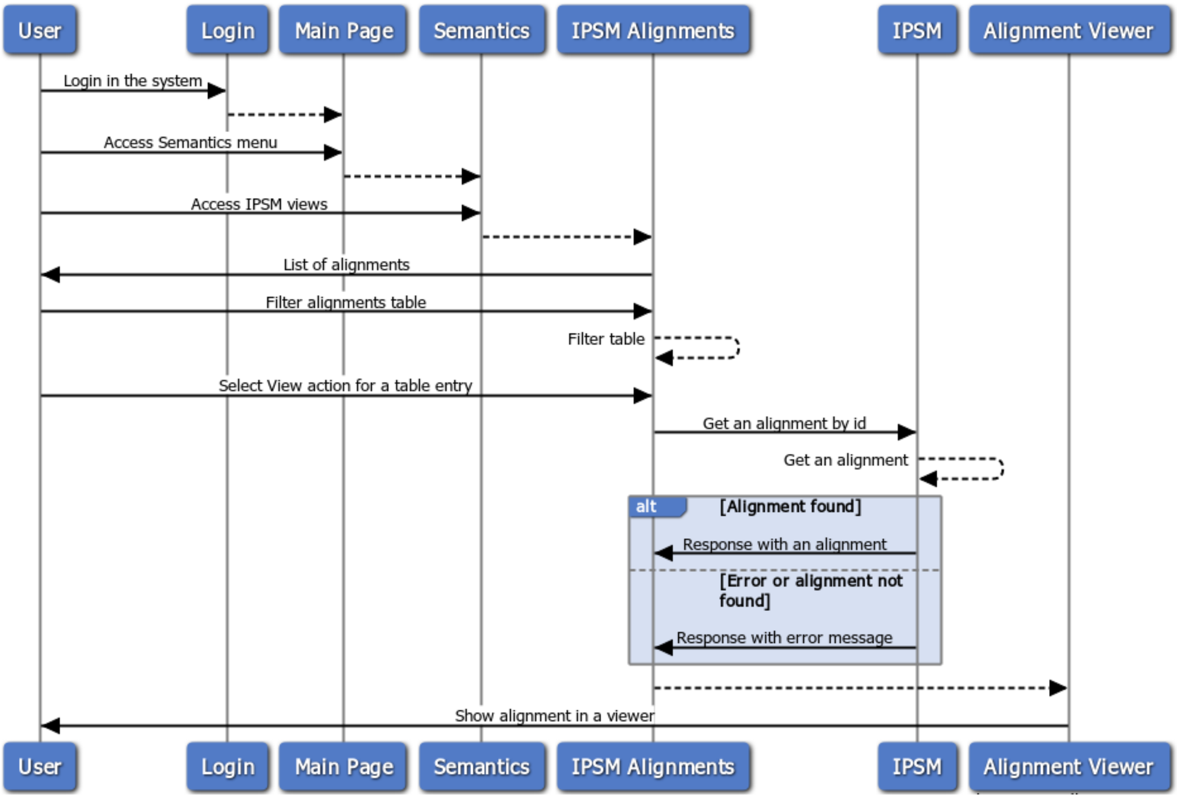


Preconditions	1. Know an identifier (or source and target ontologies URIs) of an alignment that is to be retrieved
Steps	1. Log into INTER-IoT FW Portal 2. Go to *Semantics* page to access IPSM configuration

	3. On *IPSM Configuration* view mark IPSM instance for which alignment will be retrieved 4. Select *IPSM Alignments* view 5. Filter the table of available alignments by identifier, source or target ontology, author, version, or comment 6. Select *Get* action for a specific alignment entry 7. The GET operation of IPSM REST API is invoked 8. The response is returned a. In case of success a content with an alignment is returned b. Otherwise, error message from the response is displayed to the user
Verification	The alignment with a specified identifier is returned to the user.
Postconditions	The alignment can be used by the user as a basis to introduce modifications.

2.2.23 View an alignment from IPSM

I am an INTER-IoT administrator or a developer working for integrated IoT Platform owner and I want to view an alignment between ontologies that has been uploaded to the IPSM deployment.

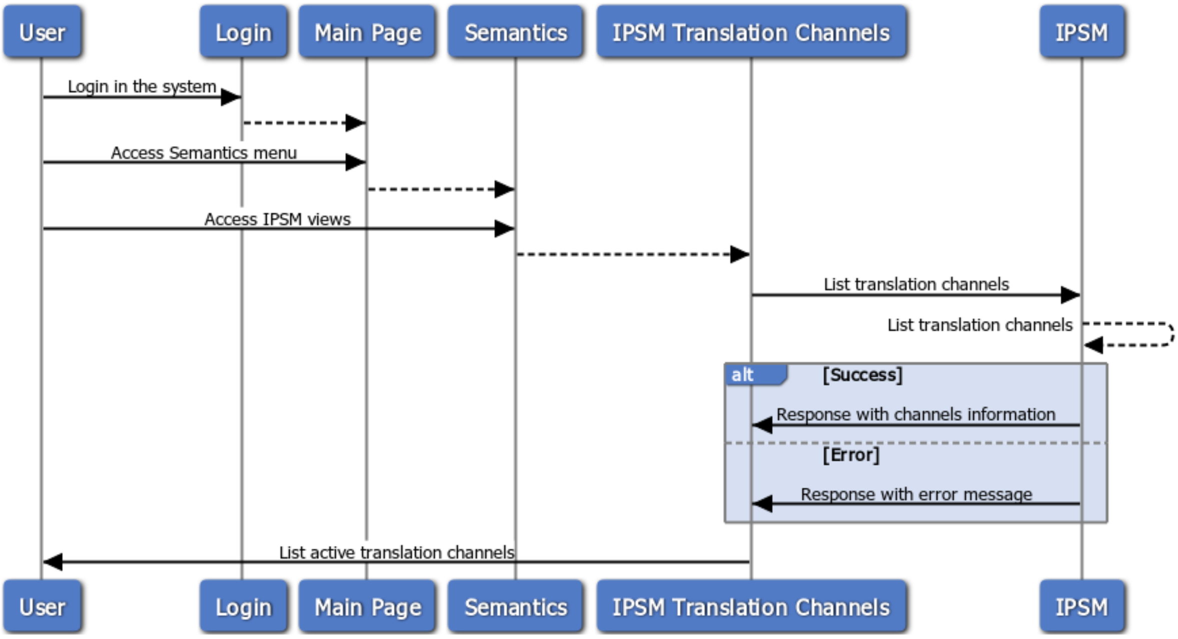


Preconditions	1. Knowa n identifier (or source and target ontologies URIs) of an alignment that is to be viewed
Steps	1. Log into INTER-IoT FW Portal 2. Go to *Semantics* page to access IPSM configuration 3. On *IPSM Configuration* view mark IPSM instance which alignment will be viewed

	4. Select *IPSM Alignments* view 9. Filter list of alignments by identifier, source or target ontology, author, version, or comment 5. Select *View* action for a selected alignment entry 6. The GET operation of IPSM REST API is invoked 7. The response is returned a. In case of success, a content with alignment is returned b. Otherwise, error message from the response is displayed to the user 8. A new tab is open with an alignment viewer displaying a retrieved alignment
Verification	The alignment with a specified identifier (or source and target ontologies URIs) is opened for viewing to the user.
Postconditions	The alignment can be used by the user to investigate the details of semantic translation.

2.2.24 List active IPSM translation channels

As an INTER-IoT administrator or a developer working for integrated IoT Platform owner I want to get information on what channels are active in the IPSM deployment.

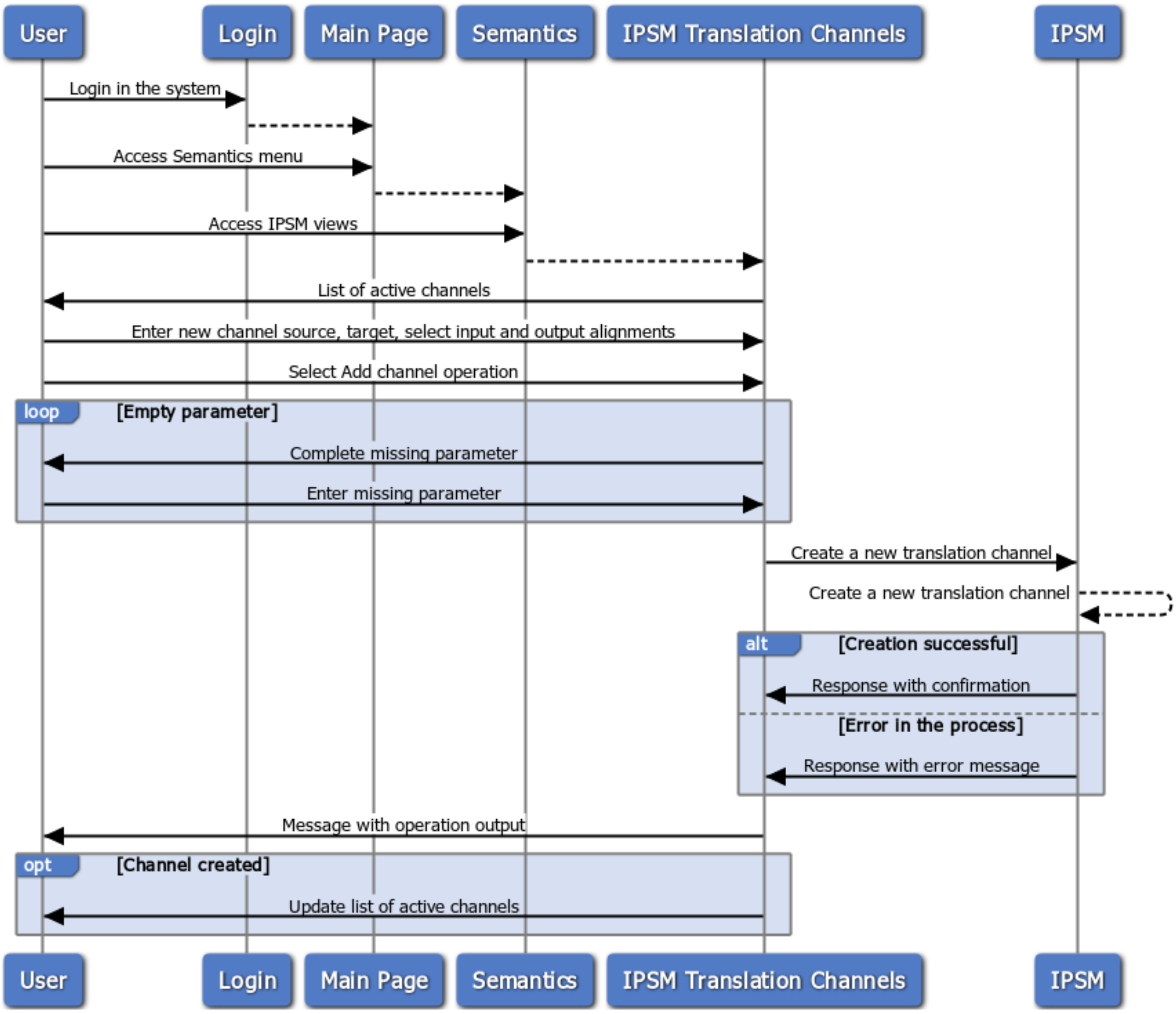

User
Login
Main Page
Semantics
IPSM Translation Channels
IPSM
Login in the system
Access Semantics menu
Access IPSM views
List translation channels
List translation channels
alt
[Success]
Response with channels information
[Error]
Response with error message
List active translation channels
User
Login
Main Page
Semantics
IPSM Translation Channels
IPSM

Preconditions	-
Steps	1. Log into INTER-IoT FW Portal 2. Go to *Semantics* page to access IPSM configuration 3. On *IPSM Configuration* view mark IPSM instance for which channels will be listed 4. Select *IPSM Translation Channels* view 5. The GET operation of IPSM REST API is invoked 6. The response with json content is returned

	a. If status 200 is returned then table with active channels is successfully populated b. Otherwise, error message returned in response is displayed to the user 7. Active channels are displayed in the table with actions available for each row
Verification	The list of channels is returned to the user containing information about identifier, source, target, input alignment and output alignment.
Postconditions	-

2.2.25 Create a new translation channel in IPSM

As a semantic engineer working for a IoT Platform owner I want to integrate my platform with INTER-IoT ecosystem at the semantic level, so that I can translate data between my platform's semantics and semantics of the other platform that is available in the INTER-IoT ecosystem.

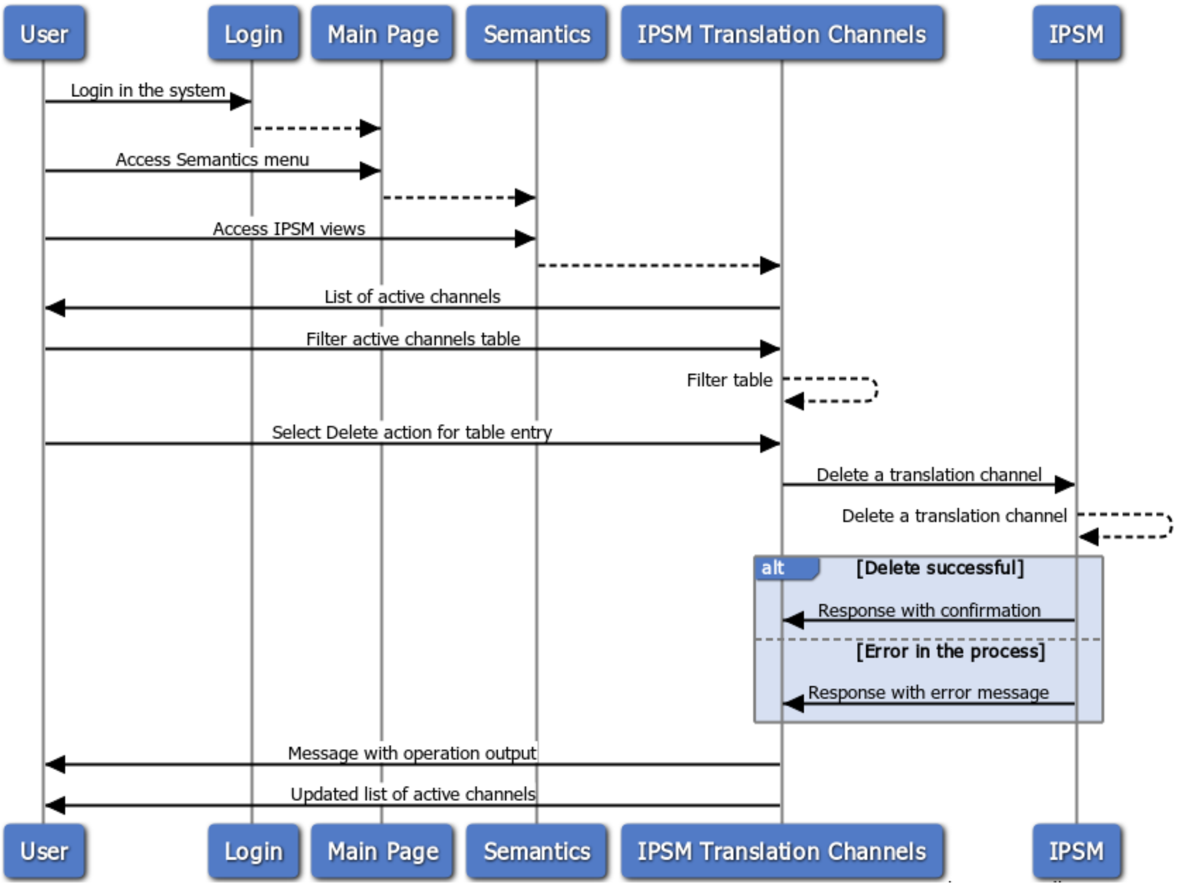

User
Login
Main Page
Semantics
IPSM Translation Channels
IPSM
Login in the system
Access Semantics menu
Access IPSM views
List of active channels
Enter new channel source, target, select input and output alignments
Select Add channel operation
loop
[Empty parameter]
Complete missing parameter
Enter missing parameter
Create a new translation channel
Create a new translation channel
alt
[Creation successful]
Response with confirmation
[Error in the process]
Response with error message
Message with operation output
opt
[Channel created]
Update list of active channels

Preconditions	Know identifiers of two alignments (from source to central ontology, from central to target ontology) that are available in the IPSM instance.

Steps	1. Log into INTER-IoT FW Portal 2. Go to *Semantics* page to access IPSM configuration 3. On *IPSM Configuration* view mark IPSM instance for which channels will be created 4. Select *IPSM Translation Channels* view 5. Enter channel's source and target names, and select identifiers of input and output alignments 6. Select *Add channel* operation 7. The PUT operation of IPSM REST API is invoked 8. The response with json content is returned a. If status 200 is returned then a message that a channel has been successfully created is displayed b. Otherwise, error message returned in response is displayed to the user
Verification	The channel is successfully created in IPSM and confirmation of success is returned by IPSM API and displayed to the user .The list of active translation channels contains the newly created channel.
Postconditions	The translation channel can be used by other INTER-IoT components.

2.2.26 Delete a translation channel in IPSM

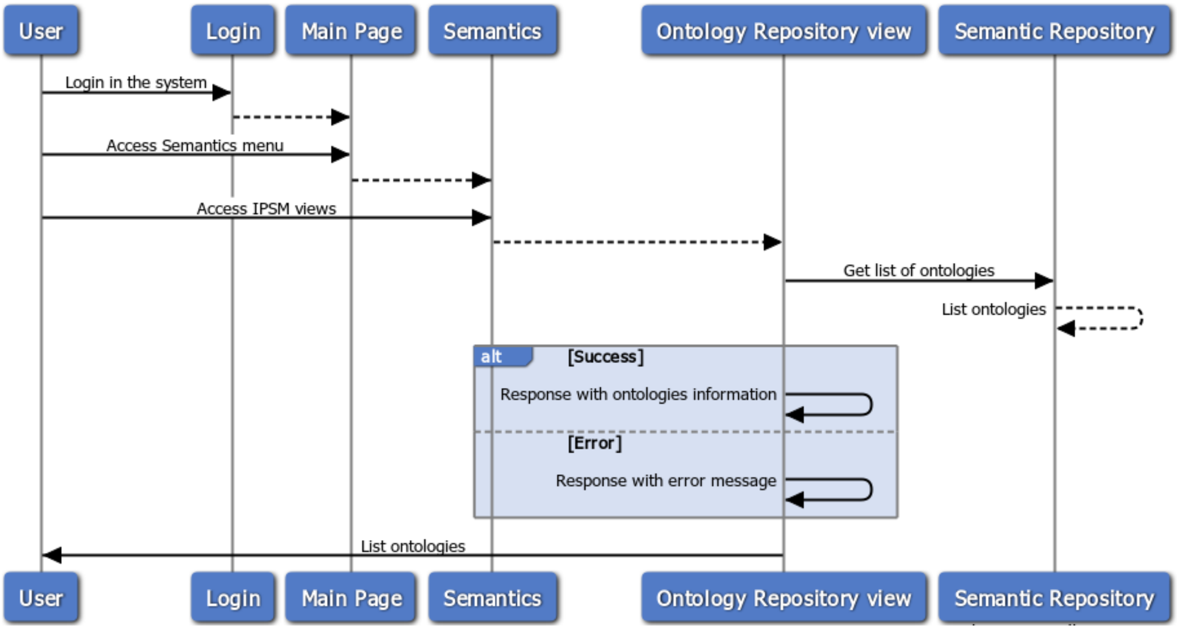
As an INTER-IoT administrator or a developer working for integrated IoT Platform owner I want to remove a translation channel that is no longer used in any translation process.


User
Login
Main Page
Semantics
IPSM Translation Channels
IPSM
Login in the system
Access Semantics menu
Access IPSM views
List of active channels
Filter active channels table
Filter table
Select Delete action for table entry
Delete a translation channel
Delete a translation channel
alt
[Delete successful]
Response with confirmation
[Error in the process]
Response with error message
Message with operation output
Updated list of active channels
User
Login
Main Page
Semantics
IPSM Translation Channels
IPSM

Preconditions	1. Know an identifier (or other identifying data) of a channel that is to be deleted
Steps	1. Log into INTER-IoT FW Portal 2. Go to *Semantics* page to access IPSM configuration 3. On *IPSM Configuration* view mark IPSM instance which channel will be deleted 4. Select *IPSM Translation Channels* view 5. Filter the table of active channels by identifier (or any other data that can be used to identify the channel) 6. Select *Delete* action for a specific channel entry 7. Confirm the operation 8. The DELETE of IPSM REST API is invoked 9. The response with json content is returned a. If status 200 is returned then message confirming than a channel has been deleted is returned b. Otherwise, error message returned in response is displayed to the user
Verification	Confirmation of successful deletion of a channel with a given identifier is returned by IPSM API. The channel is no longer visible in the table with active translation channels.
Postconditions	List active channels operation does not return deleted channel.

2.2.27 List ontologies in Semantic Repository

As an INTER-IoT administrator or a developer working for integrated IoT Platform owner I want to get information on what ontologies are available in the Semantic Repository so that I can use them to create an alignment between them.

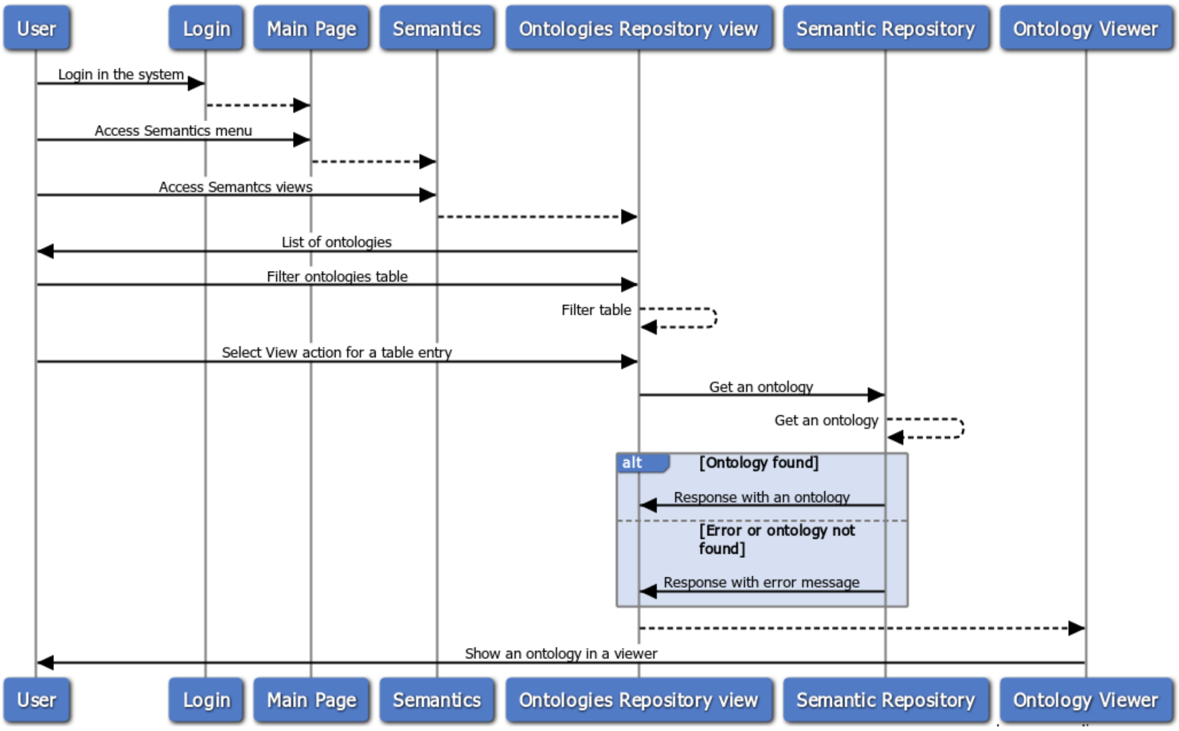


Preconditions	-

Steps	1. Log into INTER-IoT FW Portal 2. Go to *Semantics* page to access IPSM configuration 3. Select *Ontology Repository* view 4. The Semantic Repository API is invoked 5. The response is returned a. In case of success, the table with available ontologies is successfully populated b. Otherwise, error message returned in response is displayed to the user 6. Available ontologies are displayed in the table with actions available for each row
Verification	The list of ontologies is returned to the user.
Postconditions	-

2.2.28 View an ontology in Semantic Repository

As an INTER-IoT administrator or a developer working for integrated IoT Platform owner I want to quickly get information on the details of an ontology available in the INTER-IoT ecosystem.

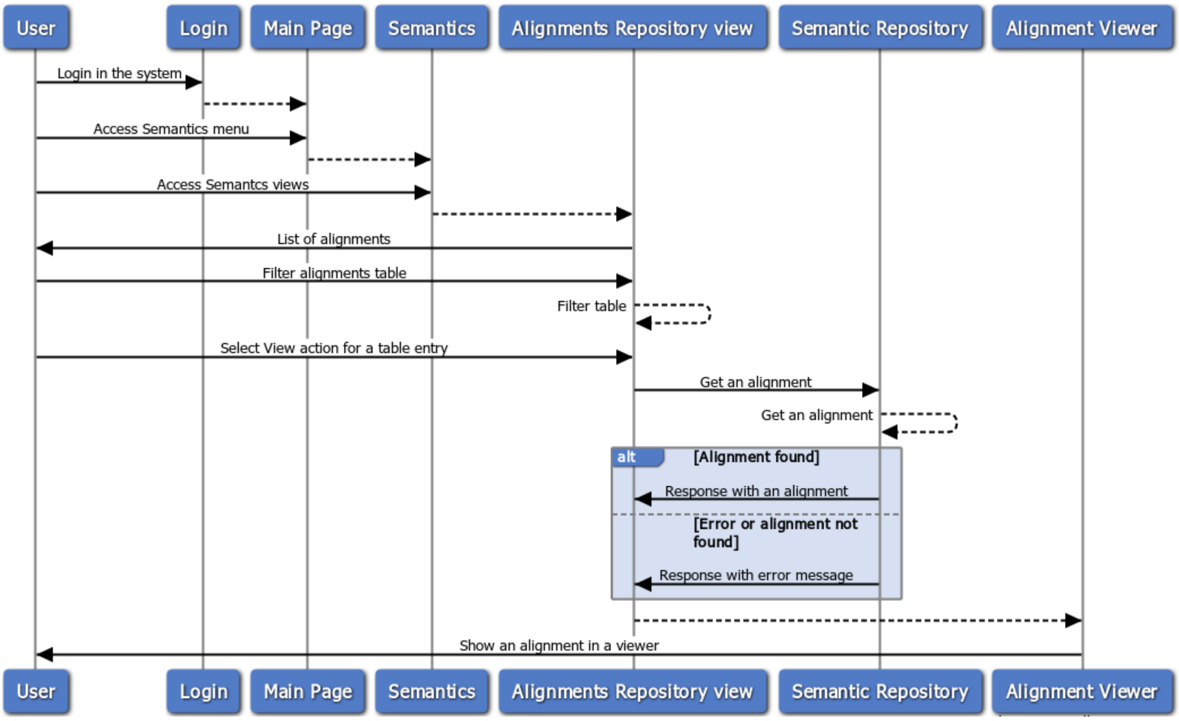


Preconditions	1. Know an identifier or a base URI of an ontology that is to be viewed
Steps	1. Log into INTER-IoT FW Portal 2. Go to *Semantics* page to access IPSM configuration 3. Select *Ontology Repository* view 4. Filter list of ontologies by any field in table header 5. Select *View* operation for a selected ontology entry 6. The Semantic Repository API is invoked a. In case of success an ontology file is returned

	b. Otherwise, error message from the response is displayed to the user 7. A new tab is opened with an ontology viewer
Verification	The ontology with a specified URI is opened for viewing to the user.
Postconditions	The ontology can be used by the user to investigate the details of data representation specific for a given platform or IPSM.

2.2.29 View an alignment in Semantic Repository

As an INTER-IoT administrator or a developer working for integrated IoT Platform owner I want to get information on the details of an alignment available in the Semantic Repository of INTER-IoT ecosystem.

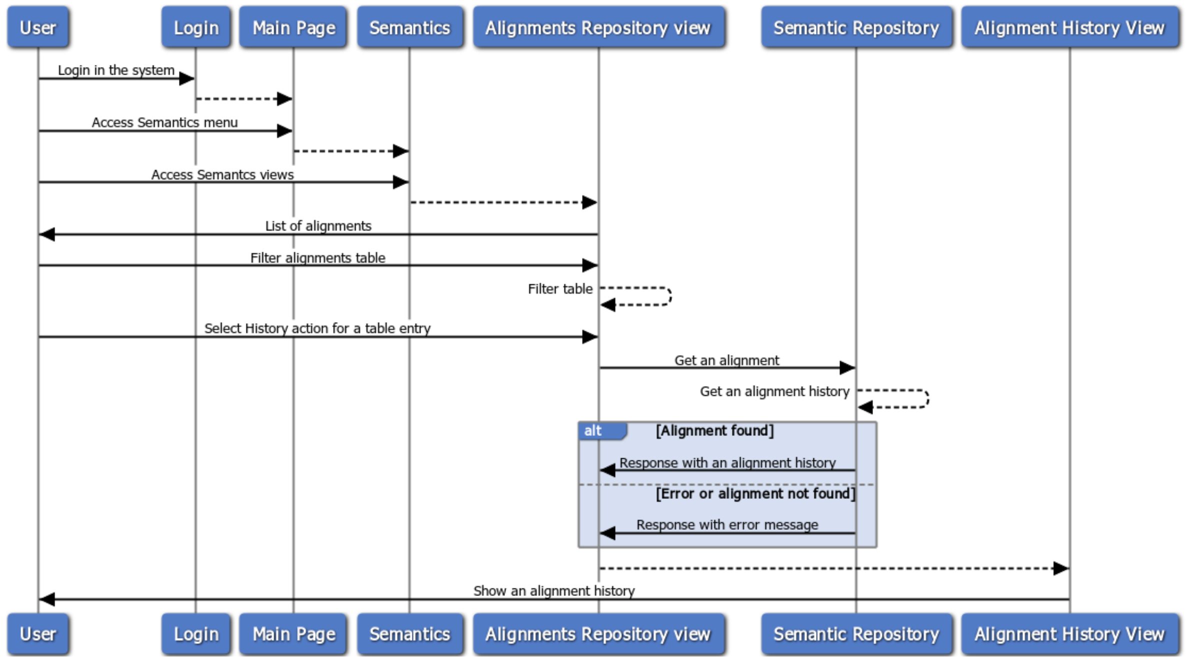


Preconditions	1. Know identifying information for an alignment (e.g. identifier, or source and target ontology)
Steps	1. Log into INTER-IoT FW Portal 2. Go to *Semantics* page to access IPSM configuration 3. Select *Alignments Repository* view 4. Filter list of alignments by any field in table header (identifier, source or target ontology, author, version, or comment) 5. Select *View* operation for a selected alignment entry 6. The Semantic Repository API is invoked a. In case of success an ontology file is returned b. Otherwise, error message from the response is displayed to the user 7. A new tab is opened with an alignment viewer

Verification	The alignment is opened for viewing to the user.
Postconditions	The alignment can be used to configure IPSM instance.

2.2.30 View an alignment history in Semantic Repository

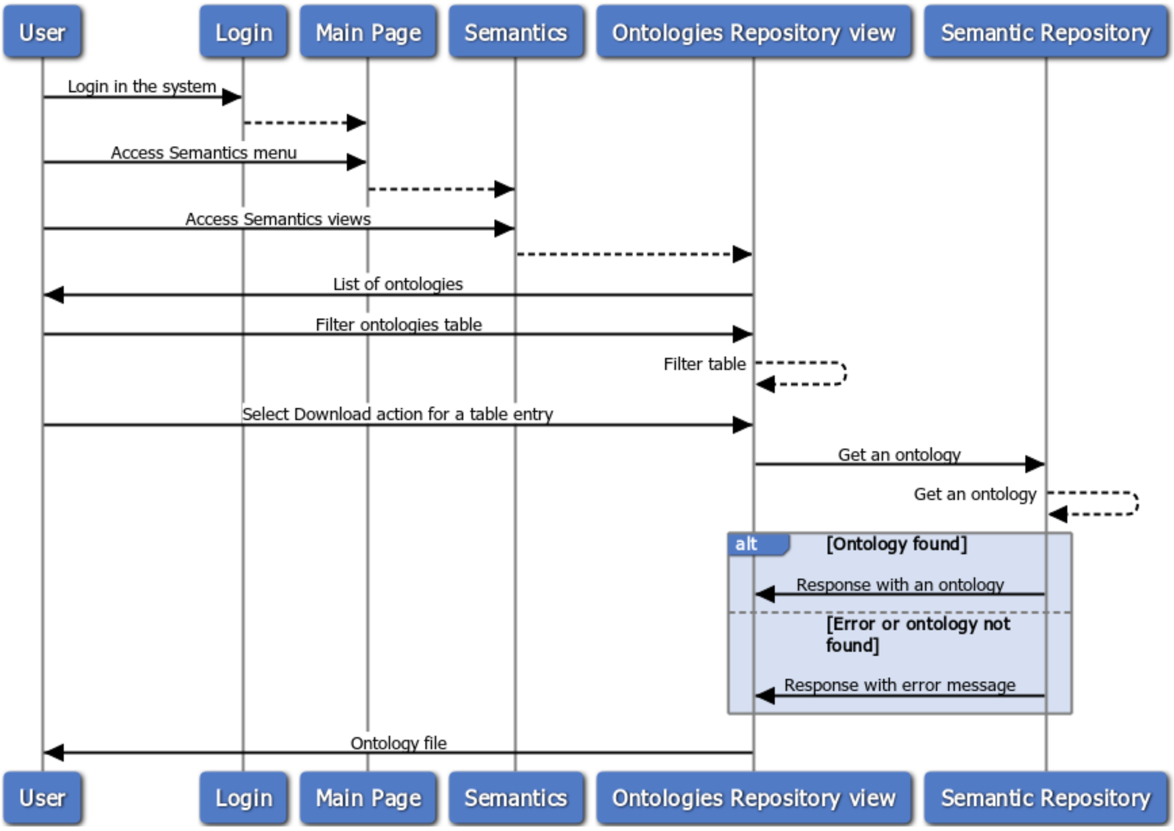
As an INTER-IoT administrator or a developer working for integrated IoT Platform owner I want to get information on the changes of an alignment available in the Semantic Repository of INTER-IoT ecosystem.



Preconditions	1. Know identifying information for an alignment (e.g. identifier, or source and target ontology)
Steps	1. Log into INTER-IoT FW Portal 2. Go to *Semantics* page to access IPSM configuration 3. Select *Alignments Repository* view 8. Filter list of alignments by any field in table header (identifier, source or target ontology, author, version, or comment) 4. Select *History* operation for a selected alignment entry 5. The Semantic Repository API is invoked a. In case of success an ontology file is returned b. Otherwise, error message from the response is displayed to the user 6. A history of alignments sorted by version is displayed.
Verification	A history of alignments sorted by version is displayed.
Postconditions	-

2.2.31 Get an ontology from Semantic Repository

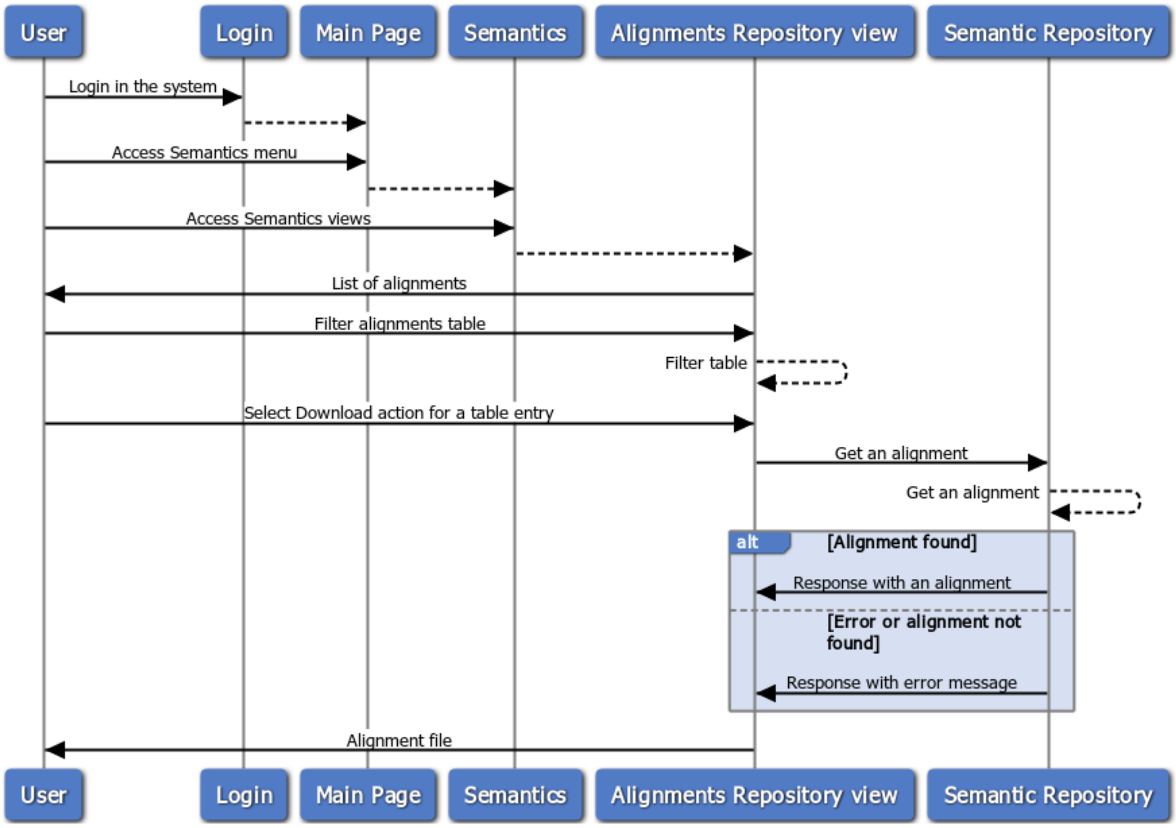
As an INTER-IoT administrator or a developer working for integrated IoT Platform owner I want to get an ontology so that I can create alignments from my platform.



Preconditions	1. Know an identifier or a base URI of an ontology that is to be viewed
Steps	1. Log into INTER-IoT FW Portal 2. Go to *Semantics* page to access IPSM configuration 3. Select *Ontology Repository* view 4. Filter list of ontologies by any field in table header 5. Select *Download* action from for a selected entry with available ontologies (central ontology(ies) is highlighted) 6. The Semantic Repository API is invoked 7. The response is returned 1. In case of success an ontology file is returned 2. Otherwise, error message from the response is displayed to the user
Verification	The ontology available in the Semantic Repository is returned to the user.
Postconditions	The ontology can be used to prepare alignments between IoT platform ontology and central ontology to be used in the translation process.

2.2.32 Get an alignment from Semantic Repository

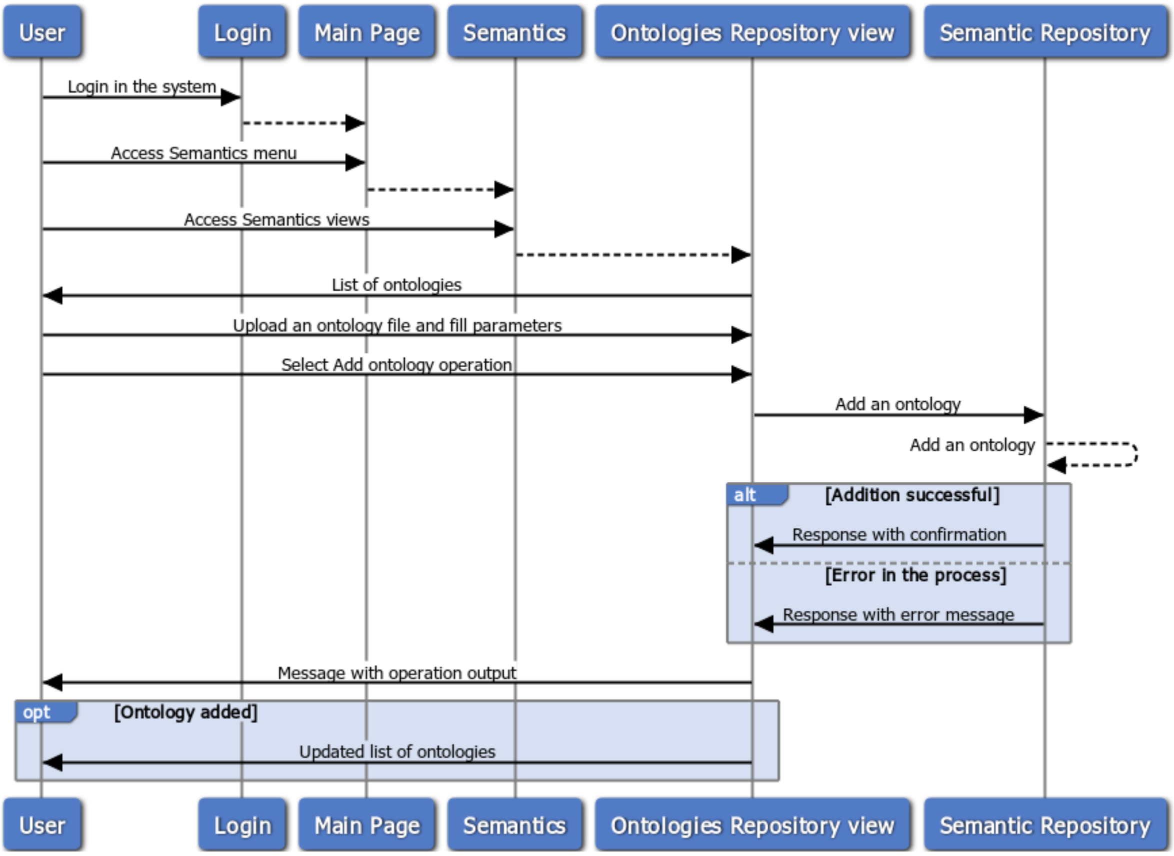
As an INTER-IoT administrator or a developer working for integrated IoT Platform owner I want to get an alignment available in the Semantic Repository so that I can verify it before adding to IPSM instance or create a new version/alignment based on it.



Preconditions	1. Know identifying information for an alignment (e.g. identifier, or source and target ontology)
Steps	1. Log into INTER-IoT FW Portal 2. Go to *Semantics* page to access IPSM configuration 3. Select *Alignments Repository* view 4. Select *Download* action from for a selected entry. 5. The Semantic Repository API is invoked 6. The response is returned a. In case of success an ontology file is returned b. Otherwise, error message from the response is displayed to the user
Verification	The alignment available in the Semantic Repository is returned to the user.
Postconditions	The alignment can be used to configure an IPSM instance or as a basis for a new alignment.

2.2.33 Add a new ontology to Semantic Repository

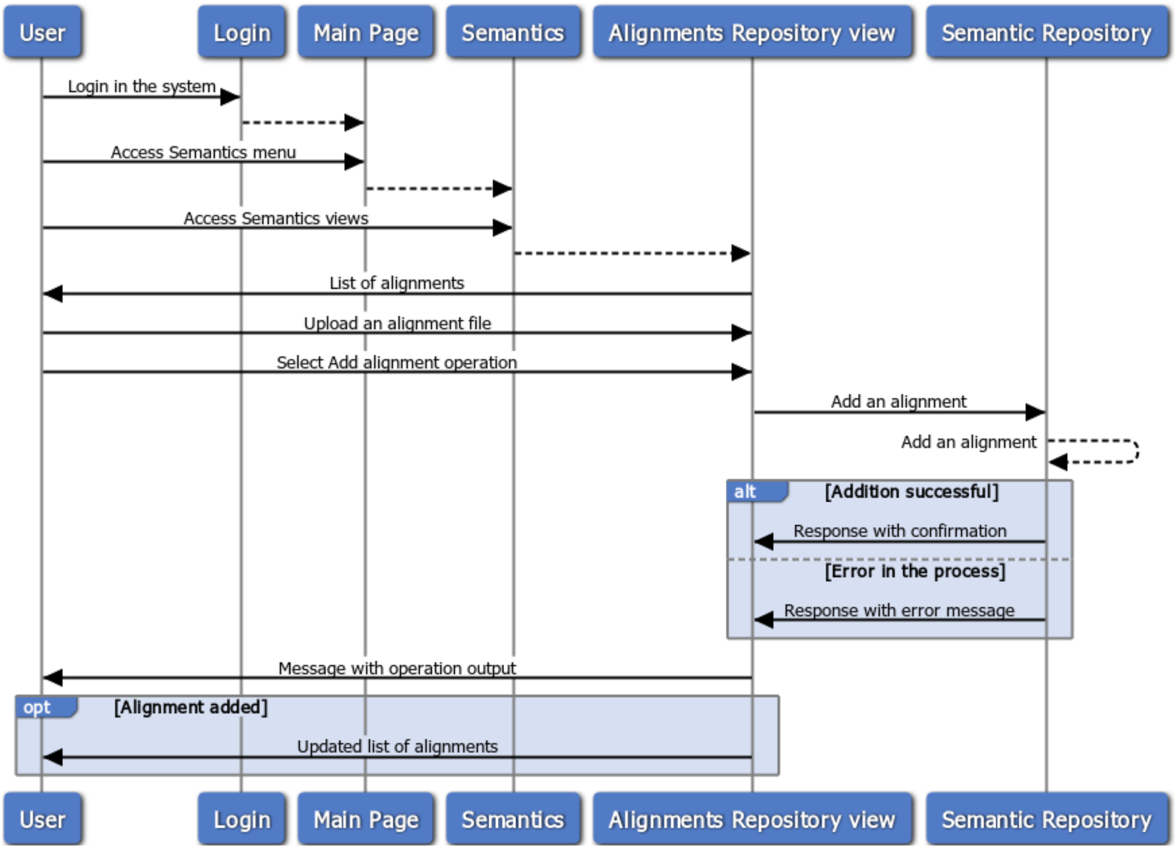
As a semantic engineer working for a IoT Platform owner I want to integrate my platform with INTER-IoT ecosystem at the semantic level, so that I can translate data between my platform’s semantics and semantics of the other platform that is available in the INTER-IoT ecosystem. To enable easier alignment creation I want to make the ontology of my IoT platform available to other users. If I am an INTER-IoT administrator I want to upload a central ontology, to enable others to define alignments to/from it.



Preconditions	1. Have a file with the ontology that is to be uploaded to the Semantic Repository
Steps	1. Log into INTER-IoT FW Portal 2. Go to *Semantics* page to access IPSM configuration 3. Select *Ontology Repository* view 4. Drag&drop or upload the ontology file 5. Fill additional parameters describing the ontology 6. Select *Add ontology* operation 7. The Semantic Repository API is invoked 8. The response is returned a. In case of success, a message that the ontology has been successfully added is displayed b. Otherwise, error message returned in response is displayed to the user (e.g. ontology with such base URI is already added)
Verification	The ontology is successfully added to the Semantic Repository and confirmation of success is returned by the API and displayed to the user. The list of available ontologies contains the newly added ontology.
Postconditions	The ontology can be used by downloaded and viewed by INTER-IoT users.

2.2.34 Add a new alignment to Semantic Repository

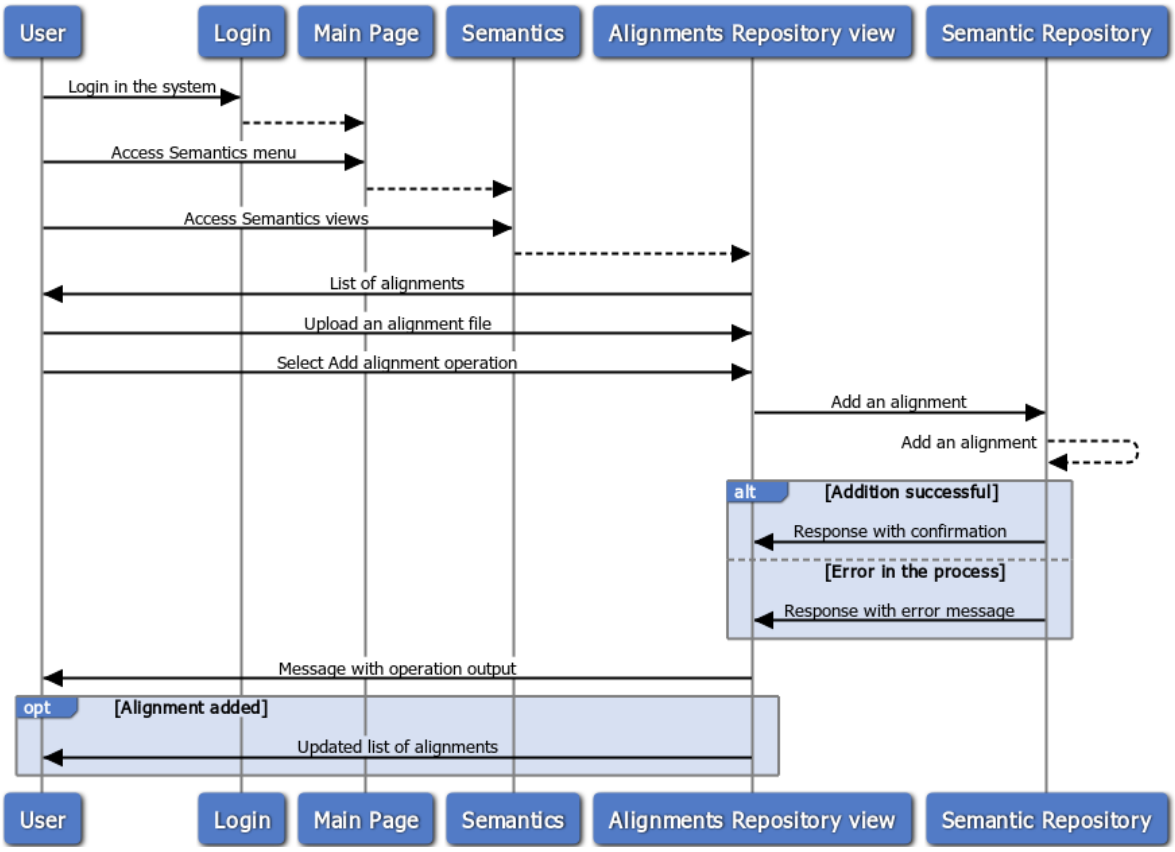
As a semantic engineer working for a IoT Platform owner I want to integrate my platform with INTER-IoT ecosystem at the semantic level, so that I can translate data between my platform's semantics and semantics of the other platform that is available in the INTER-IoT ecosystem. After preparing an alignment I want to make it available in the Semantic Repository to be able to add it to IPSM instances and reuse.



Preconditions	1. Have a file with the alignment that is to be uploaded
Steps	1. Log into INTER-IoT FW Portal 2. Go to *Semantics* page to access IPSM configuration 3. Select *Alignments Repository* view 4. Drag&drop or upload the alignment file. 5. Select *Add alignment* operation 6. The Semantic Repository API is invoked 7. The response is returned a. In case of success, a message that the alignment has been successfully added is displayed b. Otherwise, error message returned in response is displayed to the user
Verification	The alignment is successfully added to Semantic Repository and confirmation of success is returned by the API and displayed to the user. The list of available alignments contains the newly added alignment.
Postconditions	The alignment can be used by downloaded, viewed and added to IPSM instances by INTER-IoT users.

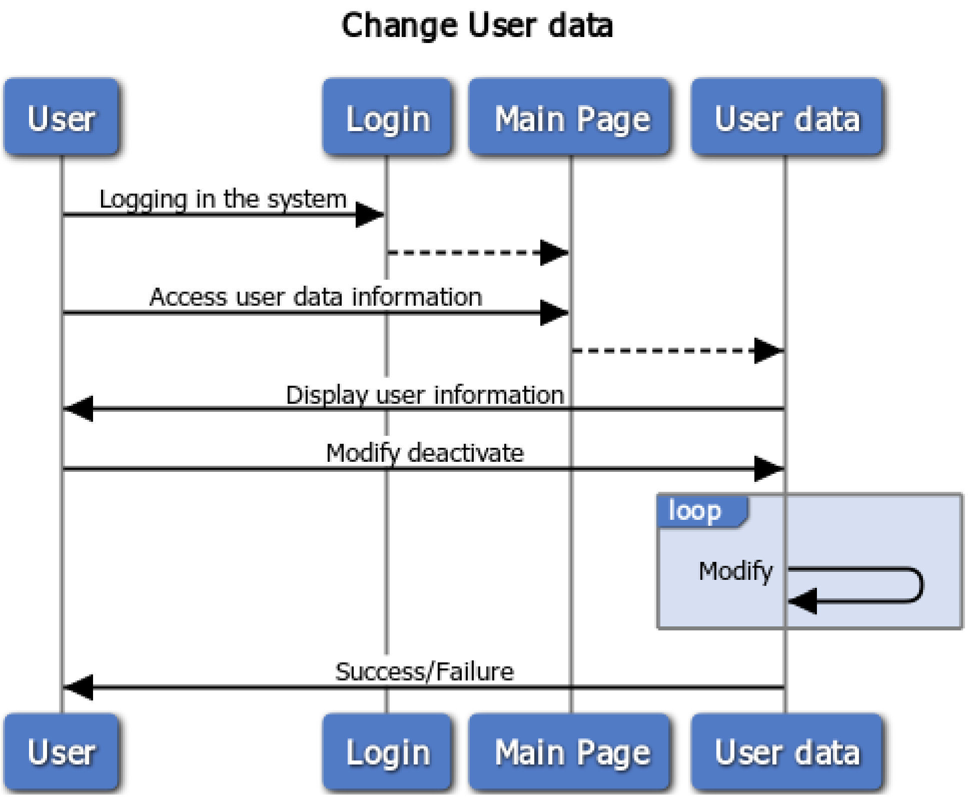
2.2.35 Add a new alignment to Semantic Repository

I am a semantic engineer working for a IoT Platform owner and I want to integrate my platform within INTER-IoT at the semantic level, so that I can translate data between my platform's semantics and semantics of the other platform that is available in the INTER-IoT ecosystem. After preparing an alignment I want to make it available in the Semantic Repository to be able to add it to IPSM instances and reused.



2.2.36 Change user data

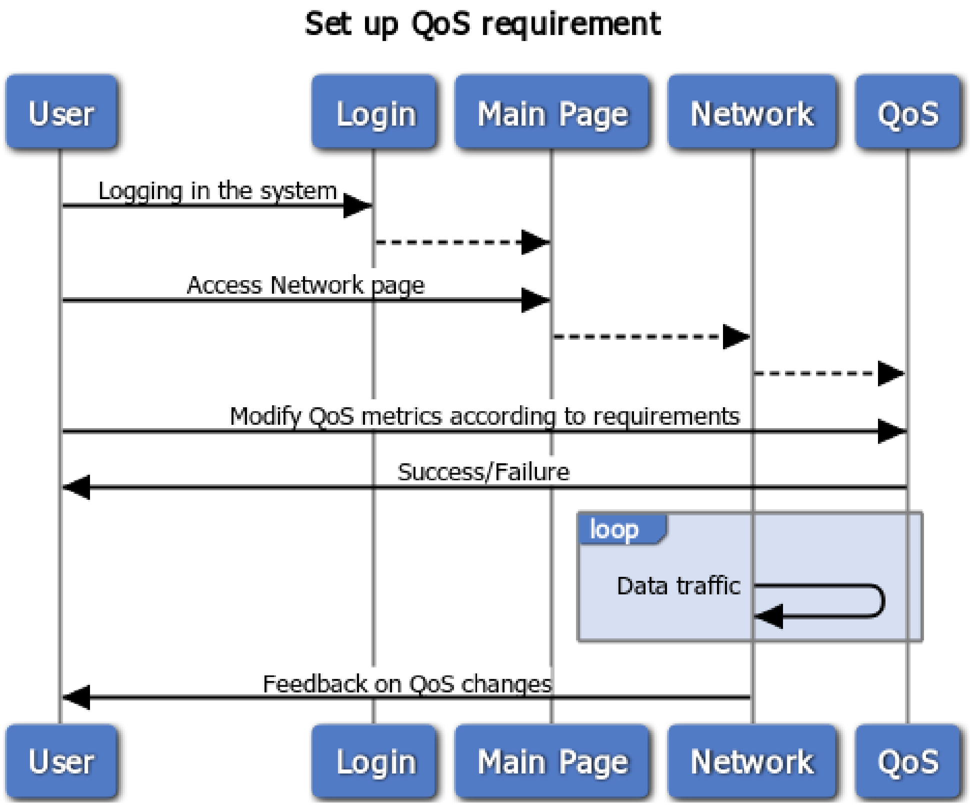
As an INTER-IoT administrator, I want to be able to access and modify user data, so that changes can be made to ensure the good functioning of all processes.



Preconditions	
Steps	1. User logs in and access its personal user data page 2. User modifies its information data
Verification	User is notified upon data change, whether it is successful of not.
Postconditions	If the operation was successful, the user data is up to date. Else, the state of the user data remains unchanged.

2.2.37 Set up QoS requirements

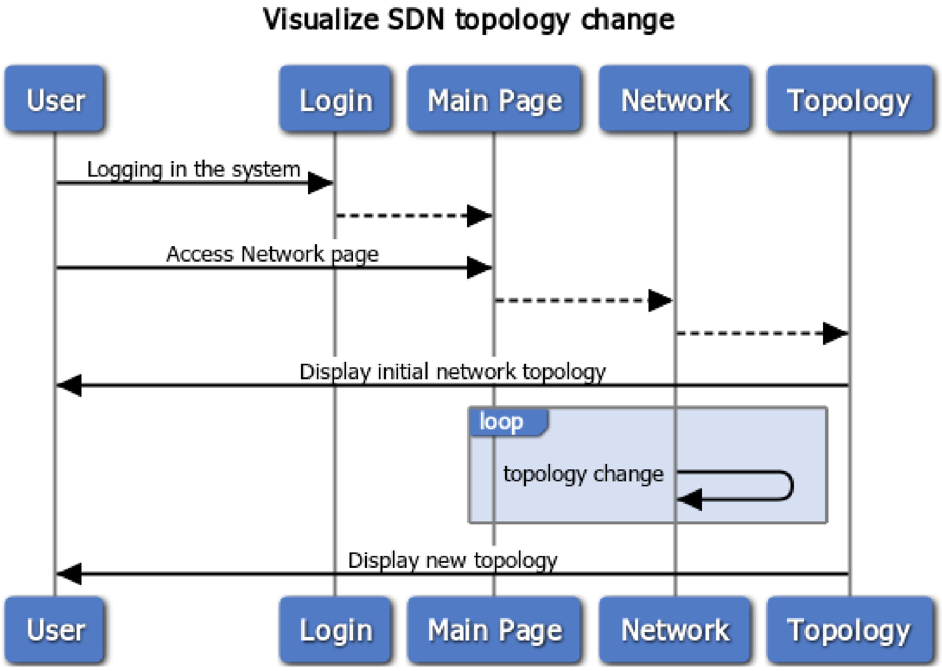
As an Inter-IoT administrator, I want to be able to modify QoS parameters for the virtual SDN network to satisfy requirements of an application.



Preconditions	At least one switch is running in the network and configured in the portal
Steps	1. Go to the network tab 2. Go to the sub-tab of QoS 3. Insert the configuration of the QoS parameter (meter, queue or rule) including the ID of the switch where you want to include the parameter. 4. POST the configuration and wait for the success reply.
Verification	In first place, a success message should appear once you include the QoS parameter in the system, later if you list the indicated parameter a new entry should appear. Thus, the switch in which this parameter is configured should act consequently.
Postconditions	After the inclusion of the parameter, the network should keep working under this specification.

2.2.38 Visualize SDN topology changes

As an Inter-IoT administrator, I want to be able to vizualise the topology changes occuring within the SDN network in order to monitor and adapt the virtual network.



Preconditions	User is already registered. At least one switch is running in the network.
Steps	1. Go to the network tab 2. Go to the sub-tab of topology and observe the topology 3. Click in any of the nodes to obtain information about the node 4. If it is needed go to the sub-tab of statistics and check and configure the parameters inside the switches 5. POST the configuration changes and wait for the success reply.
Verification	In first place, a success message should appear once you configure something within a switch of the network, later if you list the virtual network devices a new entry should appear. Thus, the switch in which this parameter is configured should act consequently and if there is a topology change this one can be observed through the topology tab.
Postconditions	After the inclusion of the parameter, the network should keep working under this specification and you can observe the changes in the topology.

2.3 Back-end analysis

The INTER-FW web application delegates a great part of its features in the LIIs APIs. However, to bring practical features that may eventually let the INTER-IoT to understand and manage the concepts of multilayer interoperability of IoT Platforms, some back-end operations are needed.

In general, back-end operations are those which take on the features serving, transforming and persisting the data of the application.

2.3.1 Requirements

A set of requirements were identified related to the back end of the INTER-FW in the initial stages of the project. Since the definition of the layers, the scenarios and, in some cases, the global functionality have been slightly modified, some extra requirements and modifications have been added at the end to cover all the features planned.

The complete list of requirements affecting the back end is as follows. To have a complete description of these requirements, see Annex I.

Title	Id
Scalability. Computing resources	3
Extensibility	10
Remote device control	26
System security	27
System privacy	28
API for third-party developers	47
API for data publication	51
API REST	52
Auditability and Accountability	58
Constraints on processing of personal data	61
Constraints on processing of personal and health data	62,143,145,146,152
Provision of authentication credentials	63
Confidentiality	69
Users manage how their public data is seen	77
Use of standards	123
Adaptability	125
Portability	132
QoS Integration	142
Design of required ontologies	186
Integration with legacy systems	188
IDEs and APIs for rapid new applications development	199
Each data unit is identified univocally	254
Each sensor has a unique INTER-IoT identifier	256
The INTER-IoT unique ID is used to find the platform-specific ID of the device	257
Manage user permission	260
Manages group-based permissions	262
Access to personal data needs to be previously authorized	263
API allows create/update/remove users	264
API allows device declaration and configuration	264
API allows resources/capabilities discovery	266
API allows historic data query	268
API allows subscription to data streams/queues	270
Stores recent data for recovery	272
Stores system status for recovery	273
Time triggers	276
Indivisibility	277
Future-proof	278
Requests filtering	280
Map publish/subscription between platforms in the middleware	282
Manage a sensor or actuator	283

Table 1 Backend related user requirements

2.3.2 UML class diagrams

In this section, there are represented the different models (entity models) that will be needed to accomplish the requirements, use cases and sequence diagrams reviewed in the previous sections in the case of the INTER-FW Web Console. Considering the modularity of the application, these models are divided in each of the different layers managed in the application, in addition to the user's model. Once implemented, these models will be the basis of the storage objects to make information such as instant status or current configuration persistent in database.

The diagrams are represented in UML class style. They represent basic relations among entities. To represent enumerations, a prefix *(enum)* is used as well as a light blue background. The level of detail of the entity fields is different depending on the degree of maturity of each Web Console section by the moment of writing this document. As mentioned, the basic input to achieve these documents comes from previous subsection 2.3.1 and the front-end analysis (sect. 2.4).

2.3.2.1 D2D Layer

The 'Gateways' tab encapsulates the framework features based on the D2D layer. Basic information of the gateway must be represented and persisted.

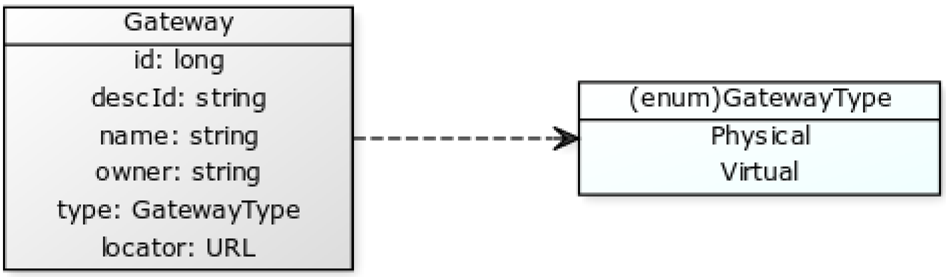


Figure 13 D2D UML class diagram

2.3.2.2 N2N Layer

Under the 'Network' tab, the web application offers the set of features related to the N2N layer. The following diagram depicts its model.

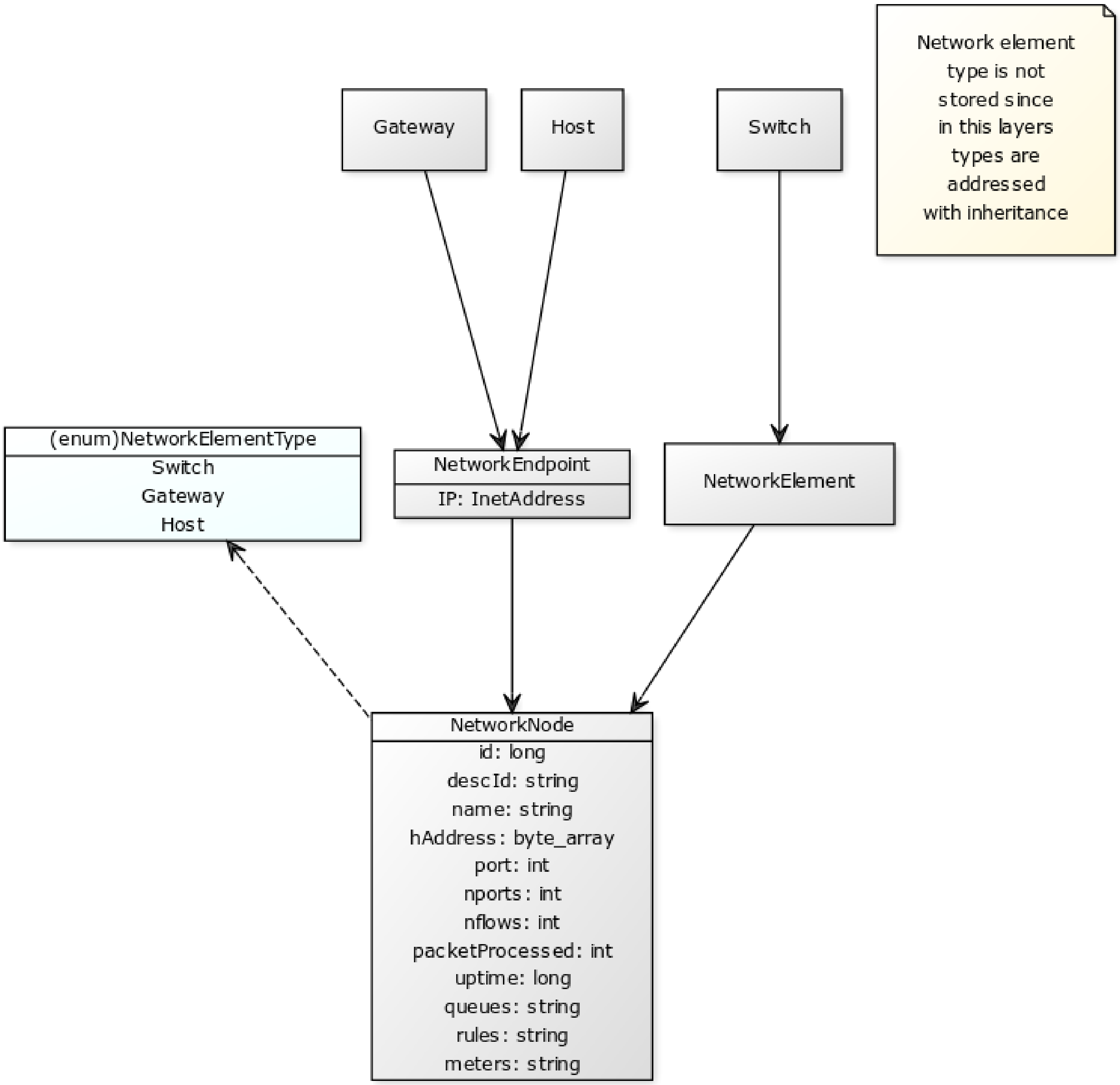


Figure 14 N2N UML class diagram

2.3.2.3 MW2MW Layer

The mechanisms exposed by the MW2MW layer are available under the tab 'Platforms', which has the following classes. It is remarkable the use of two kinds of users (see sect. 2.3.2.6), one to manage the INTER-IoT session, access to different tabs, etc. and a second one to share credentials between the connected platform and the framework. The enumeration of platform types includes all the platforms analysed for the project (not necessarily integrated) and those used/integrated by the third parties from the open call.

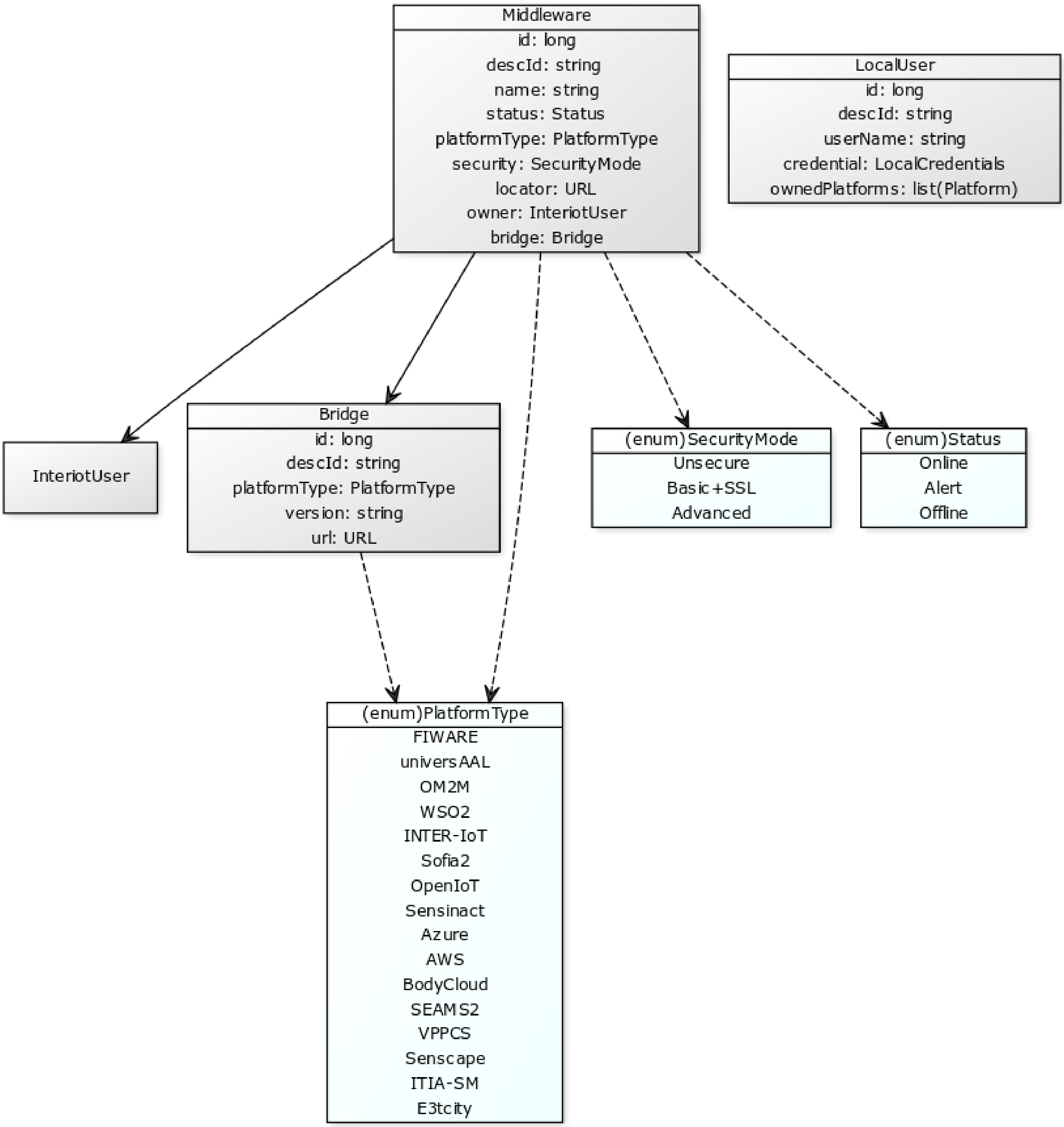


Figure 15 MW2MW UML class diagram

2.3.2.4 AS2AS Layer

The following diagram shows the main classes and relations under the ‘Services’ tab.

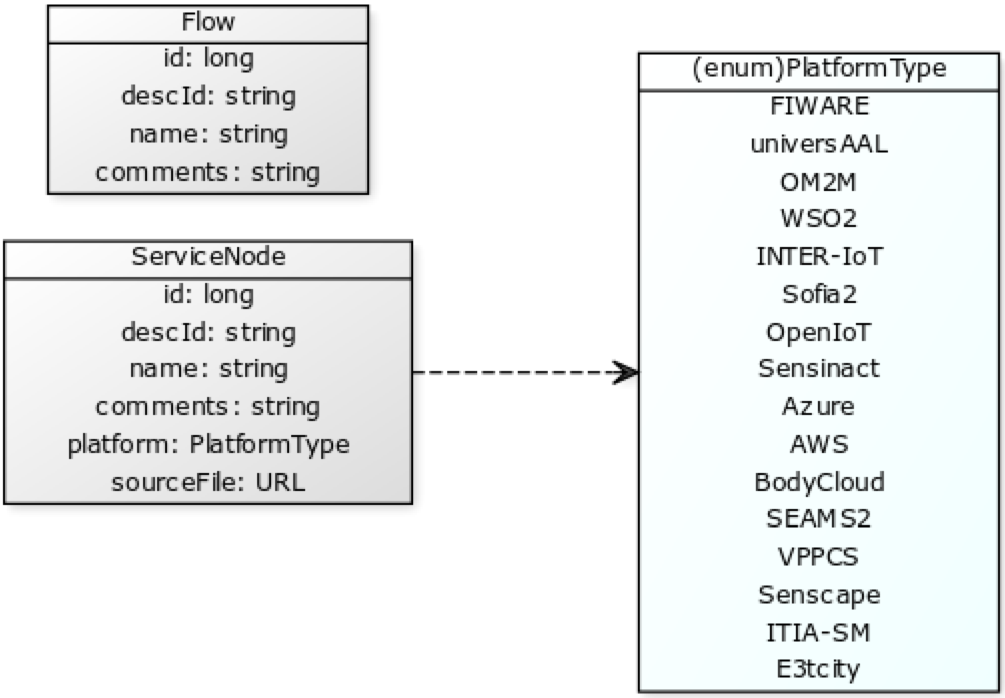


Figure 16 AS2AS UML class diagram

2.3.2.5 DS2DS Layer

The semantics layer model for INTER-FW application is depicted in the following diagram.

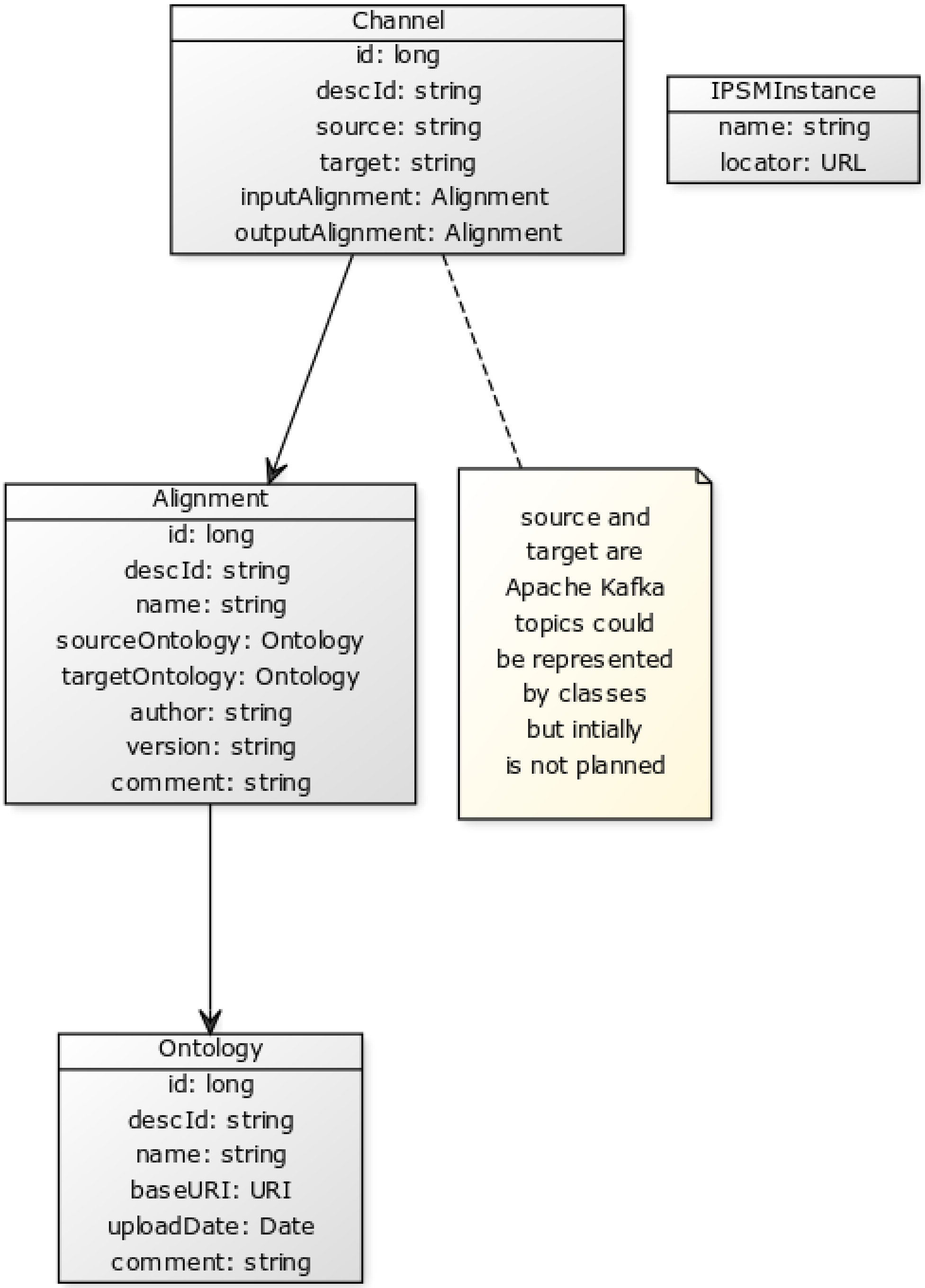


Figure 17 DS2DS UML class diagram

2.3.2.6 User and credentials management

The user management is a key element in INTER-FW, on one hand, the session of the INTER-IoT users, developers, applications connected to API, etc. need to be managed from a central system. This includes the control of the data by the owners of the connected platforms and nodes. On the other hand, there is a need to manage the different authentication and authorisation systems of each connected node.

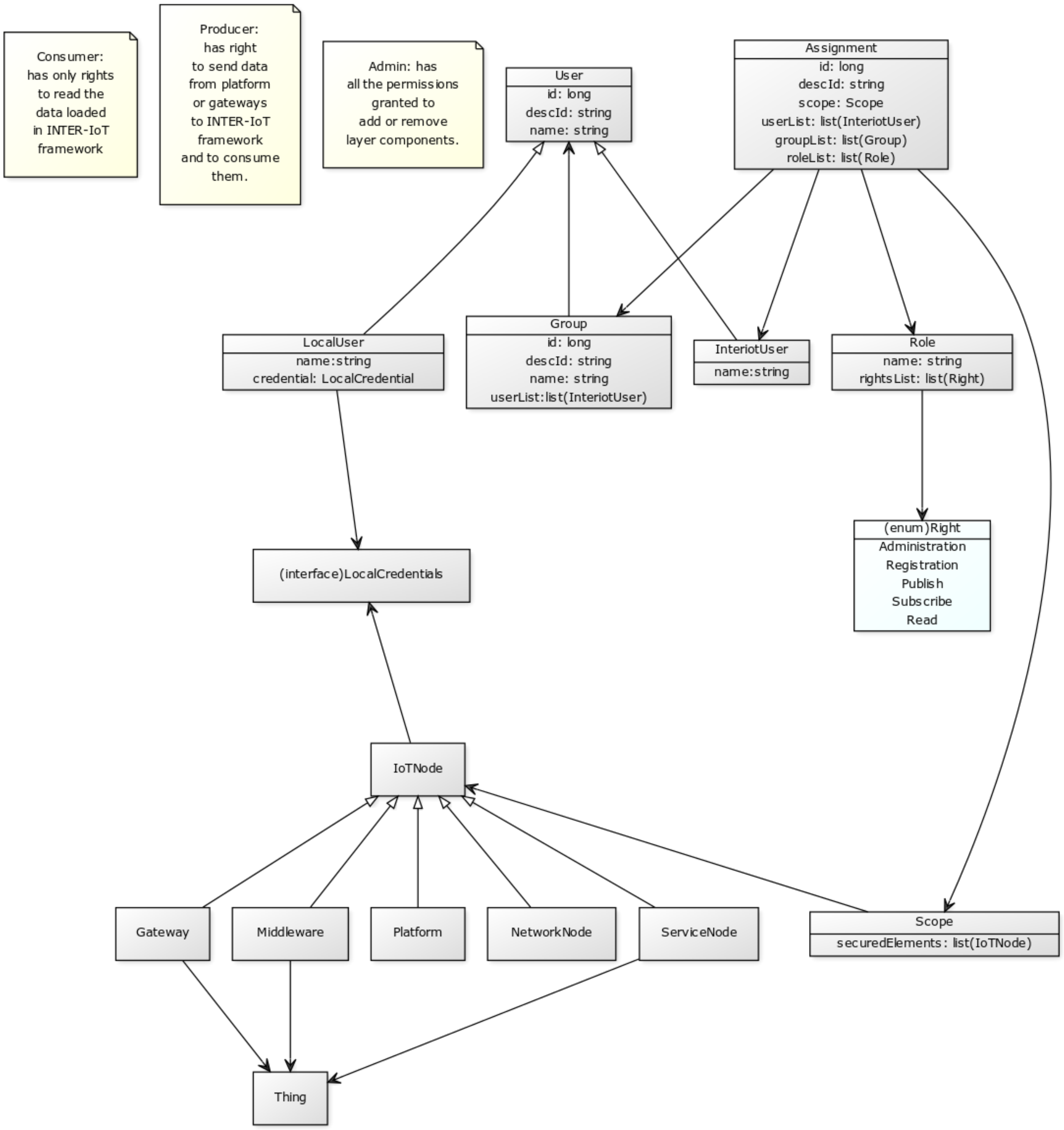

Consumer: has only rights to read the data loaded in INTER-IoT framework
Producer: has right to send data from platform or gateways to INTER-IoT framework and to consume them.
Admin: has all the permissions granted to add or remove layer components.
User
id: long
descId: string
name: string
Assignment
id: long
descId: string
scope: Scope
userList: list(InteriotUser)
groupList: list(Group)
roleList: list(Role)
LocalUser
name:string
credential: LocalCredential
Group
id: long
descId: string
name: string
userList:list(InteriotUser)
InteriotUser
name:string
Role
name: string
rightsList: list(Right)
(enum)Right
Administration
Registration
Publish
Subscribe
Read
(interface)LocalCredentials
IoTNode
Gateway
Middleware
Platform
NetworkNode
ServiceNode
Scope
securedElements: list(IoTNode)
Thing

Figure 18 User management UML class diagram

2.3.3 Technologies

The backend of the application is developed with Node.js[32] as web application framework, Mongo.db[33] to store unstructured data and Mogoose[34] for object modelling.

[32] https://nodejs.org/en/

[33] http://mongodb.org/

[34] http://mongoosejs.com/

2.4 Front-end analysis

The development of a client-side application is called front-end. It is the presentation layer where users can see and interact with content in a user-friendly interface (e.g. a website or a mobile application). It is usually developed as a mixture of Hypertext Markup Language (HTML), Cascading Style Sheet (CSS), JavaScript (JS) and ancillary libraries.
This section is divided in three subsections about the requirements identified in the task 2.3, the tools used for designing the front-end, and the tools used for developing it.

2.4.1 Requirements

A set of requirements were identified related to the front end of the INTER-FW in the initial stages of the project. Since the definition of the layers, the scenarios and, in some cases, the global functionality have been slightly modified, some extra requirements and modifications have been added at the end to cover all the features planned.

The complete list of requirements affecting the back end is:

Title	Id
Extensibility	10
Platform independency	14
System security	27
System privacy	28
Heterogeneous information representation	42
API for third-party developers	47
API for data publication	51
API REST	52
Auditability and accountability	58
Provision of authentication credentials	63
Confidentiality	69
Easy-to-use user interface	70
Application response time	71
Users manage how their public data is seen	77
Trust management	100
Adaptability	125
Provides a sensor-level interface	259
Manage user permission	260
A user knows its permissions	261
Manages group-based permissions	262
Access to personal data needs to be previously authorized	263
API allows create/update/remove users	264
API allows device declaration and configuration	265
API allows resources/capabilities discovery	266
API allows historic data query	268
API allows subscription to data streams/queues	270
Requests filtering	280
Map publish/subscription between platforms in the middleware	282
Manage a sensor or actuator	283

Table 2 Frontend related requirements

To have a complete description of these requirements, see

Annex I User requirements.

2.5 API analysis

2.5.1 API Requirements Analysis

During the initial stages of the project a set of functional and non-functional requirements related to INTER API implementation were identified. In total 13 requirements have been identified, of which only one has a priority level “Should Have” [req. 268] and all others have a priority level “Must Have”. This shows that prospective users of the system put a high priority on openness and extensibility of the platform.

There is a strong need for centralized access to all Inter-IoT functionalities [req. 47 and 199]. These general requirements are further supported by requirements to access specific INTER Layer interoperability components: Device Layer Interoperability [req. 243 and 265], Network Layer Interoperability [req. 226], Middleware Layer Interoperability [req. 237 and 266], Application Service Layer Interoperability [req. 51] and Data and Semantics Layer Interoperability [req. 51].

Non-functional requirements specifically request to provide a REST interface [req. 52] and to provide methods for interoperability with proprietary systems [req. 86].

Most of the requirements related to APIs are related to a unified way of accessing Inter Layer functionalities. However, they are tightly related to tasks of this work package because we have a very clear need by prospective stakeholders/users of the system to have a unified, well defined, consistent, flexible and controlled access to all Inter Layer components, regardless of respective implementations. For this reason the **scope of the first development iteration is the detailed specification for integration of Inter Layer APIs** into INTER FW. In addition to pure API integration, already in this phase extensibility in the sense of creating complex APIs (scripting), basic user management and API publication (“marketplace”) will be provided. This will also support the development of INTER FW core components.

In the second iteration, to be provided in D4.4./D4.6, a comprehensive set of APIs for the exposure of INTER FW functionalities will be exposed. It will include user/access management APIs [req. 264], SDK support [req. 47, 199 and 86] and other.

The complete list of requirements affecting the INTER API is provided in the table below. The first four columns (Title, Id, Functional / non-functional., Module, Type) are taken from D2.3, while the last column, “Affected components” is the identification of Inter-IoT components that provide the underlying functionality.

Title	**Id**	**functional / non-functional**	**Module**	**Type**	**Affected components**
API for third-party developers	47	F	INTER-FW	API	All (INTER FW, INTER LAYER)
API for data publication	51	F	INTER-FW	API	AS2AS, IPSM
API REST	52	N	INTER-FW	API	INTER API
API for proprietary systems interoperate with other systems	86	N	INTER-FW	Interoperability	INTER-LAYER,

					INTER-FW, INTER-API
IDEs and APIs for rapid new applications development	199	N	General	API	All (INTER FW, INTER LAYER)
API for network services	226	F	INTER-LAYER	Interoperability	NW2NW
API Middleware for interoperability between different platforms	237	F	INTER-LAYER	Middleware	MW2MW
Gateway access API	243	F	INTER-LAYER	Interface	D2D
API allows create/update/remove users	264	F	INTER-FW	API	INTER FW
API allows device declaration and configuration	265	F	INTER-FW	API	D2D
API allows resources/capabilities discovery	266	F	INTER-FW	API	MW2MW
API allows historic data query	268	F	INTER-FW	API	AS2AS, INTER-API
API allows subscription to data streams/queues	270	F	INTER-FW	API	INTERMW, INTER-FW

Table 3 API Requirements analysis

To have a complete description of these requirements, see Annex I.

2.5.2 First iteration: Inter-Layer exposed API

The aim of the API analysis of Inter-Layer components is to understand their exposed functionalities, to define requirements that will drive the development of the INTER FW API (INTER-API), as well as to create a set of supporting tools for programming and managing these. Inter-Layer APIs encompass features such as creation of channels within the IPSM, registration of a new platform and creation of a list of physical gateways, registered within INTER-IoT. INTER-API will thus expose different INTER-IoT interoperability layers as a unified set of securely managed REST APIs.

This approach will expose INTER-FW functionalities to other IoT platforms and application domains. In the first instance, the INTER FW core engine (back-end) and web application (front-end) will use INTER API as a main point of access to Inter-Layer features. These APIs will allow an easy mechanism to define new integration patterns at different interoperability layers creating the basis for the development of new products.

The following Inter-Layer components are exposing their key functionality through a set of REST APIs:

- **Device Layer Interoperability (D2D) REST API:** Provision of methods for physical and virtual gateway management, as well as management of other devices. Using this API, one can manage gateway and device configurations (using key-value pairs, KVP), start and stop the devices and physical gateways, as well as get the list of platforms, connected to individual virtual gateways.
- **Network Layer Interoperability REST API:** Provision of methods for interoperability of components at network layer. The API allow monitoring and management of switches, flows, ports, roles and queues.

- **Middleware Layer Interoperability (INTERMW) REST API:** INTERMW provides an API for platform management, as well as management of individual things. With it, we can register and unregister platforms as well as individual things connected to these platforms, subscribe to and unsubscribe from thing updates, discover things within the platforms and query about the status of these.
- **Application Service Layer Interoperability (AS2AS) REST API:** The core component of this layer is a customised deployment of Node-RED, a flow-based programing tool for the Internet of Things. In the Inter-IoT context, each node-RED instance represents an execution flow (using its available nodes). If a user wants to execute two or more flows (two or more instances of Node-RED), he needs two or more instances of Docker running Node-RED for each flow. Each Node-RED instance only provides the nodes corresponding to the platforms which the user is going to access. In order to allow flow management functionalities, Node-RED flow management API will be exposed through INTER API.
- **Data and Semantics Layer Interoperability (DS2DS):** Provision of methods for management of an IPSM (Inter Platform Semantic Mediator) instance configuration including translation channels (listing, creation and deletion) and alignments (listing, upload, deletion and retrieval). Note that alignments define correspondences between ontologies that are used in the translation process within a translation channel. Additional REST API is exposed for an independent INTER-FW Semantic Repository component that provides access to centrally stored alignments and ontologies that can be reused for an IPSM configuration, or accessed for revision. The exposed API methods allow management of ontologies (listing, upload, deletion and retrieval) and alignments (listing, upload, deletion and retrieval).

In addition to a unified access to single Inter-Layer components, the proposed solution will allow API mashups/scripting across all layers, this providing more complex functionalities.

2.5.3 Second iteration: Security and configuration management

There are several INTER FW supporting services that should be exposed through the INTER API mechanism. The definition and implementation of those has been deferred to the second iteration (D4.3 + D4.5) as the practical implementation of those APIs depends on adapted solutions for the implementation of security and configuration components to be identified and elaborated in the following sections:

- User and credentials management (2.3.2.6)
- API Key/Token generation (2.2.2)
- Security Management (3.4)
- Configuration (3.5.8)

Once the solution is in place, standard API definitions will be selected for those operations. In the design phase of above mentioned service particular attention is devoted the availability of standard REST API definitions. Standard definition helps API consumers understanding the purposes of these interfaces, eases the access to the complete set of resources and make easier document and understand the underpinning technology. The standardization of the API fosters seamless integrations with less boilerplate and more focus on features and performance.

3 INTER-FW Design

3.1 Overview

Once the analysis is done, a set of high-level functional blocks have been identified. These functional blocks describe the main features of the INTER-FW and INTER-API. Some of them can be implemented in single components, but most of them must be split down into components performing more specific tasks. This section describes the architecture of these components, which is based in the previous work performed in tasks 4.1 and 4.2 and reported in D4.1 (M12). In the section 3.2, it is explained how the INTER-IoT Reference Architecture has helped to define the components which form part of the INTER-FW solution. Sections 3.3 describe in detail the components which are part of the solution and their relation; 3.4 gives more detail in the fundamental security interactions among INTER-FW, INTER-LAYER and other modules in INTER-IoT. Finally, 3.5 reflects these backend designs in a complete set of wireframes to show how the UX will look like.

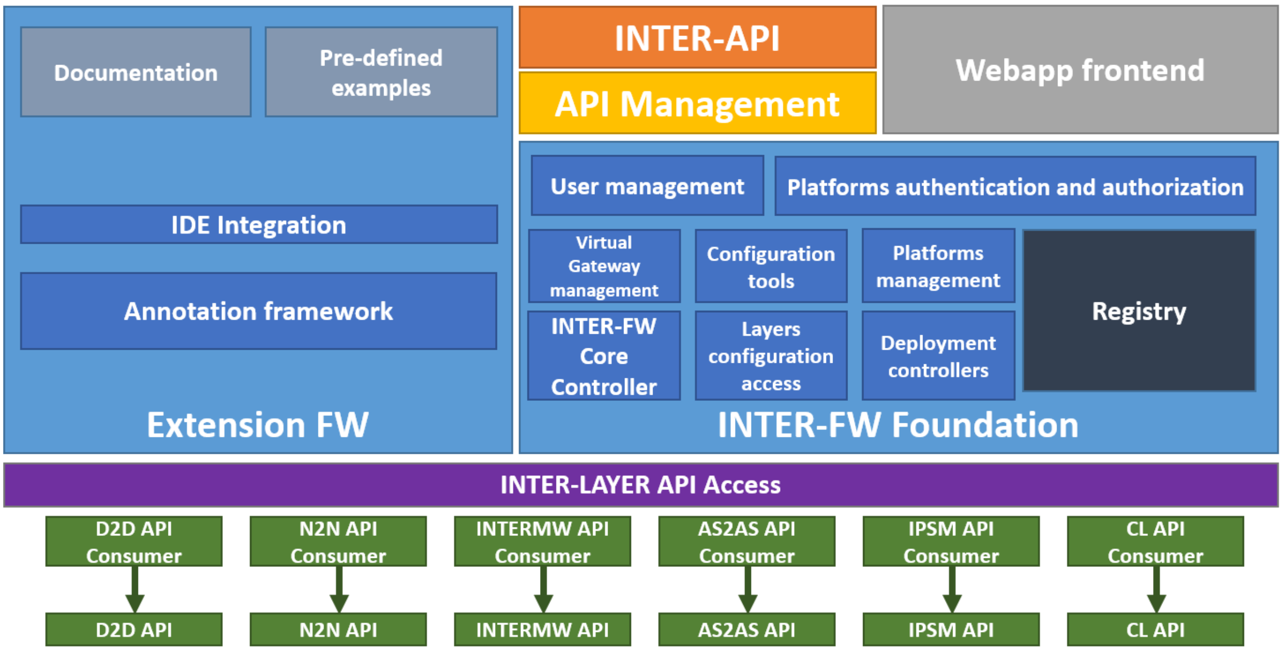


Figure 19 High-level functionalities architectural approach in INTER-FW

3.2 INTER-IoT RA Instantiation

In Deliverable D4.1, an Initial Reference IoT Platform Meta-Architecture and Meta-Data Model has been produced. In this deliverable, INTER-IoT has defined a Reference Model (RM) or “meta-model” for IoT Platforms Interoperability and a Reference Architecture (RA) (through an Architectural Reference Model-ARM) defined based on the RM as well. This work is currently under progress towards a final version of the deliverable. This document has a strong basis on the works done in IOT-A EU Project and a deep analysis of 16 heterogeneous IoT platforms carried out in the INTER-IoT project

This RA has been used for the design of the INTER-LAYER architecture, and both, the INTER-IoT RA and the INTER-LAYER architecture (specifically the API design of the different interoperability layers) has been the foundation for the design of the INTER-FW architecture.

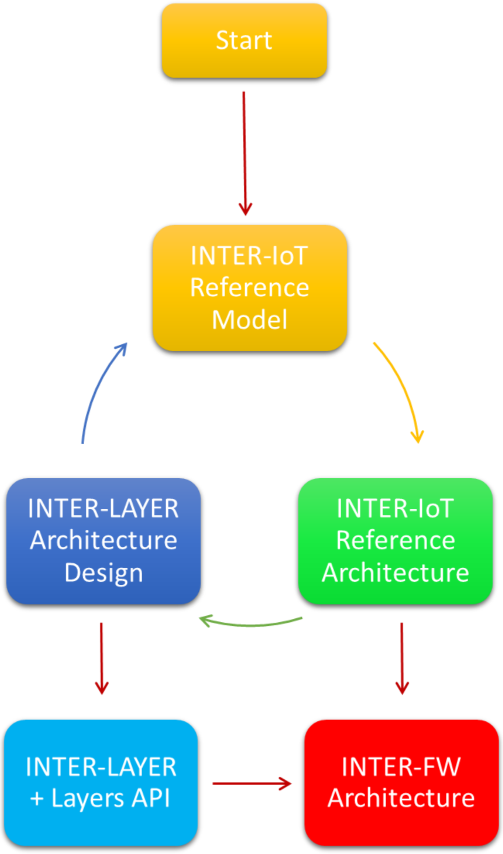

Start
INTER-IoT Reference Model
INTER-LAYER Architecture Design
INTER-IoT Reference Architecture
INTER-LAYER + Layers API
INTER-FW Architecture

Figure 20 Process followed for generating the INTER-FW architecture.

The APIs of the different layers has been used for designing the communication between the INTER-FW and each of the interoperability layers, so it's at a low level of the design. Instead, the INTER-IoT RA has been the starting point for the design of the INTER-FW architecture.

According to OASIS[35] definition:

> A reference architecture models the abstract architectural elements in the domain of interest independent of the technologies, protocols, and products that are used to implement a specific solution for the domain.

According to the same organism:

> A reference architecture is not a concrete architecture; i.e., depending on the requirements being addressed by the reference architecture, it generally will not completely specify all the technologies, components and their relationships in sufficient detail to enable direct implementation.

The INTER-IoT RA was designed for the interoperability of IoT Platforms. This was one of the objectives of the INTER-IoT project (see Objective 2). The explanation of OASIS about a reference architecture and a concrete architecture is exactly what we have done for the INTER-FW: to create an instantiation of the INTER-IoT RA to the specific needs of the INTER-FW.

[35] http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf

3.2.1 INTER-IoT RA instantiation

The starting point for this instantiation has been the Functional View for the INTER-IoT Reference Architecture that is depicted in the following figure:

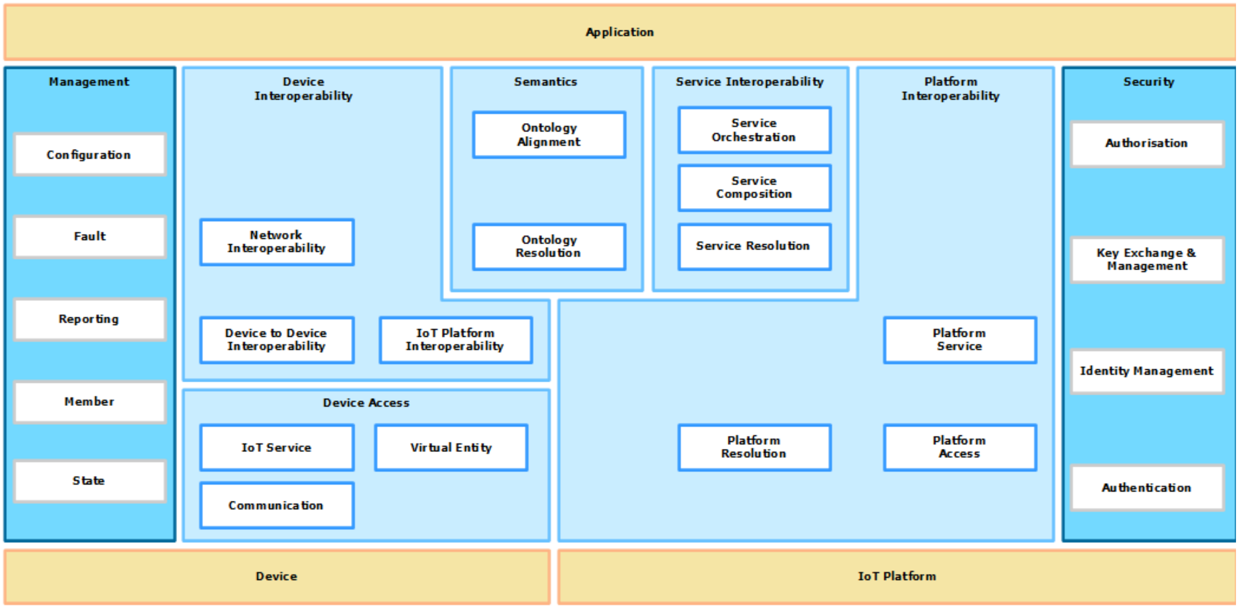


Figure 21 Functional-decomposition viewpoint of the INTER-IoT Reference Architecture

From these Functional Groups (FGs – the big blocks) and Functional Components (FCs – the inner components of each FG), some of them have been identified as relevant for the INTER-FW according to its defined requirements.

The FGs that are involved in the instantiation of the INTER-FW architecture can be seen in the following picture:

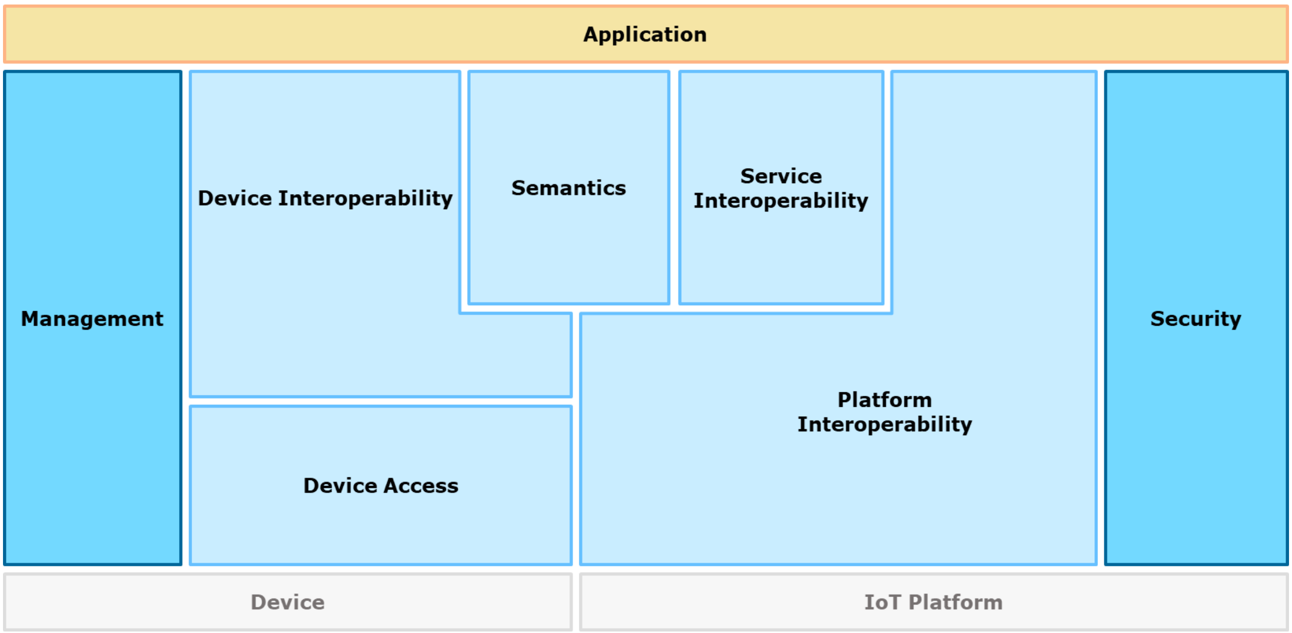


Figure 22: Functional Groups of the INTER-IoT Reference Architecture involved in the INTER-FW

The Device FG and the IoT Platform FG are clearly out of the scope of a framework that must provide interoperability. The access and communication with devices and IoT Platforms is achieved through the Device Access FG and the Platform Interoperability FG, not being the Device FG and the IoT Platform FG part of the INTER-FW. The Device Interoperability FG is also considered out of the scope of the INTER-FW, as all its functionality is within the Device-to-device and network-to-network layers, and all their features are exposed through the virtual gateway, which lies in the Device Access FG.

The role of the rest of FGs in INTER-FW is described below:

The **Application FG**, which aims at modelling the users of INTER-IoT, mainly applications or systems willing to access the different platforms, is conceived in the INTER-FW as the provision of an API (INTER-API), which will provide access to the full INTER-FW and all the interoperability capabilities underneath it in the different layers.

So, an API will be designed, as it is described in section 4 of this document.

Apart from this API, the INTER-FW needs to fulfil other requirements related with the Application FG. A Web Console has been analysed in section 2.2 of this document. It is the presentation layer where users can see and interact with content in a user-friendly interface (e.g. a website or a mobile application).

The **Management FG** is together with the **Security FG**, the two more relevant FGs of the architecture instantiation of the INTER-FW. Many of the features of the rest of the groups, although not advisable, could be accessed directly with the APIs of the different interoperability layers; but the features of the Management FG and the Security FG give meaning to the INTER-FW.

The Management FG is needed in INTER-FW to fulfil some requirements like these ones:

- Scalability. Computing resources [3].
- Portability [132].
- Future-proof [278].
- Auditability and Accountability [58].
- QoS Integration [142].
- Users manage how their public data is seen [77].
- Stores recent data for recovery [272].
- Stores system status for recovery [273].

The Security FG is a key group of the INTER-FW architecture, which is responsible for ensuring all the security aspects involved in the interoperability of IoT Platforms. The security in our realm has two faces:

- Management of the security aspects related to the connection with underlying IoT Platforms. This implies to accomplish with the different security features that the platforms require. INTER-FW will need to tackle the user authentication for connecting to a platform, the authorisation management (e.g. use of authentication tokens) and the encryption of some communications.
- Management of the internal security of INTER-IoT. The connection to INTER-IoT must be secured, with appropriate authentication capabilities, and authorisation management. The identity of each user must be preserved, so much for keeping the identification until the IoT Platform, as to keep track of the anonymization when talking with the IoT Platforms. This internal security also implies the permission assignment to specific IoT Platforms and its resources.

The Security FG is needed in INTER-FW to fulfil some requirements like these ones:

- Access to personal data needs to be previously authorized [263].
- Confidentiality [69].
- Data provenance [98].
- Manage user permission [260].
- Manages group-based permissions [262].
- API allows create/update/remove users [264].
- Users manage how their public data is seen [77].
- A user knows its permissions [261].

The **Device Access FG** will be instantiated only with the components that give access to the D2D / N2N Layers of INTER-LAYER, providing access to the gateway and more specifically to the virtual gateway.

The **Semantics FG** will also be used as the access to the Data & Semantics Interoperability Layer, although the management of ontologies for incoming requests/responses and ontology definition for connections to the different IoT Platforms must also be handled.

The **Service Interoperability FG** plays a relevant role in the INTER-FW, as there are important features to provide apart from the pure access to services from IoT Platforms. INTER-FW will provide the execution environments for orchestrating services from different IoT Platforms with user independence among environments.

Finally, the **Platform Interoperability FG** will be used for platform resolution and cataloguing the IoT Platforms that are available at a specific deployment of INTER-IoT.

Inside each Functional Group, some Functional Components have been identified as necessary for INTER-FW. Please note that other FCs are not discarded in INTER-IoT, but addressed by other products of INTER-IoT like INTER-LAYER.

3.2.2 INTER-FW Functional Components

The FCs within each FG that are involved in the instantiation of the INTER-FW architecture can be seen in the following picture:

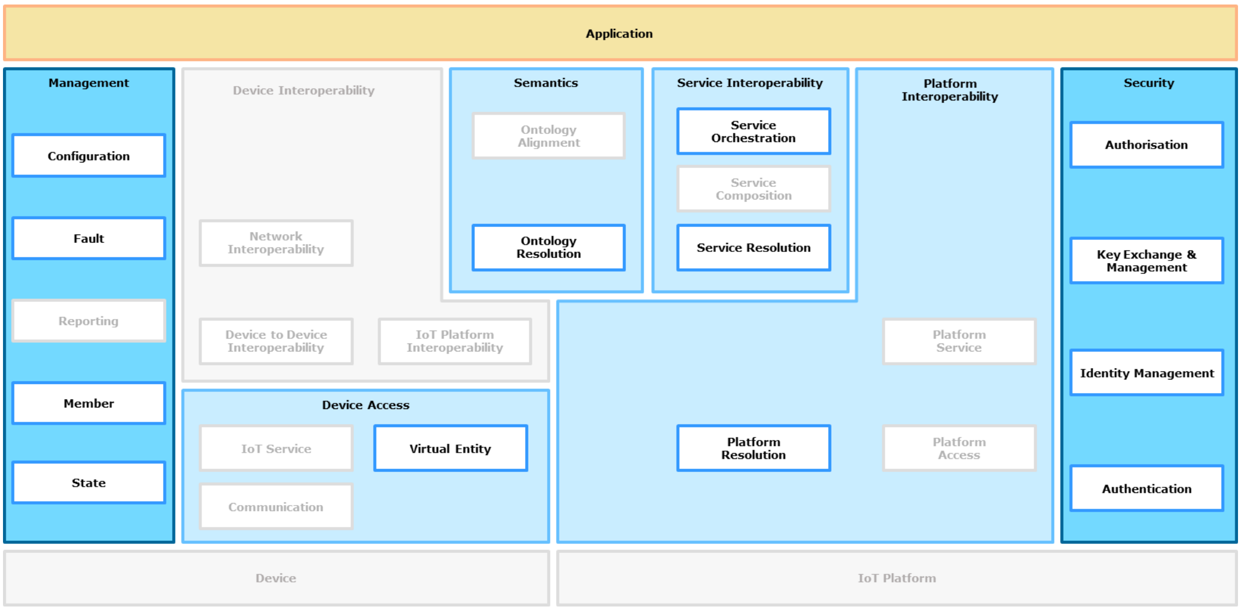


Figure 23: Functional Components of the INTER-FW architecture according to the INTER-IoT Reference Architecture.

A more detailed description of each FC and their traced requirements is described below. Please, consider that the INTER-FW exposes all the functionality of INTER-IoT, but does not implement it, what is primarily done by INTER-LAYER. In this document, only the features implemented by INTER-FW and the necessary functionality to manage and provide access to INTER-LAYER are designed.

For each FC, a list of analysed use cases and sequence diagrams as defined in the INTER-FW analysis (section 2.2 of this document) are listed. Also, a pre-candidate definition of architectural components is made for each FC.

3.2.2.1 Application FG

The Application FG does not include a pre-defined set of FCs in INTER-IoT Reference Architecture, but INTER-FW will provide some features of the Application FG so as to give a full-fledged solution to INTER-IoT uses. The new Functional Components designed for the INTER-FW are, thus, described below.

Functional Component	Description
INTER-API	Provides access to all the functionality of INTER-IoT through an API. To be used by system integrators and developers.
List of supported Use Cases	2.2.1. Log In 2.2.2. API Key/Token generation 2.2.3. Add a new platform 2.2.4. Update platform configuration 2.2.5. Remove a platform 2.2.6. Add or register a virtual instance of a gateway 2.2.7. Update configuration of a virtual instance of a gateway 2.2.8. Remove a virtual instance of a gateway 2.2.14. List IPSM instances 2.2.15. Register IPSM instance 2.2.16. Unregister IPSM instance 2.2.17. List alignments in IPSM

	2.2.18. List alignments in Semantic Repository 2.2.19. Add a new alignment to Semantic Repository 2.2.20. Add a new alignment to IPSM 2.2.21. Delete an alignment from IPSM 2.2.22. Delete an alignment from Semantic Repository 2.2.23. Delete an ontology from Semantic Repository 2.2.24. Get an alignment from IPSM 2.2.25. View an alignment from IPSM 2.2.26. List active IPSM translation channels 2.2.27. Create a new translation channel in IPSM 2.2.28. Delete a translation channel in IPSM 2.2.29. List ontologies in Semantic Repository 2.2.30. View an ontology in Semantic Repository 2.2.31. View an alignment in Semantic Repository 2.2.32. View an alignment history in Semantic Repository 2.2.33. Get an ontology from Semantic Repository 2.2.34. Get an alignment from Semantic Repository 2.2.35. Add a new ontology to Semantic Repository 2.2.36. Add a new alignment to Semantic Repository 2.2.40. Change user data
Architectural Components	• *API controller*: it receives API calls from the different components of the application and forwards them to the specific API of the layer. • *API request validator*: it checks the fields, verbs and paths of the API requests.

Table 4 INTER-API Frontend in the INTER-IoT RA

Functional Component	Description
Web App Frontend	The presentation layer where users can see and interact with content in a user-friendly interface (e.g. a website or a mobile application)
List of supported Use Cases	2.2.1. Log In 2.2.3. Add a new platform 2.2.4. Update platform configuration 2.2.5. Remove a platform 2.2.6. Add or register a virtual instance of a gateway 2.2.7. Update configuration of a virtual instance of a gateway 2.2.8. Remove a virtual instance of a gateway 2.2.9. Connect services with AS2AS tool 2.2.10. Visualize INTER-IoT status 2.2.11. List permissions 2.2.12. Manage platform permissions 2.2.13. Revoke permissions 2.2.14. List IPSM instances 2.2.15. Register IPSM instance 2.2.16. Unregister IPSM instance 2.2.17. List alignments in IPSM 2.2.18. List alignments in Semantic Repository 2.2.19. Add a new alignment to Semantic Repository 2.2.20. Add a new alignment to IPSM 2.2.21. Delete an alignment from IPSM

	2.2.22. Delete an alignment from Semantic Repository 2.2.23. Delete an ontology from Semantic Repository 2.2.24. Get an alignment from IPSM 2.2.25. View an alignment from IPSM 2.2.26. List active IPSM translation channels 2.2.27. Create a new translation channel in IPSM 2.2.28. Delete a translation channel in IPSM 2.2.29. List ontologies in Semantic Repository 2.2.30. View an ontology in Semantic Repository 2.2.31. View an alignment in Semantic Repository 2.2.32. View an alignment history in Semantic Repository 2.2.33. Get an ontology from Semantic Repository 2.2.34. Get an alignment from Semantic Repository 2.2.35. Add a new ontology to Semantic Repository 2.2.36. Add a new alignment to Semantic Repository 2.2.40. Change user data 2.2.41. Set up QoS requirements 2.2.42. Visualize SDN topology changes
Architectural Components	• *Inter-fw frontend*: The collection of views that will be rendered to the users to interact with them. • *Inter-fw controller*: Process and forward data requests to the model, ask for fields, credentials, etc. • *User registry*. a store of the INTER-IoT users. • *Nodes registry*: a storage point for the registered nodes: platforms, gateways, services, etc. • *Node configuration validator*: it checks that the configuration submitted for each node is correct, before calling the associated API.

Table 5 Web Console Frontend in the INTER-IoT RA

3.2.2.2 Management FG

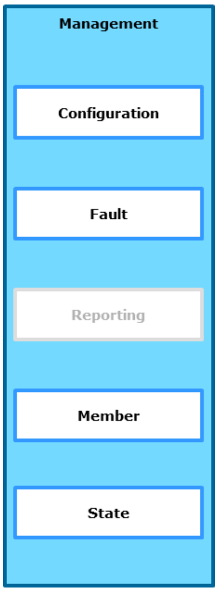


Figure 24: Functional Components of the Management FG instantiated for INTER-FW concrete architecture.

The list of instantiated FCs of the Management FG is the following:

Functional Component	Description
Configuration	Responsible for initialising the system configuration such as gathering and storing configuration about IoT Platforms, users and so on.
List of supported Use Cases	2.2.3. Add a new platform 2.2.4. Update platform configuration 2.2.5. Remove a platform 2.2.6. Add or register a virtual instance of a gateway 2.2.7. Update configuration of a virtual instance of a gateway 2.2.8. Remove a virtual instance of a gateway 2.2.15. Register IPSM instance 2.2.16. Unregister IPSM instance 2.2.19. Add a new alignment to Semantic Repository 2.2.20. Add a new alignment to IPSM 2.2.21. Delete an alignment from IPSM 2.2.22. Delete an alignment from Semantic Repository 2.2.23. Delete an ontology from Semantic Repository 2.2.40. Change user data
Architectural Components	• *User registry*. a store of the INTER-IoT users. • *Nodes registry*: a storage point for the registered nodes: platforms, gateways, services, etc. • *Node configuration validator*: it checks that the configuration submitted for each node is correct.

Table 6 Web Configuration FC use cases and analysis

Functional Component	Description
Fault	The goal of the Fault FC is to identify, isolate, correct and log faults that occur in the IoT system.
List of supported Use Cases	No specific use cases. It's a transversal component used by all the inner INTER-FW components.
Architectural Components	• *Inter fw log handler*. Central management of INTER FW logs.

Table 7 Fault FC use cases and analysis

Functional Component	Description
Member	Responsible for the management of the membership and associated information of any relevant entity.
List of supported Use Cases	2.2.3. Add a new platform 2.2.4. Update platform configuration 2.2.5. Remove a platform 2.2.6. Add or register a virtual instance of a gateway 2.2.7. Update configuration of a virtual instance of a gateway 2.2.8. Remove a virtual instance of a gateway 2.2.40. Change user data

Architectural Components	• *API manager*. It features a wide set of operations including API requests accounting, metrics, versioning, security, etc. • *Nodes registry*: a storage point for the registered nodes: platforms, gateways, services, etc.

Table 8 Member FC use cases and analysis

Functional Component	Description
State	It monitors the state of the INTER-FW and offers a view of the current state to the INTER-FW administrators. It is also responsible for measuring and eventually displaying the performance of the usage of the API.
List of supported Use Cases	2.2.10. Visualize INTER-IoT status
Architectural Components	• *Nodes registry*: a storage point for the registered nodes: platforms, gateways, services, etc. • *API manager*. It features a wide set of operations including API requests accounting, metrics, versioning, security, etc.

Table 9 State FC use cases and analysis

3.2.2.3 Security FG

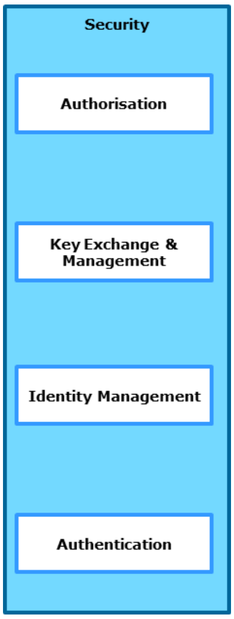


Figure 25: Functional Components of the Security FG instantiated for INTER-FW concrete architecture.

The list of instantiated FCs of the Management FG is the following:

Functional Component	Description
Authorisation	It is a front end for managing policies and performing access control decisions based on access control policies. This access control decision can be called whenever access to a restricted resource is requested

List of supported Use Cases	2.2.3. Add a new platform 2.2.4. Update platform configuration 2.2.5. Remove a platform 2.2.6. Add or register a virtual instance of a gateway 2.2.7. Update configuration of a virtual instance of a gateway 2.2.8. Remove a virtual instance of a gateway 2.2.9. Connect services with AS2AS tool 2.2.10. Visualize INTER-IoT status 2.2.11. List permissions 2.2.12. Manage platform permissions 2.2.13. Revoke permissions 2.2.14. List IPSM instances 2.2.15. Register IPSM instance 2.2.16. Unregister IPSM instance 2.2.17. List alignments in IPSM 2.2.18. List alignments in Semantic Repository 2.2.19. Add a new alignment to Semantic Repository 2.2.20. Add a new alignment to IPSM 2.2.21. Delete an alignment from IPSM 2.2.22. Delete an alignment from Semantic Repository 2.2.23. Delete an ontology from Semantic Repository 2.2.24. Get an alignment from IPSM 2.2.25. View an alignment from IPSM 2.2.26. List active IPSM translation channels 2.2.27. Create a new translation channel in IPSM 2.2.28. Delete a translation channel in IPSM 2.2.29. List ontologies in Semantic Repository 2.2.30. View an ontology in Semantic Repository 2.2.31. View an alignment in Semantic Repository 2.2.32. View an alignment history in Semantic Repository 2.2.33. Get an ontology from Semantic Repository 2.2.34. Get an alignment from Semantic Repository 2.2.35. Add a new ontology to Semantic Repository 2.2.36. Add a new alignment to Semantic Repository 2.2.40. Change user data 2.2.41. Set up QoS requirements 2.2.42. Visualize SDN topology changes
Architectural Components	• *Auth proxy*: it receives authentication and authorization requests and replies if the action requested is allowed for a given user. • *Auth server*: it grants access to users to the web environment and API requests. • *Node credentials manager*: It stores the credentials for the nodes and also process authentication and authorization requests to the nodes.

Table 10 Authorisation FC use cases and analysis

Functional Component	Description
Authentication	It is involved in user and service authentication. It checks the credentials provided by a user, and, if valid, it returns an assertion as result.

List of supported Use Cases	2.2.1. Log In 2.2.2. API Key/Token generation
Architectural Components	• *Inter-fw user manager*: it processes and forwards user related data. • *Auth proxy*: it receives authentication and authorization requests and replies if the action requested is allowed for a given user. • *Auth server*: it grants access to users to the web environment and API requests. • *Node credentials manager*: It stores the credentials for the nodes and also process authentication and authorization requests to the nodes.

Table 11 Authentication FC use cases and analysis

Functional Component	Description
Key Exchange & Management	It is involved to enable secure communications of the INTER-API with INTER-IoT users and among IoT Platforms with different owners, keeping the confidentiality.
List of supported Use Cases	2.2.1. Log In 2.2.2. API Key/Token generation
Architectural Components	• *Inter-fw user manager*: it processes and forwards user related data. • *API manager*. It features a wide set of operations including API requests accounting, metrics, versioning, security, etc.

Table 12 Key Exange and Management FC use cases and analysis

Functional Component	Description
Identity Management	It addresses privacy questions by issuing and managing pseudonyms and accessory information to trusted subjects so that they can operate (use or provide services) anonymously.
List of supported Use Cases	2.2.1. Log In 2.2.2. API Key/Token generation
Architectural Components	• *Inter-fw user manager*: it processes and forwards user related data. • *Node credentials manager*: It stores the credentials for the nodes and also process authentication and authorization requests to the nodes.

Table 13 Identity Management FC use cases and analysis

3.2.2.4 Device Access FG

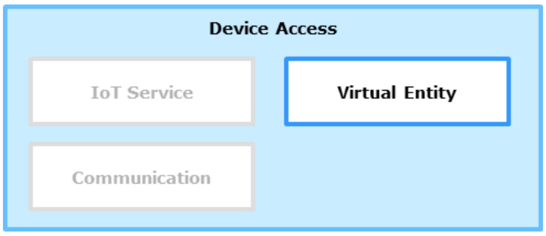


Figure 26: Functional Components of the Device Access FG instantiated for INTER-FW concrete architecture.

The list of instantiated FCs of the Device Access FG is the following:

Functional Component	Description
Virtual Entity	The Virtual Entity FC allows the interaction with an IoT Platform on the basis of Virtual Entities. It provides the interaction with the Virtual Gateway of INTER-LAYER for the D2D and N2N interoperability layers.
List of supported Use Cases	2.2.3. Add a new platform 2.2.4. Update platform configuration 2.2.5. Remove a platform 2.2.6. Add or register a virtual instance of a gateway 2.2.7. Update configuration of a virtual instance of a gateway 2.2.8. Remove a virtual instance of a gateway
Architectural Components	• *API controller*: it receives API calls from the different components of the application and forwards them to the specific API of the layer. • *API request validator*: it checks the fields, verbs and paths of the API requests. • *D2D API*. API of the Device-to-Device interoperability layer of INTER-LAYER. • *N2N API*. API of the Network-to- Network interoperability layer of INTER-LAYER.

Table 14 Virtual Entity FC use cases and analysis

3.2.2.5 Platform Interoperability FG

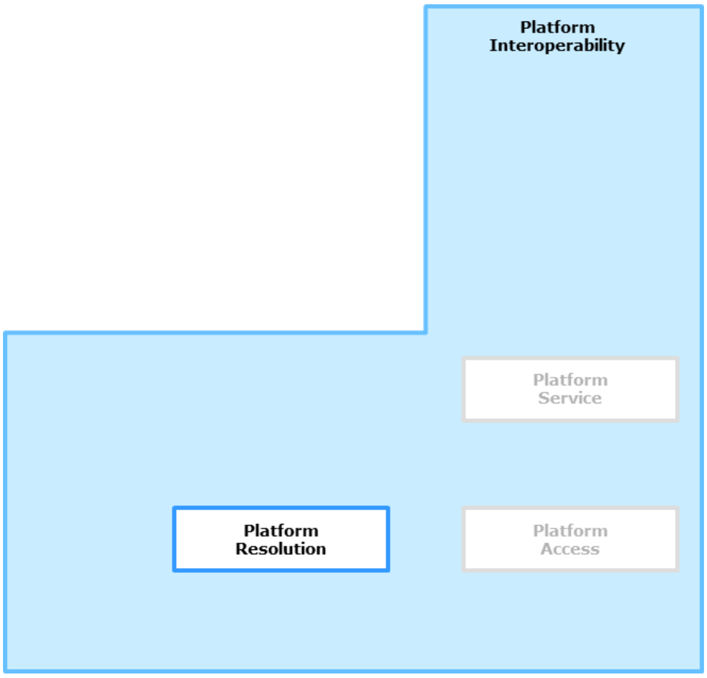


Figure 27: Functional Components of the Platform Interoperability FG instantiated for INTER-FW concrete architecture.

The list of instantiated FCs of the Platform Interoperability FG is the following:

Functional Component	Description
Platform Resolution	It is responsible for discovering and cataloguing the IoT Platforms that are available at a specific deployment of INTER-IoT as well as their devices, capabilities and IoT Platform Services, so that they can easily be found, when needed. This allows the remaining groups to not to know about the location of the platforms, or how the devices are connected to them. When any Functional Component needs to access a specific resource of an IoT Platform, it will use the Platform Resolution FC to get its identification, location and way of accessing.
List of supported Use Cases	2.2.3. Add a new platform 2.2.4. Update platform configuration 2.2.5. Remove a platform 2.2.6. Add or register a virtual instance of a gateway 2.2.7. Update configuration of a virtual instance of a gateway 2.2.8. Remove a virtual instance of a gateway
Architectural Components	• *API controller*: it receives API calls from the different components of the application and forwards them to the specific API of the layer. • *API request validator*: it checks the fields, verbs and paths of the API requests. • *D2D API*. API of the Device-to-Device interoperability layer of INTER-LAYER.

	• *N2N API*. API of the Network-to-Network interoperability layer of INTER-LAYER. • *M2M API*. API of the Middleware-to-Middleware interoperability layer of INTER-LAYER. • *AS2AS API*. API of the Application&Services-to-Application&Services interoperability layer of INTER-LAYER.

Table 15 Platform Resolution FC use cases and analysis

3.2.2.6 Service Interoperability FG

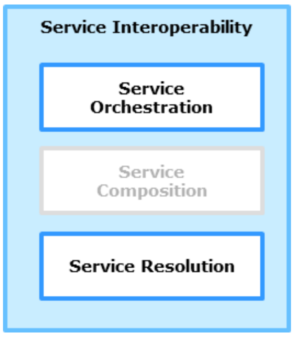


Figure 28: Functional Components of the Service Interoperability FG instantiated for INTER-FW concrete architecture.

The features of the Service Interoperability FG will be spread between the AS2AS of INTER-LAYER and the INTER-FW. The INTER-FW will provide access to the AS2AS API, but due to the nature of the AS2AS execution environment, with a specific GUI, the INTER-FW will be responsible for managing those execution environment controlling the user authorization and usage of resources quota.

The list of instantiated FCs of the Service Interoperability FG is the following:

Functional Component	Description
Service Resolution	The features of the Service Resolution will be spread between the AS2AS of INTER-LAYER
List of supported Use Cases	2.2.3. Add a new platform 2.2.4. Update platform configuration 2.2.5. Remove a platform 2.2.6. Add or register a virtual instance of a gateway 2.2.7. Update configuration of a virtual instance of a gateway 2.2.8. Remove a virtual instance of a gateway
Architectural Components	• *API controller*: it receives API calls from the different components of the application and forwards them to the specific API of the layer. • *API request validator*: it checks the fields, verbs and paths of the API requests. • *D2D API*. API of the Device-to-Device interoperability layer of INTER-LAYER. • *N2N API*. API of the Network-to-Network interoperability layer of INTER-LAYER.

	• *M2M API*. API of the Middleware-to-Middleware interoperability layer of INTER-LAYER. • *AS2AS API*. API of the Application&Services-to-Application&Services interoperability layer of INTER-LAYER. • *Docker Swarm controller*: Controller of clusters fo docker containers. • *Docker container controller*: Controller of the docker container used for supporting the software infrastructure for the execution environments of AS2AS.

Table 16 Service resolution FC use cases and analysis

3.2.2.7 Semantics FG

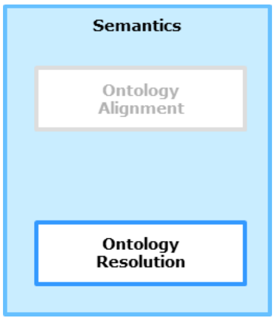


Figure 29: Functional Components of the Semantics FG instantiated for INTER-FW concrete architecture.

The list of instantiated FCs of the Semantics FG is the following:

Functional Component	Description
Ontology Resolution	The Ontology Resolution FC is responsible for managing the different ontologies used at the various IoT Platforms that are connected through INTER-IoT. These ontologies have a double approach: ● Syntactic knowledge: the syntax that the IoT Platforms uses for interchanging data ● Semantic knowledge: it is about being aware of the structure and meaning of the data, usually through JSON-LD in INTER-IoT.
List of supported Use Cases	2.2.14. List IPSM instances 2.2.15. Register IPSM instance 2.2.16. Unregister IPSM instance 2.2.17. List alignments in IPSM 2.2.18. List alignments in Semantic Repository 2.2.19. Add a new alignment to Semantic Repository 2.2.20. Add a new alignment to IPSM 2.2.21. Delete an alignment from IPSM 2.2.22. Delete an alignment from Semantic Repository 2.2.23. Delete an ontology from Semantic Repository 2.2.24. Get an alignment from IPSM 2.2.25. View an alignment from IPSM

	2.2.26. List active IPSM translation channels 2.2.27. Create a new translation channel in IPSM 2.2.28. Delete a translation channel in IPSM 2.2.29. List ontologies in Semantic Repository 2.2.30. View an ontology in Semantic Repository 2.2.31. View an alignment in Semantic Repository 2.2.32. View an alignment history in Semantic Repository 2.2.33. Get an ontology from Semantic Repository 2.2.34. Get an alignment from Semantic Repository 2.2.35. Add a new ontology to Semantic Repository 2.2.36. Add a new alignment to Semantic Repository
Architectural Components	• *API controller*: it receives API calls from the different components of the application and forwards them to the specific API of the layer. • *API request validator*: it checks the fields, verbs and paths of the API requests. • *IPSM API*. API of the Inter-Platform Semantic Mediator of INTER-LAYER.

Table 17 Ontology resolution FC use cases and analysis

3.2.3 Functional Components traceability

For ensuring the right engineering process of development, we have elaborated a traceability matrix of the different Functional Components against the Use Cases and the architectural components:

	Application FG		Management FG				Security FG				Device Access FG	Platform Interoperability FG	Service Interoperability FG	Semantics FG
	INTER-API	Web App	Configuration	Fault	Member	State	Authorisation	Authentication	Key Exchange & Management	Identity Management	Virtual Entity	Platform Resolution	Service Resolution	Ontology Resolution
2.2.1. Log In	✓	✓						✓	✓	✓				
2.2.2. API Key/Token generation	✓	✓						✓	✓	✓				
2.2.3. Add a new platform	✓	✓	✓		✓		✓				✓	✓	✓	
2.2.4. Update platform configuration	✓	✓	✓		✓		✓				✓	✓	✓	
2.2.5. Remove a platform	✓	✓	✓		✓		✓				✓	✓	✓	
2.2.6. Add or register a virtual instance of a gateway	✓	✓	✓		✓		✓				✓	✓	✓	
2.2.7. Update configuration of a virtual instance of a gateway	✓	✓	✓		✓		✓				✓	✓	✓	
2.2.8. Remove a virtual instance of a gateway	✓	✓	✓		✓		✓				✓	✓	✓	
2.2.9. Connect services with AS2AS tool		✓					✓							
2.2.10. Visualize INTER-IoT status		✓				✓	✓							
2.2.11. List permissions		✓					✓							
2.2.12. Manage platform permissions		✓					✓							
2.2.13. Revoke permissions		✓					✓							
2.2.14. List IPSM instances	✓	✓					✓							✓
2.2.15. Register IPSM instance	✓	✓	✓				✓							✓
2.2.16. Unregister IPSM instance	✓	✓	✓				✓							✓
2.2.17. List alignments in IPSM	✓	✓					✓							✓
2.2.18. List alignments in Semantic Repository	✓	✓					✓							✓
2.2.19. Add a new alignment to Semantic Repository	✓	✓	✓				✓							✓
2.2.20. Add a new alignment to IPSM	✓	✓	✓				✓							✓
2.2.21. Delete an alignment from IPSM	✓	✓	✓				✓							✓
2.2.22. Delete an alignment from Semantic Repository	✓	✓	✓				✓							✓
2.2.23. Delete an ontology from Semantic Repository	✓	✓	✓				✓							✓
2.2.24. Get an alignment from IPSM	✓	✓					✓							✓
2.2.25. View an alignment from IPSM	✓	✓					✓							✓
2.2.26. List active IPSM translation channels	✓	✓					✓							✓
2.2.27. Create a new translation channel in IPSM	✓	✓					✓							✓
2.2.28. Delete a translation channel in IPSM	✓	✓					✓							✓
2.2.29. List ontologies in Semantic Repository	✓	✓					✓							✓
2.2.30. View an ontology in Semantic Repository	✓	✓					✓							✓
2.2.31. View an alignment in Semantic Repository	✓	✓					✓							✓
2.2.32. View an alignment history in Semantic Repository	✓	✓					✓							✓
2.2.33. Get an ontology from Semantic Repository	✓	✓					✓							✓
2.2.34. Get an alignment from Semantic Repository	✓	✓					✓							✓
2.2.35. Add a new ontology to Semantic Repository	✓	✓					✓							✓
2.2.36. Add a new alignment to Semantic Repository	✓	✓					✓							✓
2.2.37. Add a new bridge							✓							
2.2.38. Update a bridge							✓							
2.2.39. Remove a bridge							✓							
2.2.40. Change user data	✓	✓	✓		✓		✓							
2.2.41. Set up QoS requirements		✓					✓							
2.2.42. Visualize SDN topology changes		✓												

Table 18 Traceability matrix of Functional Components of INTER-FW vs. use cases.

	Application FG		Management FG				Security FG				Device Access FG	Platform Interoperability FG	Service Interoperability FG	Semantics FG
Architectural components	INTER-API	Web App	Configuration	Fault	Member	State	Authorisation	Authentication	Key Exchange & Management	Identity Management	Virtual Entity	Platform Resolution	Service Resolution	Ontology Resolution
API controller	✓										✓	✓	✓	✓
API request validator	✓										✓	✓	✓	✓
API manager					✓	✓			✓					
Inter-fw frontend		✓												
Inter-fw controller		✓												
User registry		✓	✓											
Nodes registry		✓	✓		✓	✓								
Node configuration validator		✓	✓											
Inter-fw log handler				✓										
Auth proxy							✓	✓						
Auth server							✓	✓						
Node credentials manager							✓	✓		✓				
Inter-fw user manager								✓	✓	✓				
D2D API											✓	✓	✓	
N2N API											✓	✓	✓	
MW2MW API												✓	✓	
AS2AS API												✓	✓	
IPSM API												✓	✓	✓
Docker Swarm controller													✓	
Docker container controller													✓	

Table 19 Traceability matrix of Functional Components of INTER-FW vs. INTER-FW architectural components

3.3 INTER-FW Components

The following diagram depicts the components taking part in the INTER-FW administration and configuration framework. For simplicity, some components that share the same interactions have been represented in one single box. Colours have been applied to identify the main functional groups.

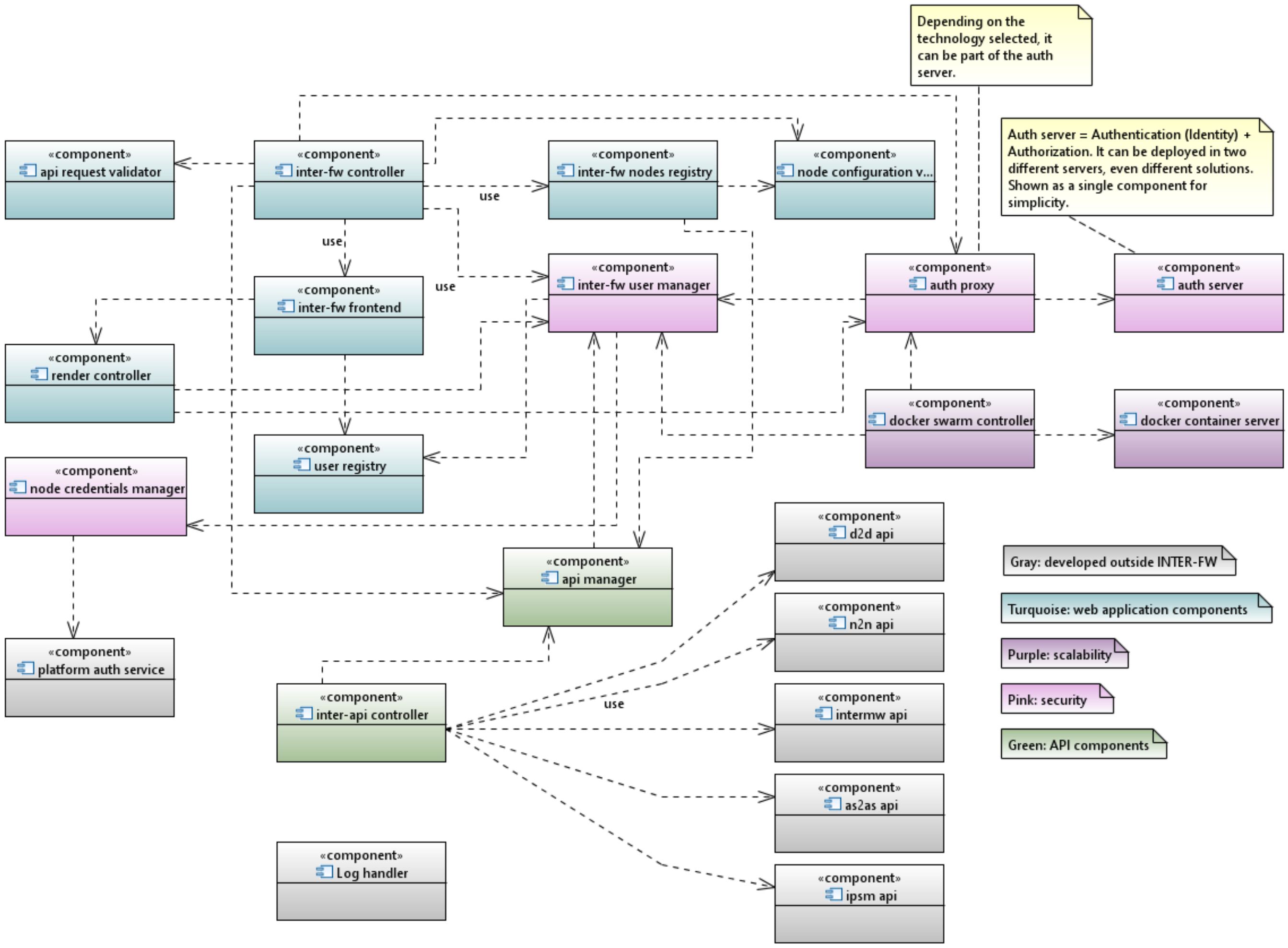

Depending on the technology selected, it can be part of the auth server.
Auth server = Authentication (Identity) + Authorization. It can be deployed in two different servers, even different solutions. Shown as a single component for simplicity.
«component» api request validator
«component» inter-fw controller
«component» inter-fw nodes registry
«component» node configuration v...
use
use
«component» inter-fw frontend
use
«component» inter-fw user manager
«component» auth proxy
«component» auth server
«component» render controller
«component» user registry
«component» docker swarm controller
«component» docker container server
«component» node credentials manager
«component» d2d api
Gray: developed outside INTER-FW
«component» api manager
«component» n2n api
Turquoise: web application components
Purple: scalability
«component» platform auth service
«component» inter-api controller
use
«component» intermw api
Pink: security
Green: API components
«component» as2as api
«component» Log handler
«component» ipsm api

Figure 30 UML Component diagram of INTER-FW administration and configuration framework

The functional groups identified are:

- **Web application components**: it follows the classical design pattern model-view-controller. Includes all components related to the data storage, organization and visualization for the end-user. This group contains:
  - **Inter-fw frontend**: The collection of views that will be rendered to the user (main view, platforms, gateways, network, services, semantics, configuration and users).
  - **Inter-fw controller**: This component performs the classical operations of a controller: process and forward data requests to the model, ask for fields, credentials and permission validation,etc.
  - **Api request validator**: it checks the fields, verbs and paths of the api requests before sending them.
  - **User registry**: a store of the INTER-IoT users.

  - **Nodes registry**: a storage point for the registered nodes: platforms, gateways, services, etc. Each entry contains the need information to univocally identify the node (see 2.3.2)
  - **Node configuration validator**: it checks that the configuration submitted for each node is correct, before calling the associated API.
- **Security**: components intended to avoid the access to anonymous or unregistered users. It also prevents the unauthorised access to data.
  - **Inter-fw user manager**: it processes and forwards user related data.
  - **Auth proxy**: it receives authentication and authorization requests and replies if the action requested is allowed for a given user. The use of a proxy is appropriate to decouple the auth servers from the web application environment, enabling technology independence and better scalability.
  - **Auth server**: it grants access to users to the web environment, as well as determines whether the actions requested are permitted for a given user or not.
  - **Node credentials manager**: stores the credentials for the nodes in their appropriate format. It also process authentication and authorization requests to the nodes, using the stored credentials.
- **Scalability**: it contributes to enable the growing of connected devices and users to the INTER-FW. Based in containerization technologies, it is composed by two components:
  - **Docker swarm controller**: it receives requests of new containers, build the calls based on the docker-swarm api and forwards the to the container repository. It also associates the containers with the user that requested them.
  - **Docker container repository**: it contains a set of containers providing INTER-IoT features.
- **API**: these components perform operations related to the global API, INTER-API.
  - **API Manager**: it features a wide set of operations (see INTER-API, Section 4), including API requests security, accounting, metrics, versioning, etc.
  - **API controller**: it receives API calls from the different components of the application and forwards them to the specific API of the layer, through the API Manager.

3.3.1 Communication diagrams

The following diagrams represent the main interactions and the message flow between objects in an OO application and also imply the basic associations (relationships) between classes. The rectangles represent the various objects involved that make up the application. The components identified in the previous section, in this case are represented as lifelines. In these communication diagrams, for the sake of simplicity, only most important interactions are represented, also, response messages are avoided. The idea of these diagrams in to depict in one single figure the multiple interactions that can occur in the main operations of the INTER-FW management application, so that the relation and the chain of responsibilities of the components can be determined in a set of features. Alternatively, these diagrams can be represented into sequence diagrams. The following format, however, shows equivalent information while keeping a more compact style. The numbering of the messages shows three figures. This differs slightly from the UML standard.

```
Id: actionId.sequenceInAction
```

The first Id is a unique identifier for the message. After the colon, there is an identifier for the action (making possible to represent different user operations in the same diagram) while the third number represent the sequence in the given action.

3.3.1.1 Login

The following diagram depicts the main communications between components in the Login operation.

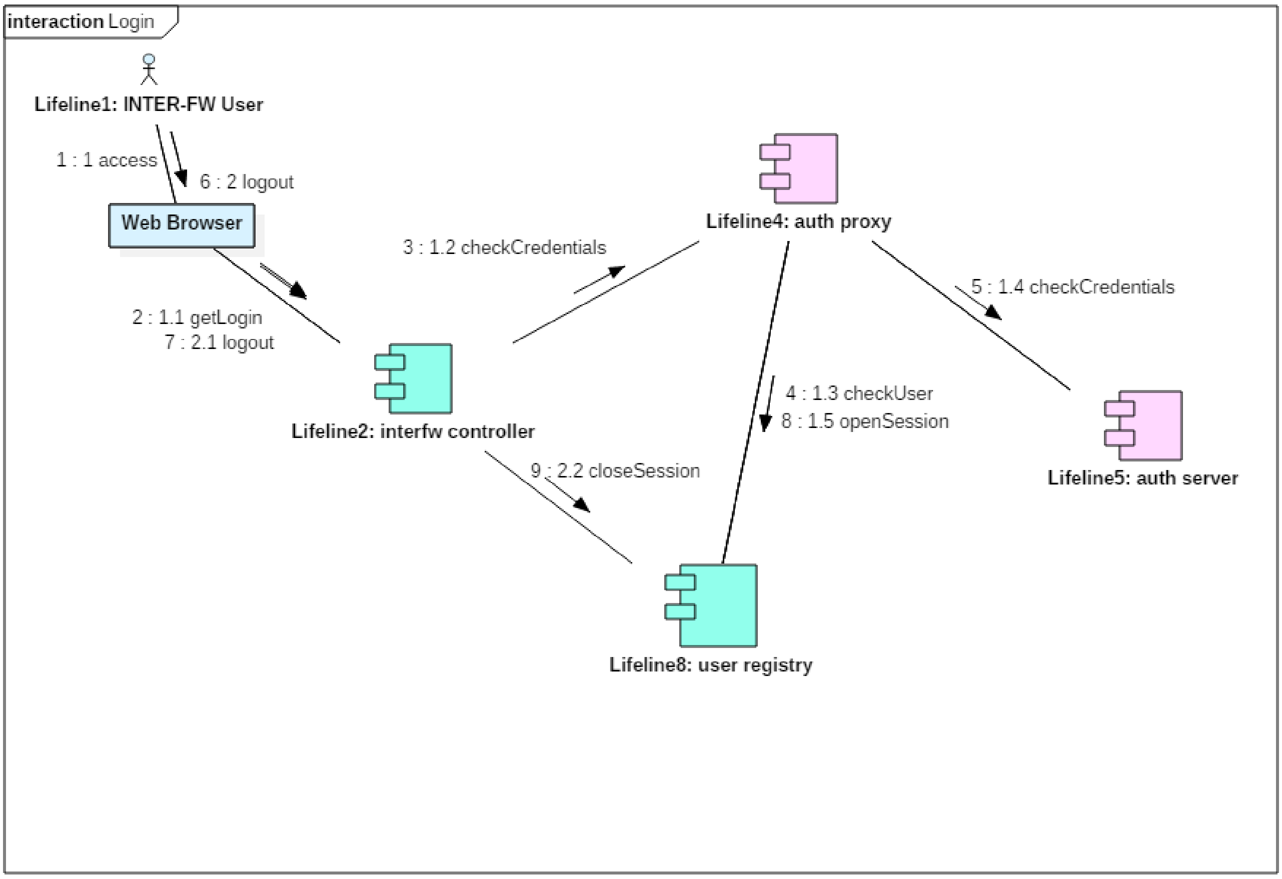


Figure 31 Login use cases family: communication diagram

3.3.1.2 Manage platforms

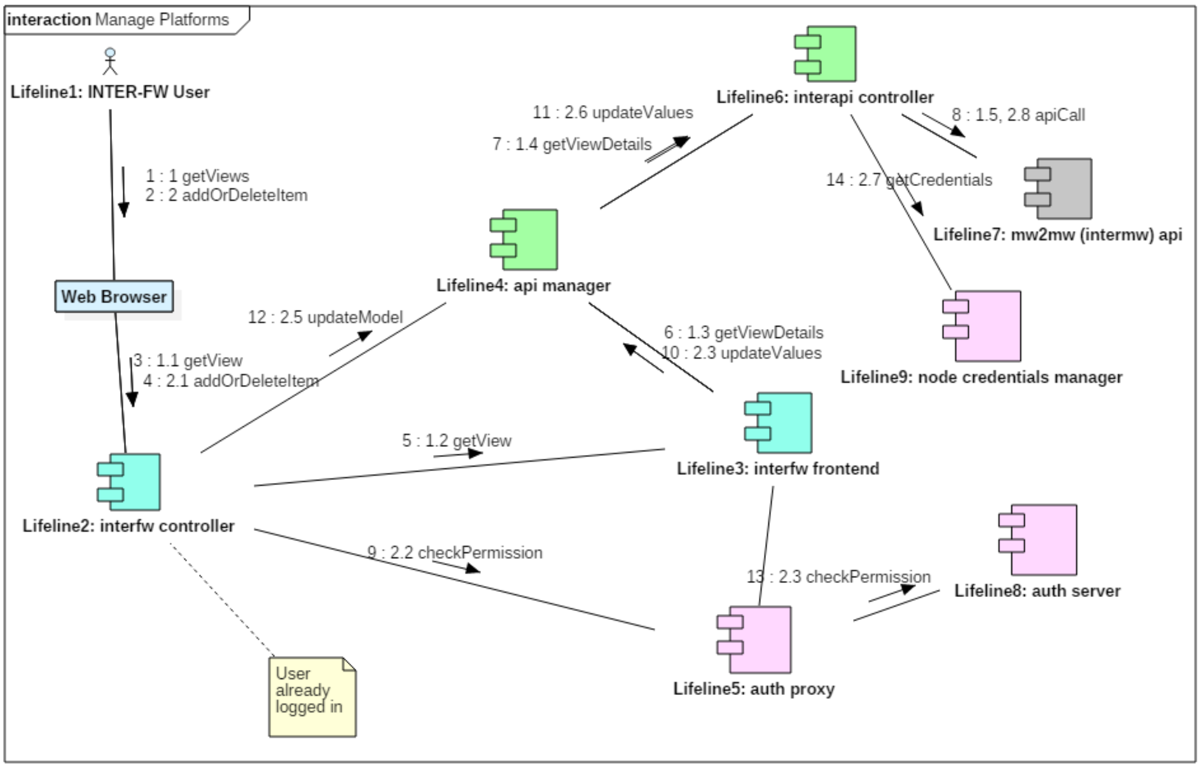


Figure 32 Manage platforms use cases family: communication diagram

3.3.1.3 Manage gateways

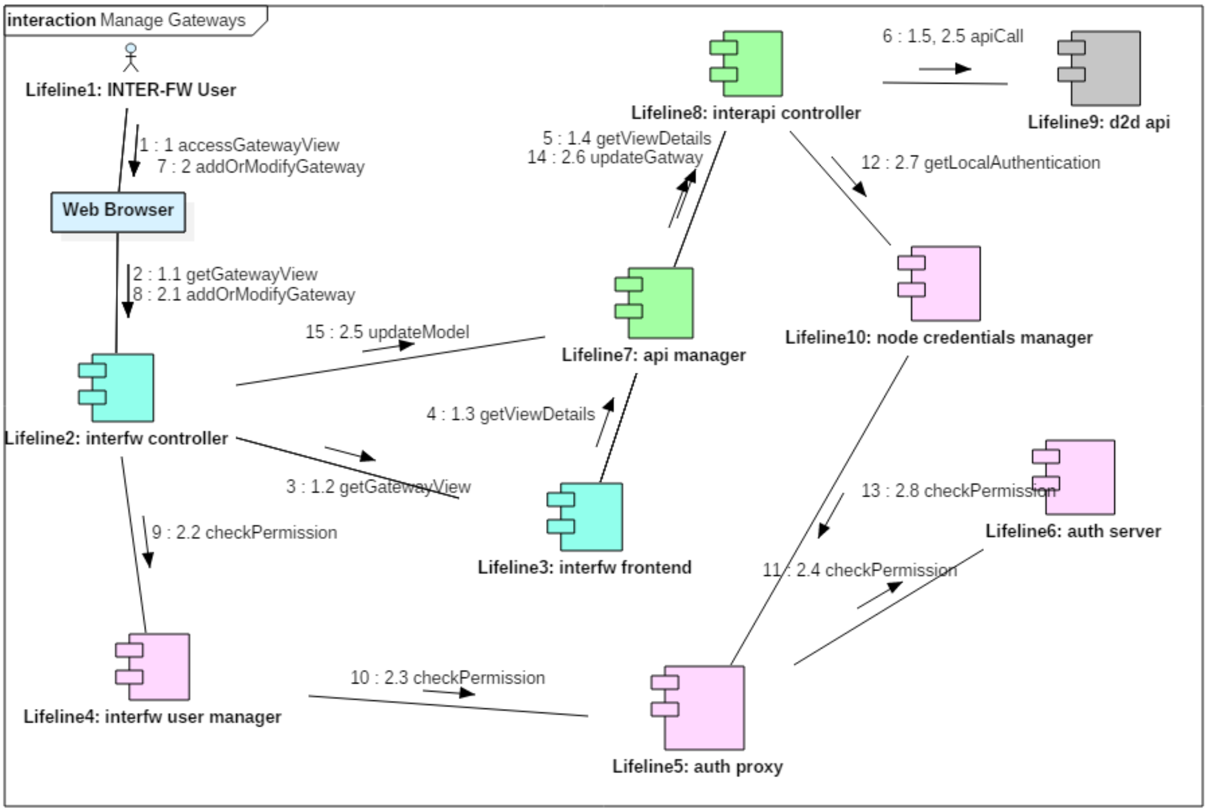


Figure 33 Manage gateways use cases family: communication diagram

3.3.1.4 Manage permissions

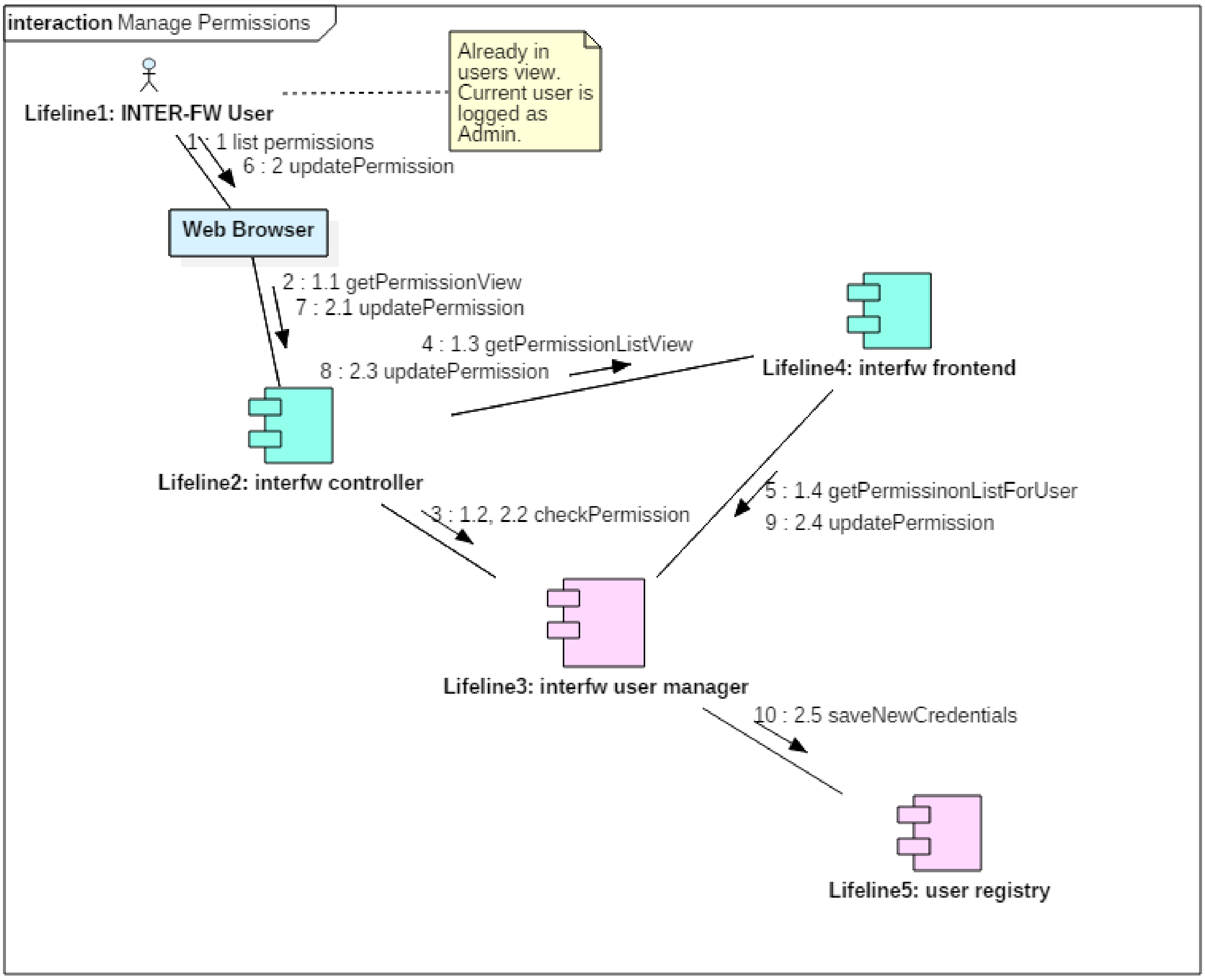

interaction Manage Permissions
Lifeline1: INTER-FW User
Already in users view. Current user is logged as Admin.
1 : 1 list permissions
6 : 2 updatePermission
Web Browser
2 : 1.1 getPermissionView
7 : 2.1 updatePermission
4 : 1.3 getPermissionListView
8 : 2.3 updatePermission
Lifeline4: interfw frontend
Lifeline2: interfw controller
5 : 1.4 getPermissinonListForUser
9 : 2.4 updatePermission
3 : 1.2, 2.2 checkPermission
Lifeline3: interfw user manager
10 : 2.5 saveNewCredentials
Lifeline5: user registry

Figure 34 Manage permissions use cases family: communication diagram

3.3.1.5 Manage IPSM

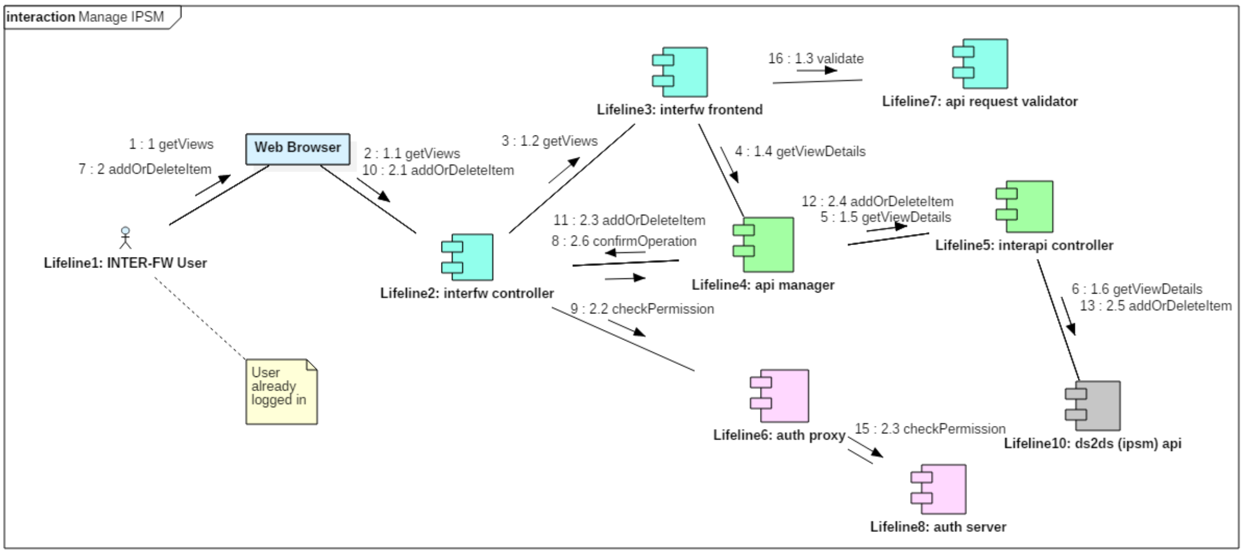


Figure 35 Manage IPSM use cases family: communication diagram

3.3.1.6 Manage networks

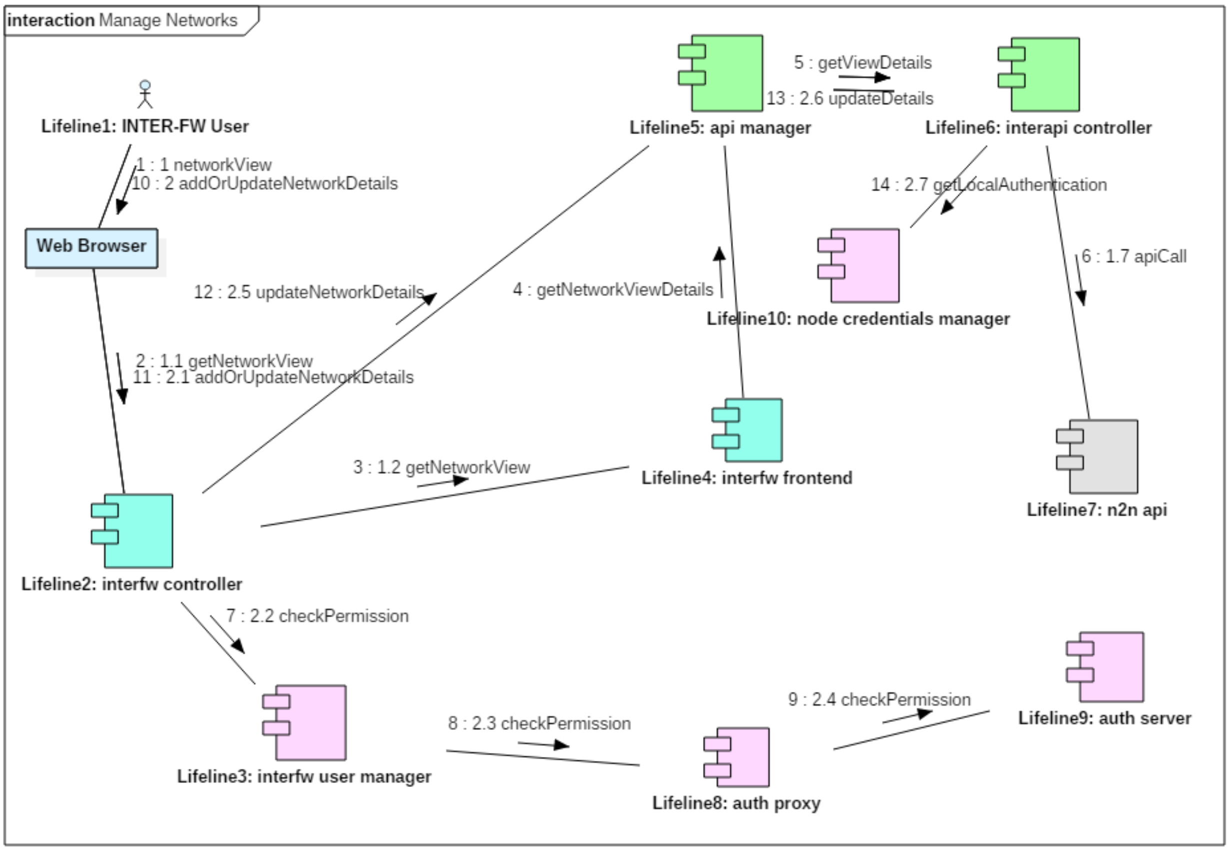

interaction Manage Networks
Lifeline1: INTER-FW User
1 : 1 networkView
10 : 2 addOrUpdateNetworkDetails
Web Browser
2 : 1.1 getNetworkView
11 : 2.1 addOrUpdateNetworkDetails
Lifeline2: interfw controller
12 : 2.5 updateNetworkDetails
3 : 1.2 getNetworkView
Lifeline4: interfw frontend
4 : getNetworkViewDetails
Lifeline5: api manager
5 : getViewDetails
13 : 2.6 updateDetails
Lifeline6: interapi controller
14 : 2.7 getLocalAuthentication
Lifeline10: node credentials manager
6 : 1.7 apiCall
Lifeline7: n2n api
7 : 2.2 checkPermission
Lifeline3: interfw user manager
8 : 2.3 checkPermission
Lifeline8: auth proxy
9 : 2.4 checkPermission
Lifeline9: auth server

Table 20 Manage networks use cases family: communication diagram

3.3.1 Entity relationship diagram

According the entities analysed in section 2.3.2, the following diagram represents the entities relationship. This is a representation of how the different entities are organised in the database and how they can be accessed.

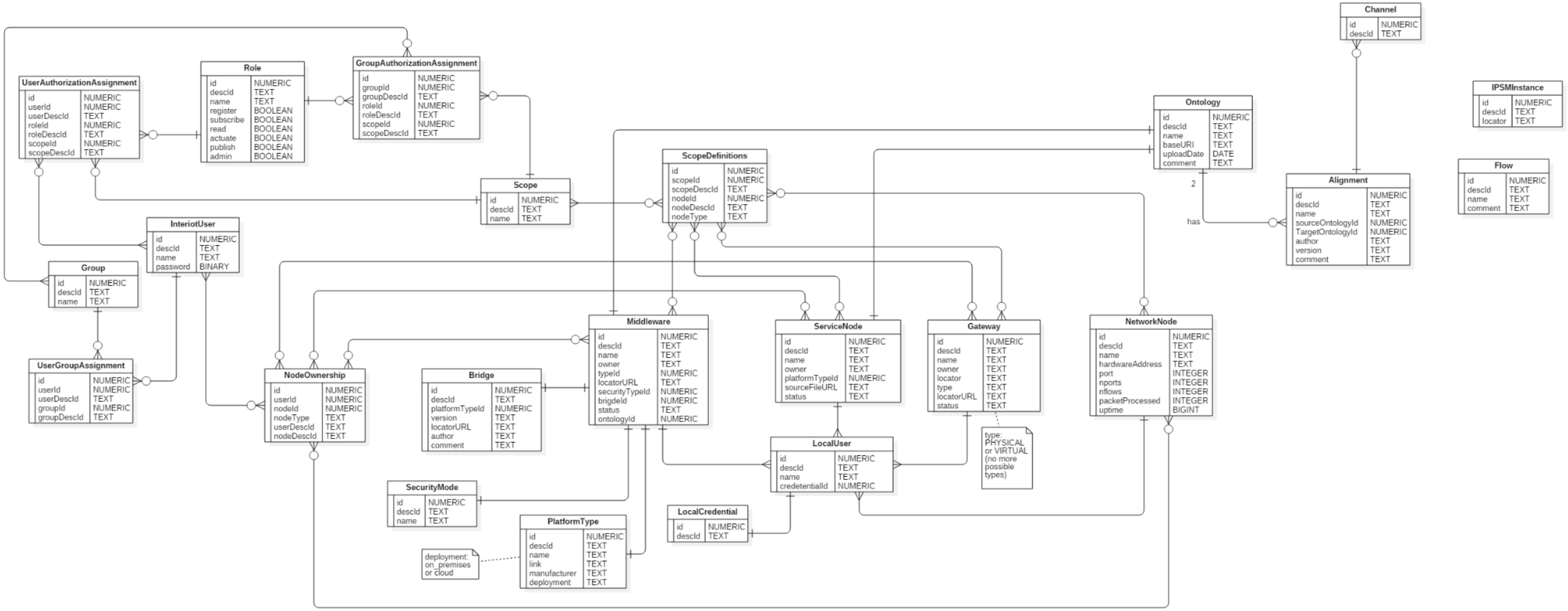


Figure 36 ERD of INTER-FW (SQL style)

3.4 INTER-FW security management

Security is a key aspect to take in account when more than one solution and more than one client access at the same system. One of the main problems with security is that there are vast amount of point gathering data (devices, sensor, tags, etc) and of course not all of them has the enough capacity or processing power to implement trustable security mechanisms to ensure the privacy and authenticity of the given data. For that reason, the upper layers have to make up for this flaws and take the leading role when we talk about implementing security.
But when we talk about this security we can define two key points; trust and privacy.

As we can see in [3] trust englobes several parts of a system:

- Device Trust: Need to interact with reliable devices.
- Processing Trust: Need to interact with correct and meaningful data.
- Connection Trust: Requirement to exchange the right data with the right service providers and nobody else
- System Trust: Desire to leverage a dependable overall system. This can be achieved by providing as much transparency of the system as possible.

And for other side the privacy encompasses the sensible information managed by an IoT system that can put in harm a person, company or the system itself. Hence, the security in IoT cannot be treated as in the typical deployment and it has to be focus in preserve the privacy and trust of the elements included in the schema.

Since INTER-IoT is composed of several layers and modules, the security implementation cannot be monolithic. INTER-FW provides a portal to access several layers of the interoperability architecture, and each one of them has to be isolated and just communicate through the official mechanisms and channels.

INTER-FW aims at providing global and open platform-level interoperability among heterogeneous IoT platforms coupled through specifically developed ILIs. In this sense, the security-awareness and implementation in this component has to take into account those particular aspects and depict an architecture that encloses them and creates a reliable security framework across the different layers of the global INTER-IoT solution.

Each layer will be responsible of ensuring the integrity and encryption (if necessary) of the data it stores and the data exchanged with other components and INTER-FW. Nevertheless, the authentication mechanism will be centralised in order to have a coherent access over all components of an INTER-IoT deployment.

An INTER-IoT deployment will interact with one or more IoT Platforms and those IoT Platforms could implement their own security mechanisms. For that reason, when a platform is registered in INTER-FW it will also store (encrypted) every authentication detail needed to interact with that IoT Platform.

In the following figure, we can see a scheme of how this security architecture is implemented:

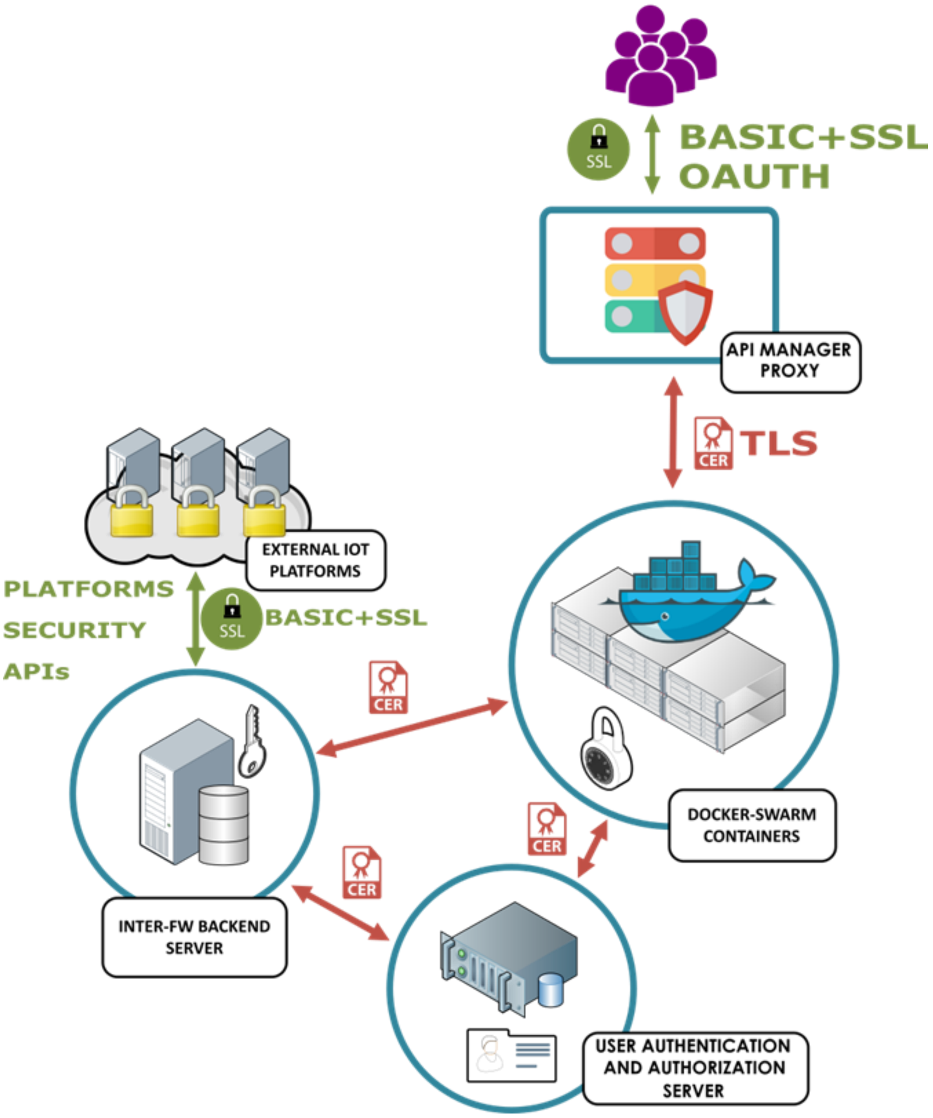


Figure 37 Security agents in INTER-IoT

As it can be observed on the figure above, there are security-specific components that, brought together with previously mentioned agents, comprise the whole design of the privacy and security model:

- **External IoT Platforms:** These are the external IoT platforms that will be configured and available to the rest of the layers

- **INTER-FW Backend Server:** Here is where INTER-FW will be installed and deployed; it is also the frontend for the management and configuration of the INTER-IoT deployment. Regarding security aspects, all the authentication credentials and security specific data to interact with the external IoT Platforms will be stored under the **Cryptex**[36] library (more detailed information available in D3.2).

- **User Authentication Server:** This server will be responsible to handle all authentication related mechanisms. It will interact with the INTER-FW backend server to provide the configuration and management points of the authentication integrated with the INTER-FW

[36] https://github.com/scrogson/cryptex

frontend, but its main purpose is to act as a centralised point of authentication for all the other layers deployed in the docker-swarm containers.

For this purpose, the **WSO2 Identity Server** has been chosen as the centralised authentication server. It provides single sign-on and identity federation capabilities, multi-factor authentication, management of users, groups and roles, monitoring and auditing and multiple connectors and libraries for easy integration. More detailed information is available in D3.2.

- **Docker-Swarm Container:** Here is where different layers components will be deployed and managed, as described in the point INTER-FW relation with INTER-Layer . Regarding security and authentication, each layer will be responsible to implement their security mechanism in the framework and language chosen for each component. In order to have a coherent and centralised authentication mechanism, each layer will have their connector to the **User Authentication Server** to take advantage of all the authentication capabilities that provides.

3.5 INTER-FW GUI design

From the analysis performed in Chapter 2, the necessary INTER-FW functionalities have been extracted and a mock-up was designed. With this design, which also allows you to navigate among the different menus, we can start to develop the webapp more efficiently.

The following images show the appearance that the different INTER-FW webapp menus should have.

Login

In the below figure there is the access page to the webapp with the login and the password.

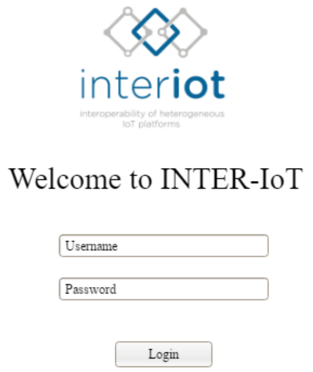
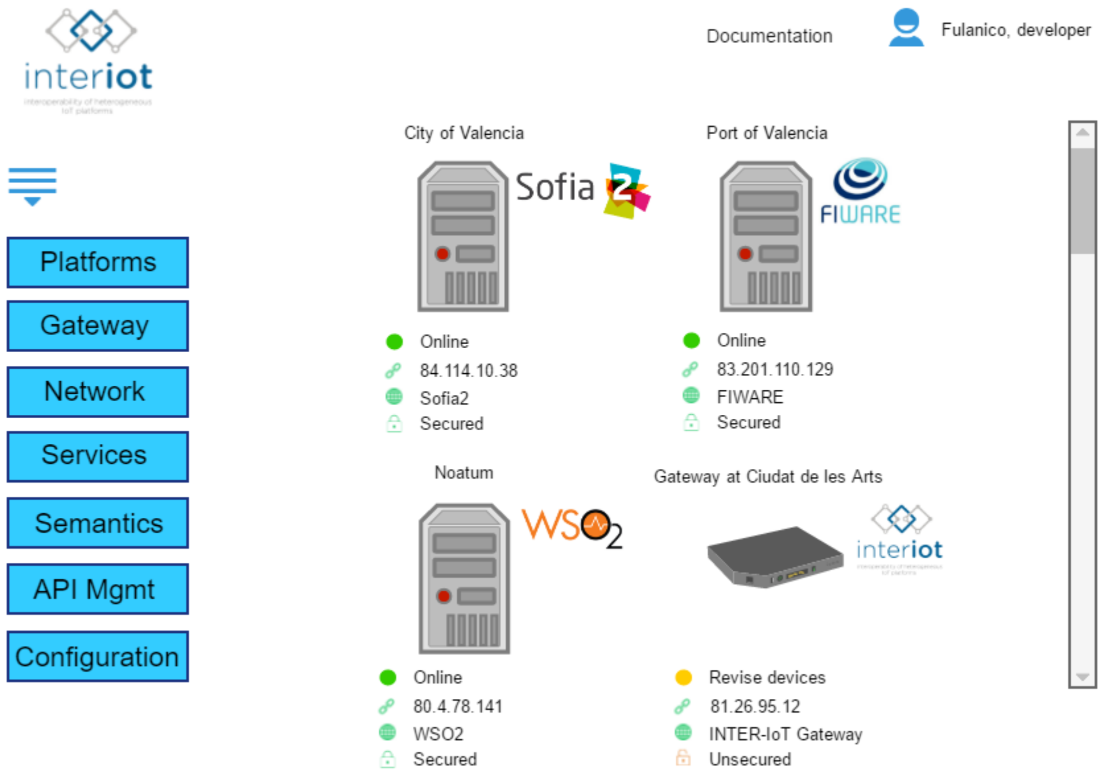


Figure 38. INTER-FW web application login page

3.5.1 Main screen (landing page)

In the Figure 39, you can find the main page of the web application. There is information of the registered platforms and gateway. On the right is the menu to navigate among the different pages.Figure 23, you can find the main page of the web application. There is information of the registered platforms and gateway. On the right is the menu to navigate among the different pages.



City of Valencia
Online
84.114.10.38
Sofia2
Secured

Port of Valencia
Online
83.201.110.129
FIWARE
Secured

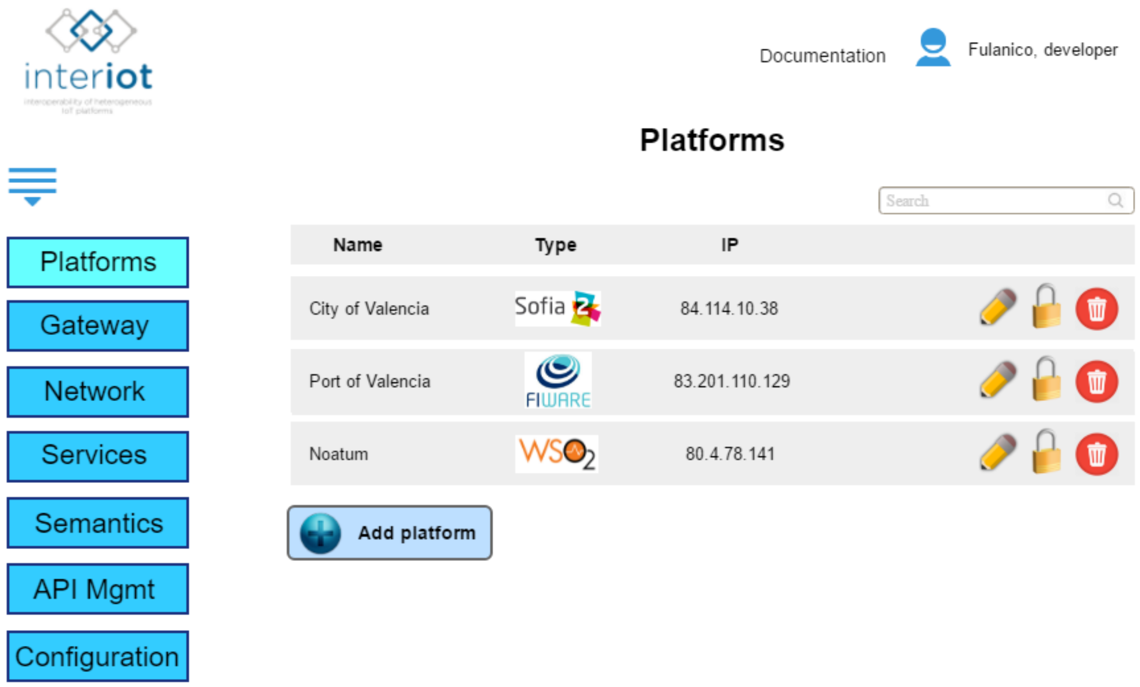
Noatum
Online
80.4.78.141
WSO2
Secured

Gateway at Ciudat de les Arts
Revise devices
81.26.95.12
INTER-IoT Gateway
Unsecured

Figure 39. INTER-FW web application main view

3.5.2 Platforms view

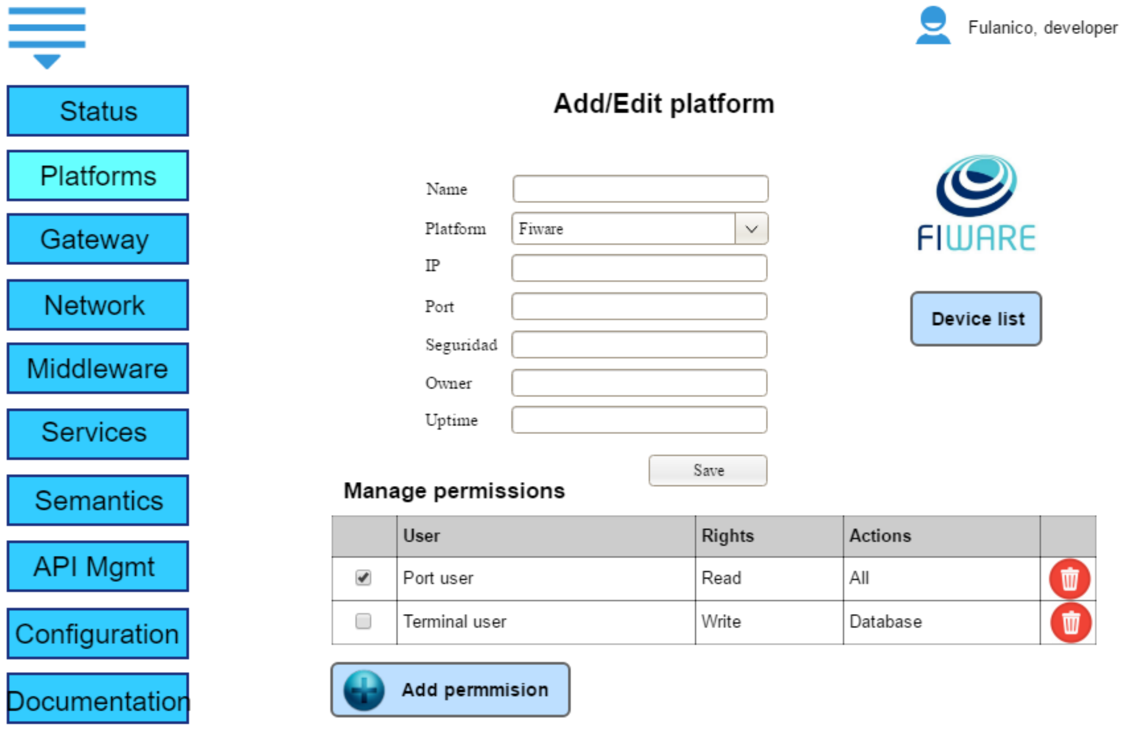
In the platforms view main page (Figure 40) there is a list of registered platforms with their main properties. There is a button to add more platforms (Figure 41). Each platform has three buttons to edit the platform (Figure 41), show device list (Figure 42), and delete the platform.Figure 24) there is a list of registered platforms with their main properties. There is a button to add more platforms (). Each platform has three buttons to edit the platform (), show device list (), and delete the platform.



Platforms

Name	Type	IP
City of Valencia	Sofia2	84.114.10.38
Port of Valencia	FIWARE	83.201.110.129
Noatum	WSO2	80.4.78.141

Figure 40. INTER-FW web application platforms view



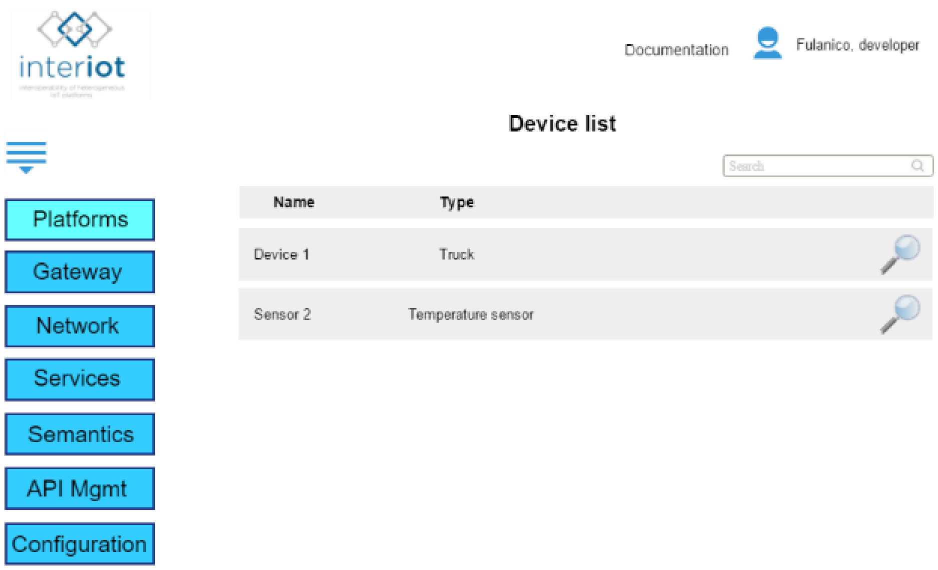
Add/Edit platform

Name
Platform Fiware
IP
Port
Seguridad
Owner
Uptime

Manage permissions

	User	Rights	Actions
☑	Port user	Read	All
☐	Terminal user	Write	Database

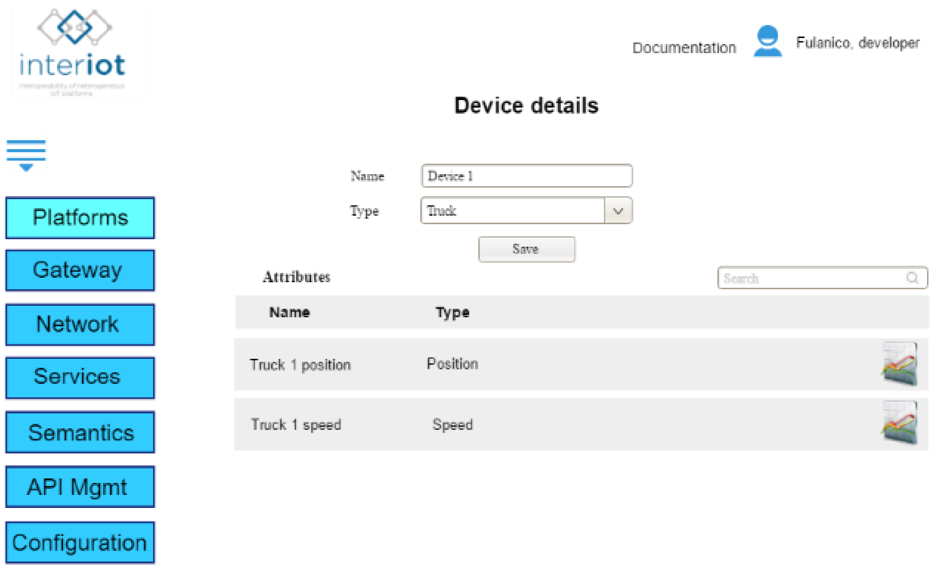
Figure 41. INTER-FW web application add platform view



Device list

Name	Type
Device 1	Truck
Sensor 2	Temperature sensor

Figure 42. INTER-FW web application platform's device list view



Device details

Name: Device 1
Type: Truck

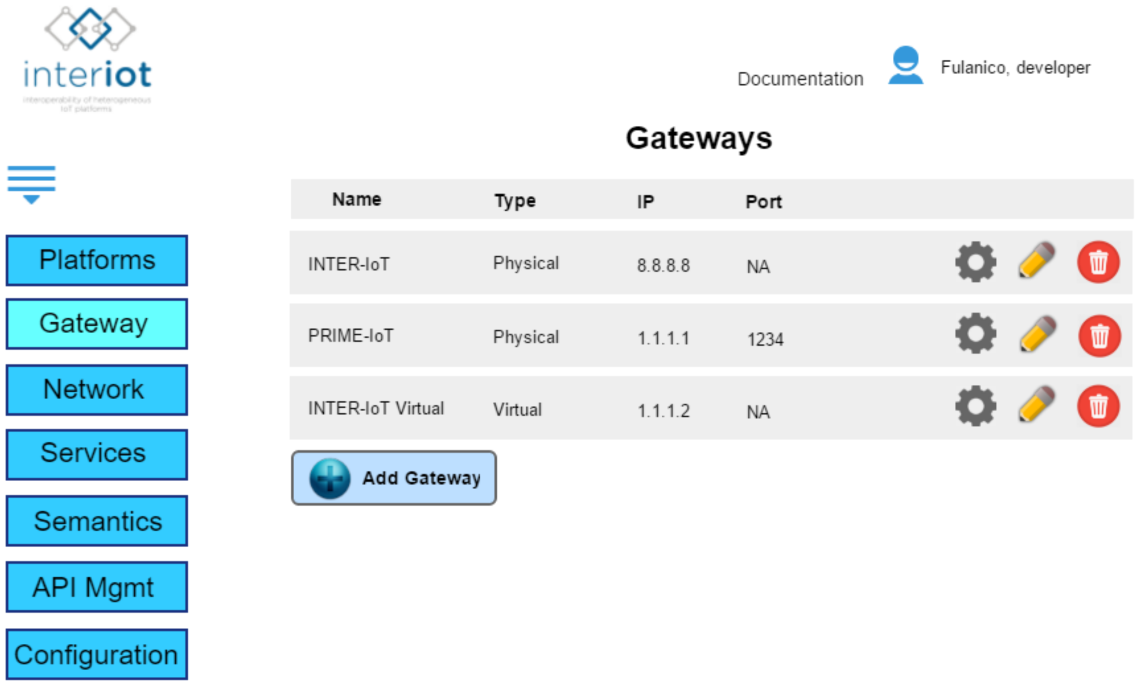
Attributes

Name	Type
Truck 1 position	Position
Truck 1 speed	Speed

Figure 43. INTER-FW web application platform's device details view

3.5.3 Gateway view

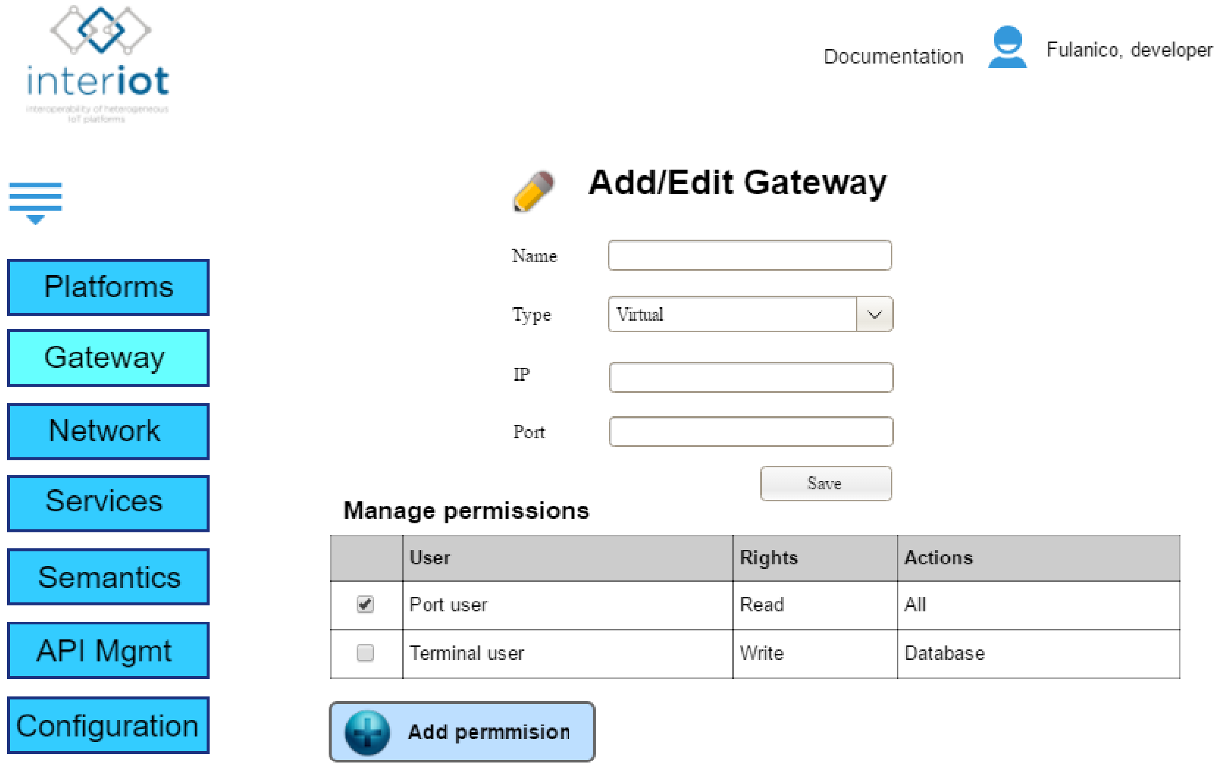
In the gateway main view main page (Figure 44) there is a list of registered gateways with their main properties. There is a button to add more gateways (Figure 45). Each gateway has three buttons to edit the gateway (Figure 45, Figure 41), manage devices (Figure 46), and delete the gateway. Figure 28) there is a list of registered gateways with their main properties. There is a button to add more gateways (Figure 44). Each gateway has three buttons to edit the gateway , manage devices , and delete the gateway.



Gateways

Name	Type	IP	Port
INTER-IoT	Physical	8.8.8.8	NA
PRIME-IoT	Physical	1.1.1.1	1234
INTER-IoT Virtual	Virtual	1.1.1.2	NA

Figure 44. INTER-FW web application gateway view



Add/Edit Gateway

Name

Type Virtual

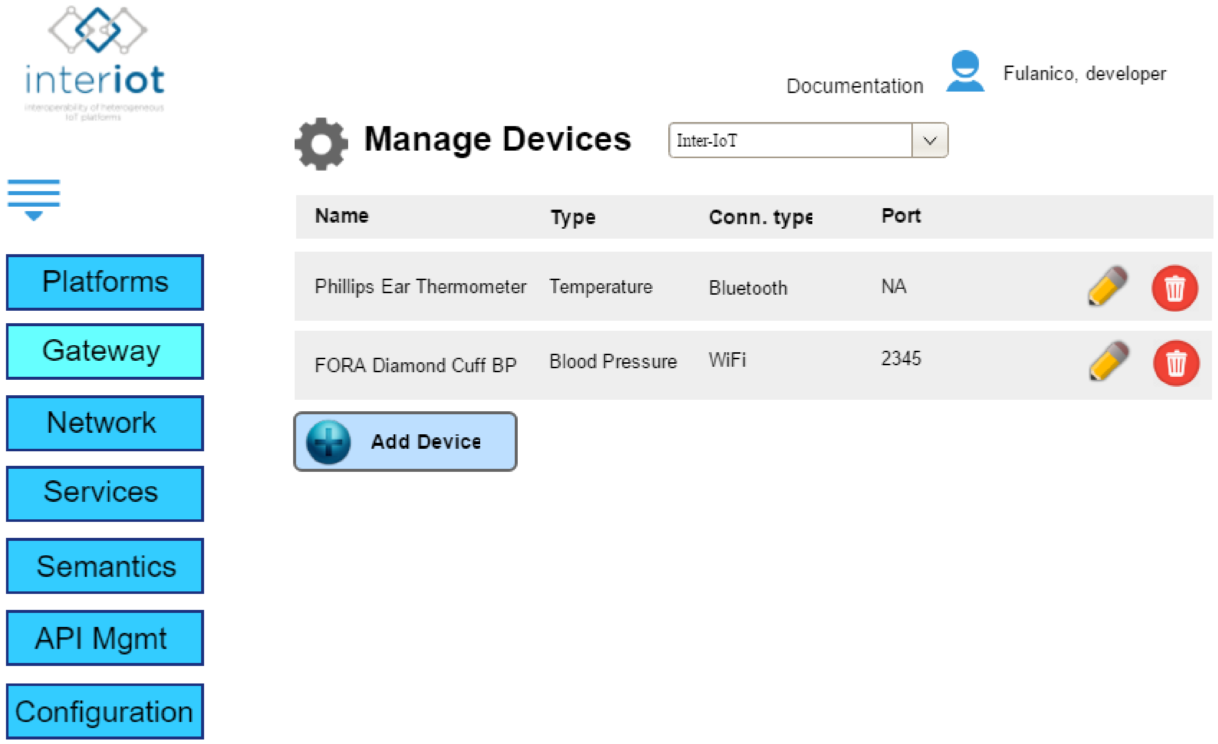
IP

Port

Manage permissions

	User	Rights	Actions
☑	Port user	Read	All
□	Terminal user	Write	Database

Figure 45. INTER-FW web application add gateway view

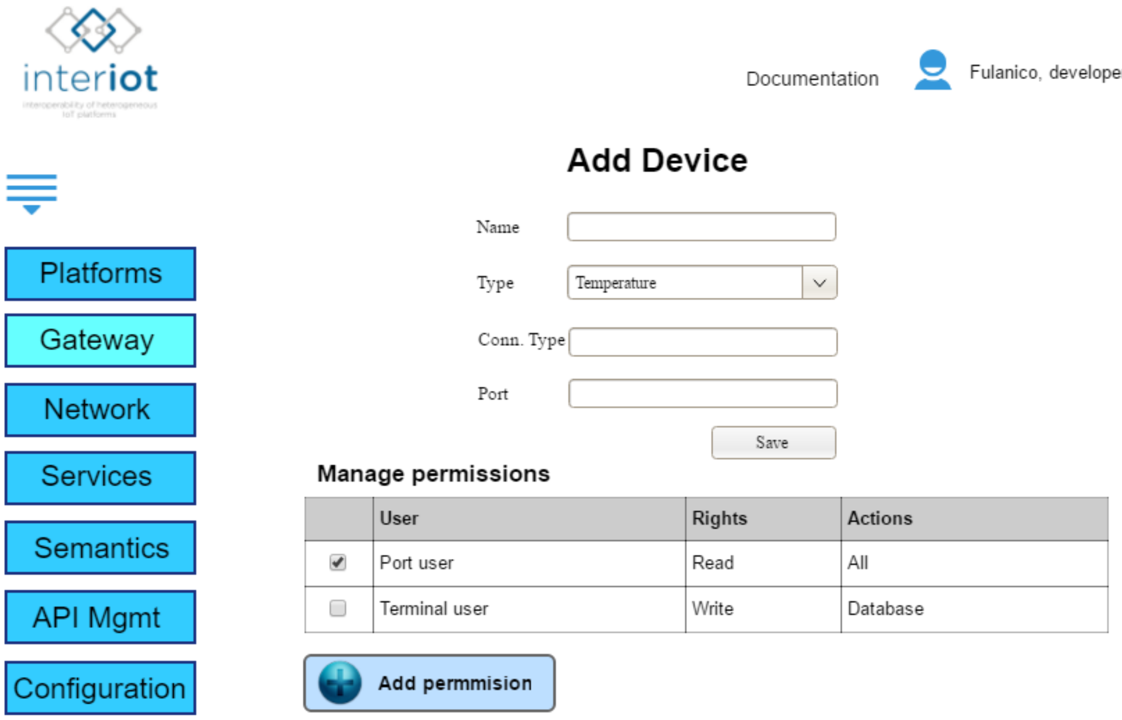


Manage Devices

Inter-IoT

Name	Type	Conn. type	Port
Phillips Ear Thermometer	Temperature	Bluetooth	NA
FORA Diamond Cuff BP	Blood Pressure	WiFi	2345

Figure 46. INTER-FW web application manage gateway devices view



Add Device

Name

Type: Temperature

Conn. Type

Port

Manage permissions

	User	Rights	Actions
☑	Port user	Read	All
□	Terminal user	Write	Database

Figure 47. INTER-FW web application show gateway device details view

3.5.4 Network view

In the network main view main page (Figure 48) there is a topology of the network with all the elements. There are three buttons to manage the elements (Figure 49), add new elements (Figure 50) and Quality of service. Figure 41 there is a topology of the network with all the elements. There are three buttons to: manage the elements, add new elements and Quality of service.

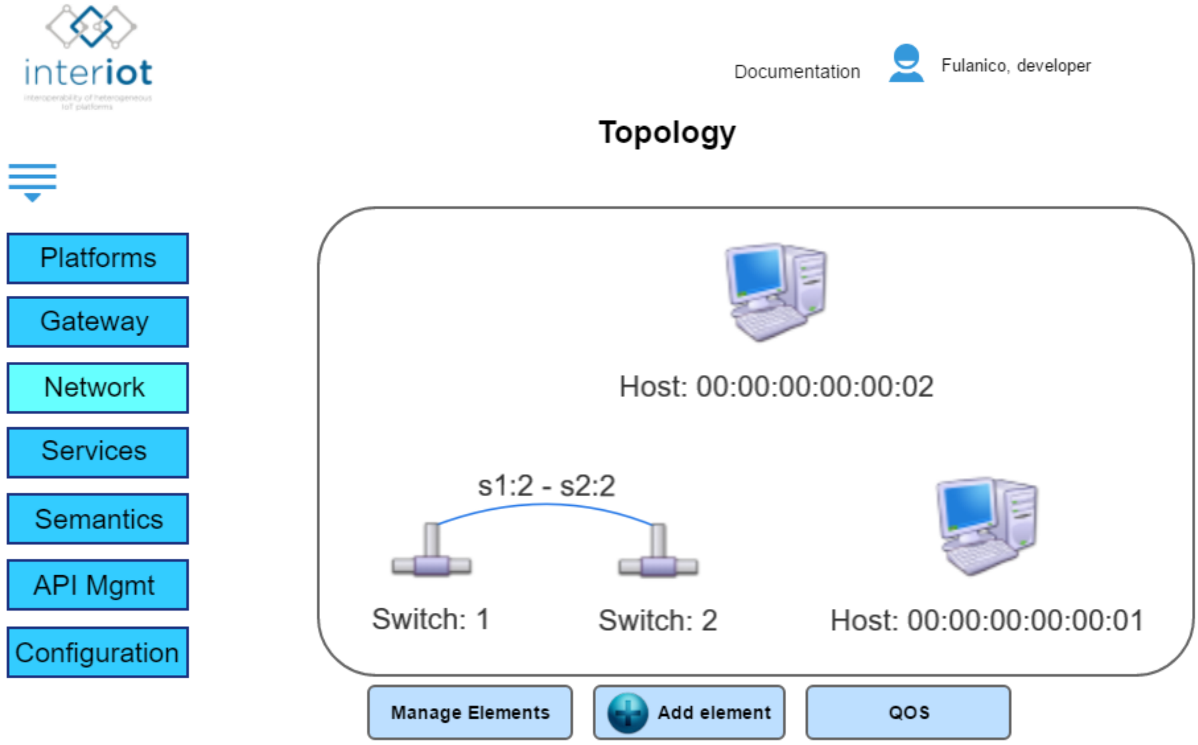
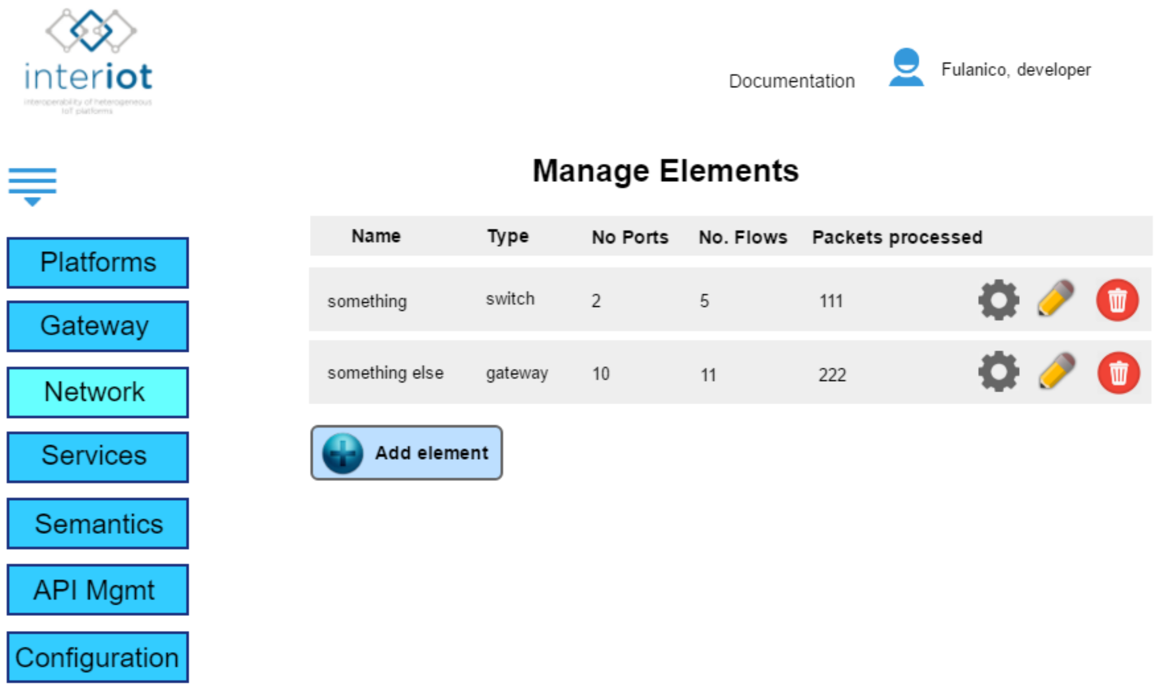


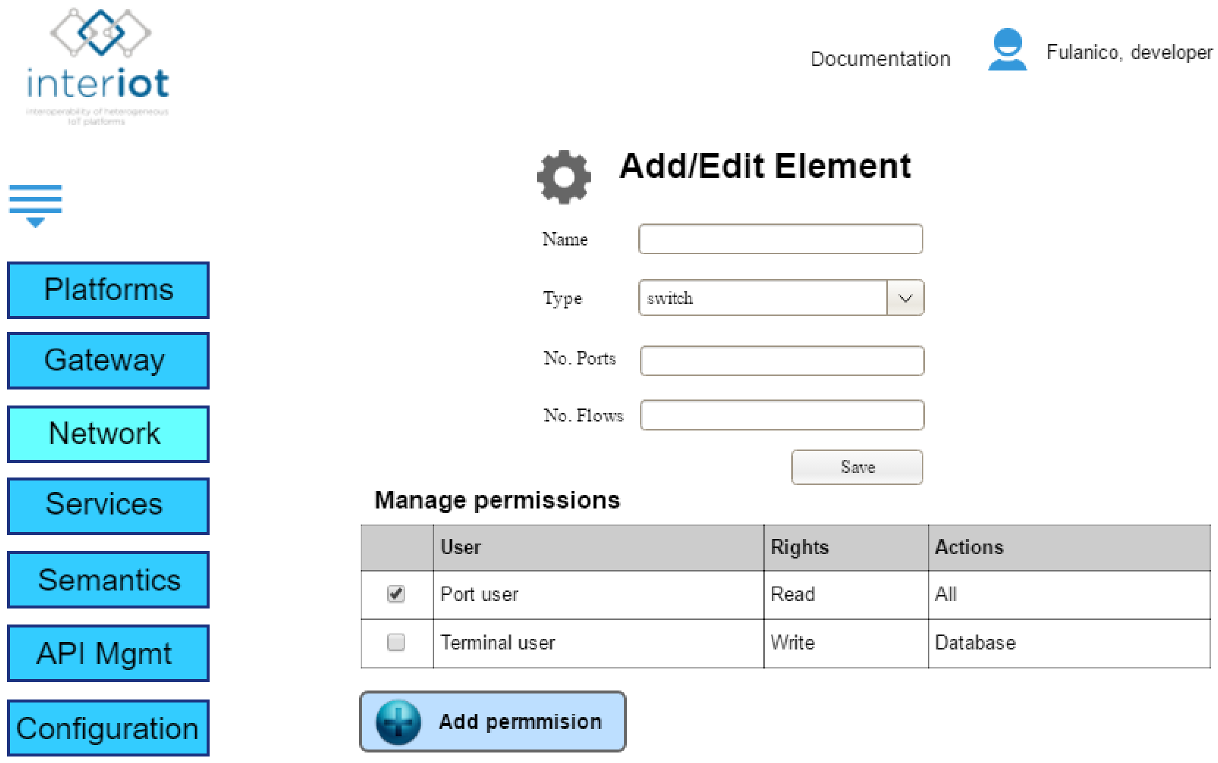
Figure 48. INTER-FW web application network view



Manage Elements

Name	Type	No Ports	No. Flows	Packets processed
something	switch	2	5	111
something else	gateway	10	11	222

Figure 49. INTER-FW web application manage network elements view



Add/Edit Element

Name

Type switch

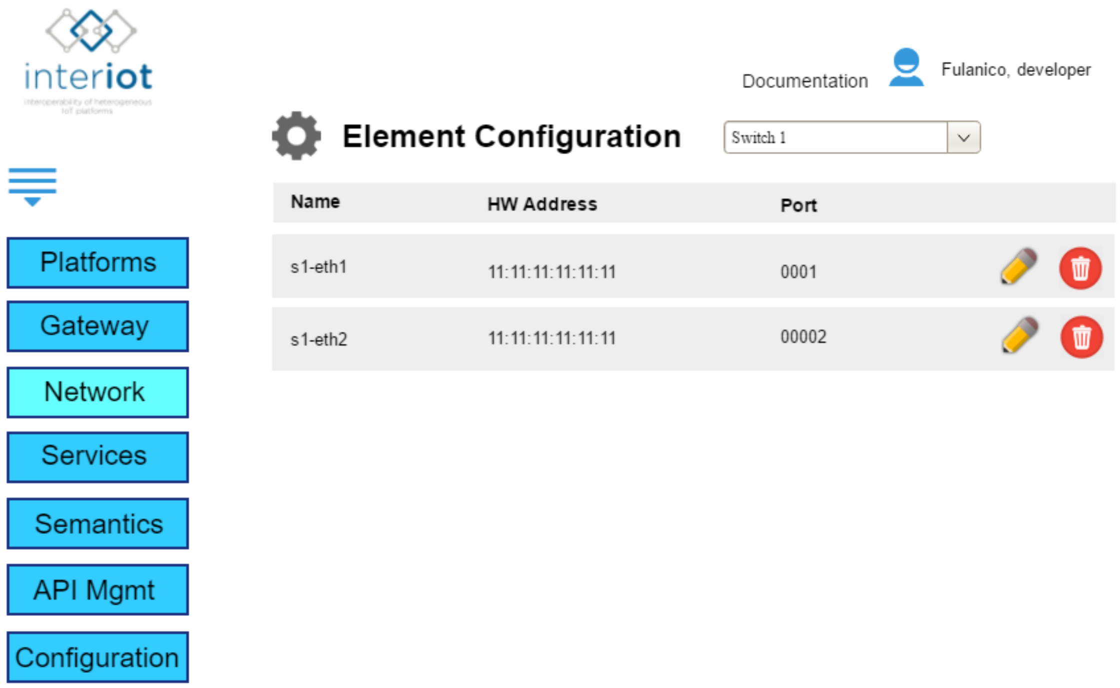
No. Ports

No. Flows

Manage permissions

	User	Rights	Actions
☑	Port user	Read	All
□	Terminal user	Write	Database

Figure 50. INTER-FW web application add network elements view

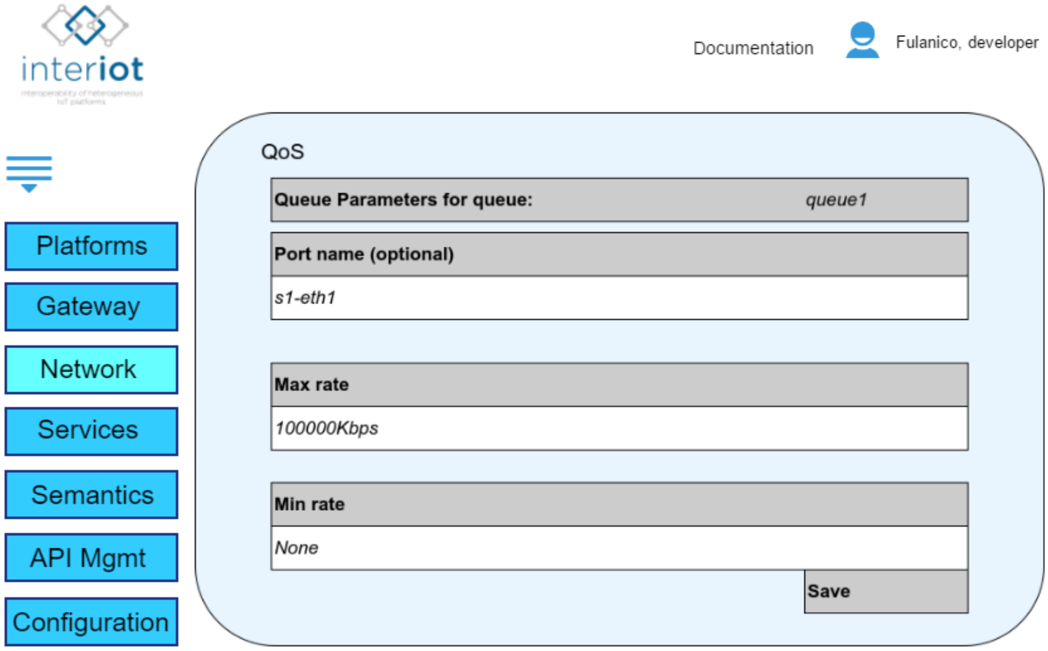


Element Configuration

Switch 1

Name	HW Address	Port
s1-eth1	11:11:11:11:11:11	0001
s1-eth2	11:11:11:11:11:11	00002

Figure 51. INTER-FW web application network element configuration view



QoS

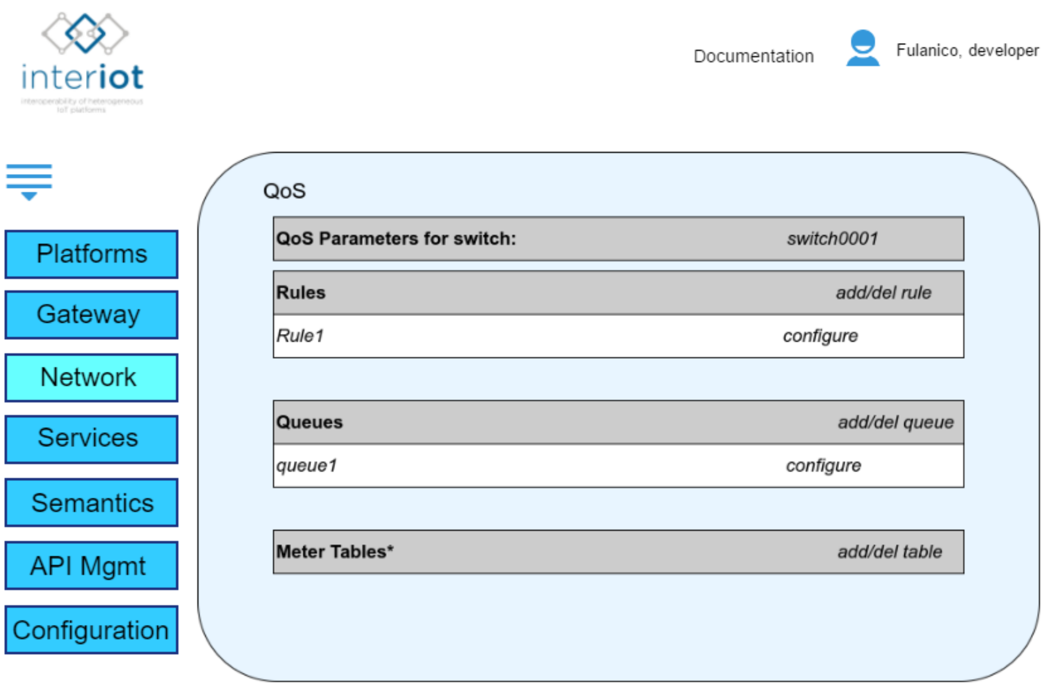
Queue Parameters for queue: *queue1*

Port name (optional): *s1-eth1*

Max rate: *100000Kbps*

Min rate: *None*

Figure 52 QoS queue configuration



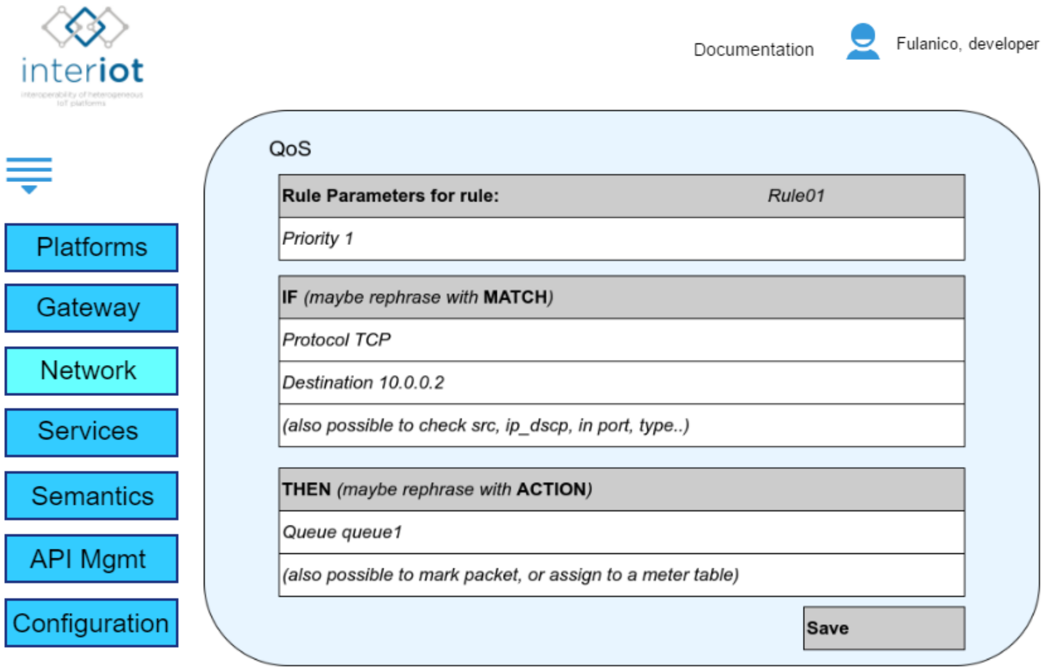
QoS

QoS Parameters for switch:	*switch0001*
Rules	*add/del rule*
Rule1	*configure*

Queues	*add/del queue*
queue1	*configure*

Meter Tables*	*add/del table*

Figure 53 QoS virtual switch configuration



QoS

Rule Parameters for rule:	*Rule01*
Priority 1	

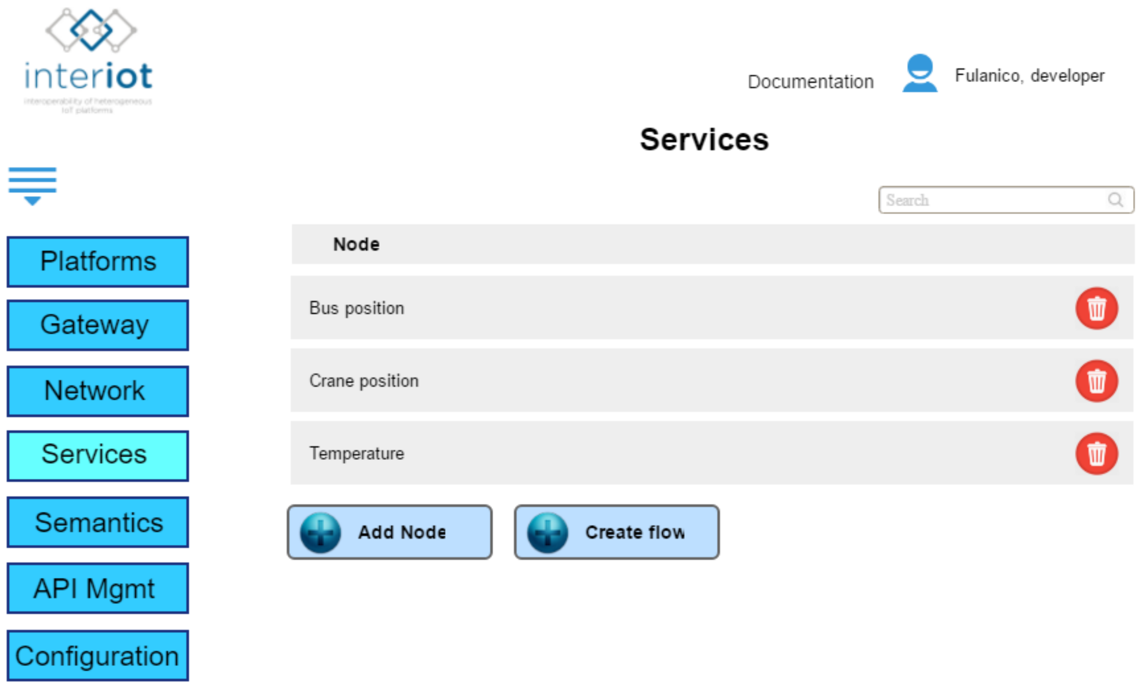
IF *(maybe rephrase with* **MATCH***)*
Protocol TCP
Destination 10.0.0.2
(also possible to check src, ip_dscp, in port, type..)

THEN *(maybe rephrase with* **ACTION***)*
Queue queue1
(also possible to mark packet, or assign to a meter table)

Figure 54 QoS rules configuration view

3.5.5 Service view

In the service main view main page (Figure 55), there is a list of registered nodes of Node-RED. There are two buttons to add or edit nodes (Figure 56) and create or edit flows (Figure 57) through an embedded version of Node-Red. Each node has a button to delete it. Figure 36), there is a list of registered nodes of Node-RED. There are two buttons to add or edit nodes and create or edit flows through an imbibed version of Node-Red. Each node has a button to delete it.



Node
Bus position
Crane position
Temperature

Figure 55. INTER-FW web application service view

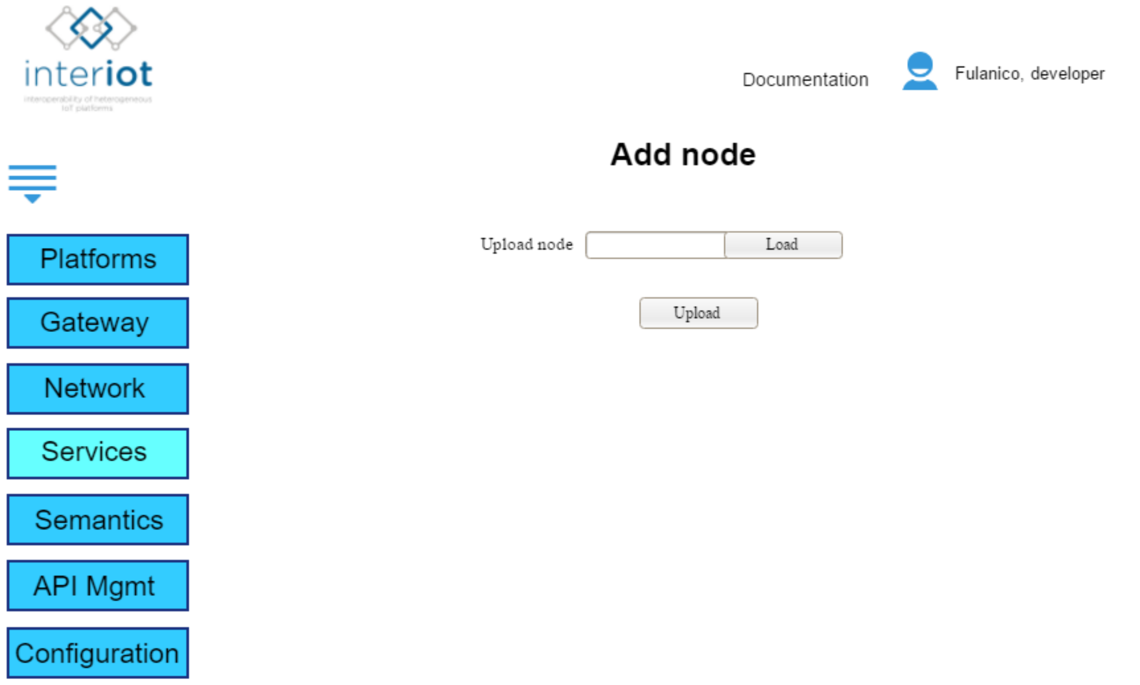


Figure 56. INTER-FW web application add node view

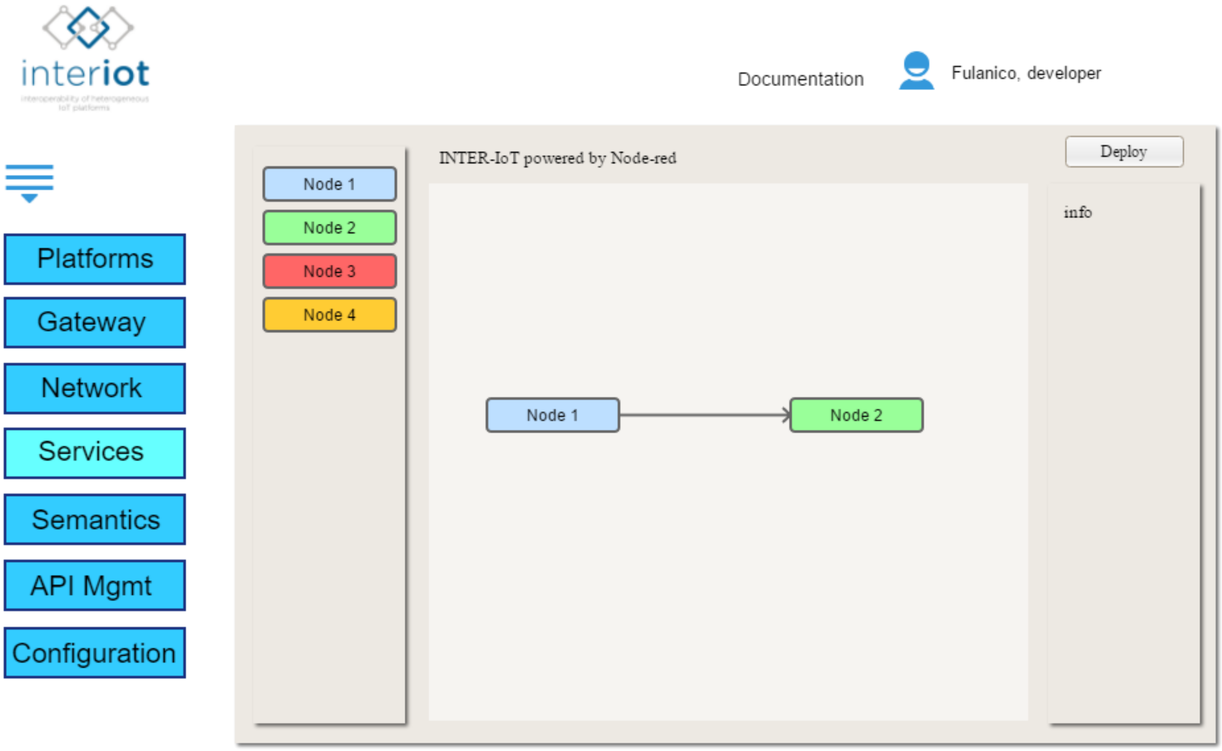
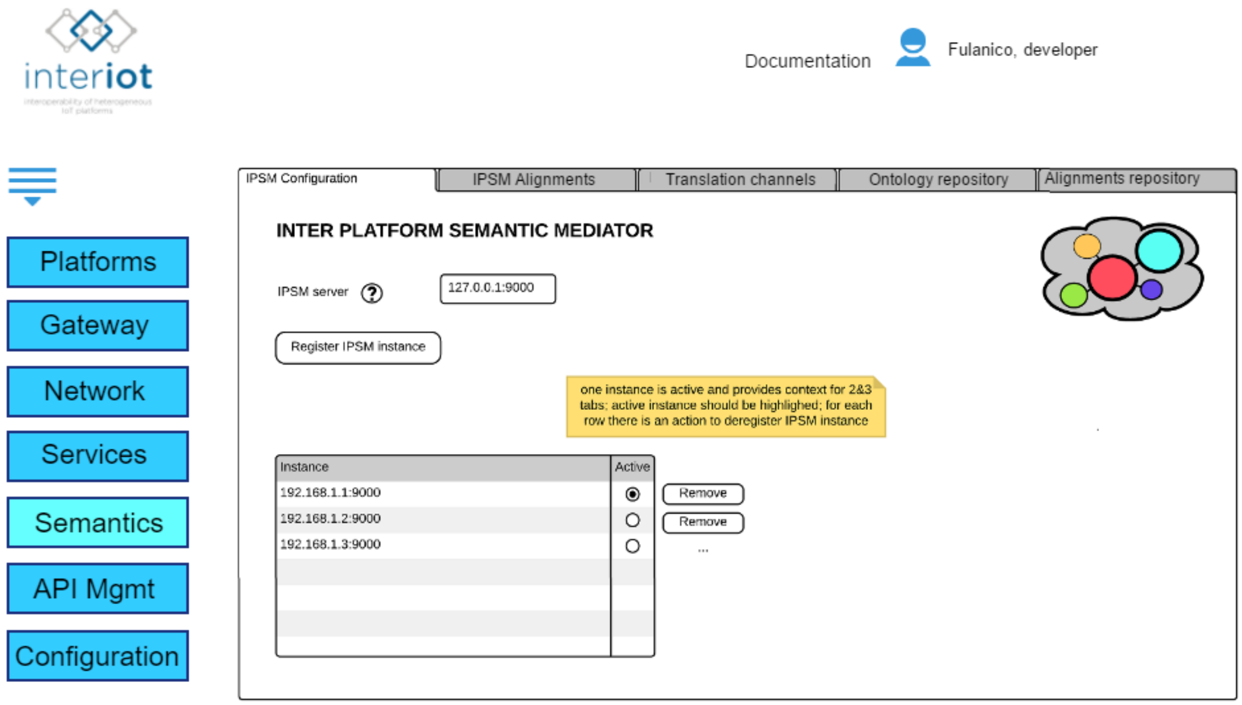


Figure 57. INTER-FW web application add flow view

3.5.6 Semantic view

In the semantic main view (Figure 58 and following), it is shown the configuration of the IPSM module, including semantic alignments enabled, translation channels, general configuration of the module (URL, ports) and two repositories, one for ontologies and another for alignments, so that they can be shared among different instances. Figure 39 and following), it is shown the configuration of the IPSM module, including semantic alignments enabled, translation channels, general configuration of the module (URL, ports) and two repositories, one for ontologies and another for alignments, so that they can be shared among different instances.



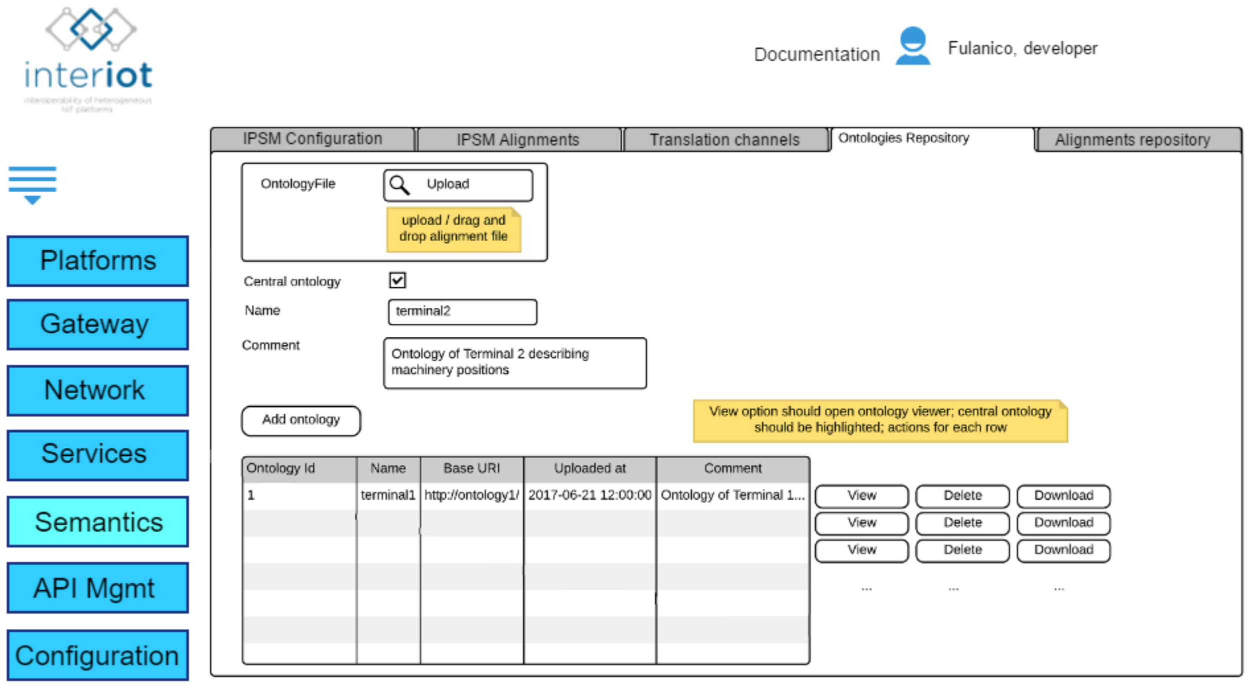
INTER PLATFORM SEMANTIC MEDIATOR

IPSM server ? 127.0.0.1:9000

one instance is active and provides context for 2&3 tabs; active instance should be highlighed; for each row there is an action to deregister IPSM instance

Instance	Active
192.168.1.1:9000	●
192.168.1.2:9000	○
192.168.1.3:9000	○

Figure 58. IPSM general configuration view



OntologyFile Upload

upload / drag and drop alignment file

Central ontology ☑

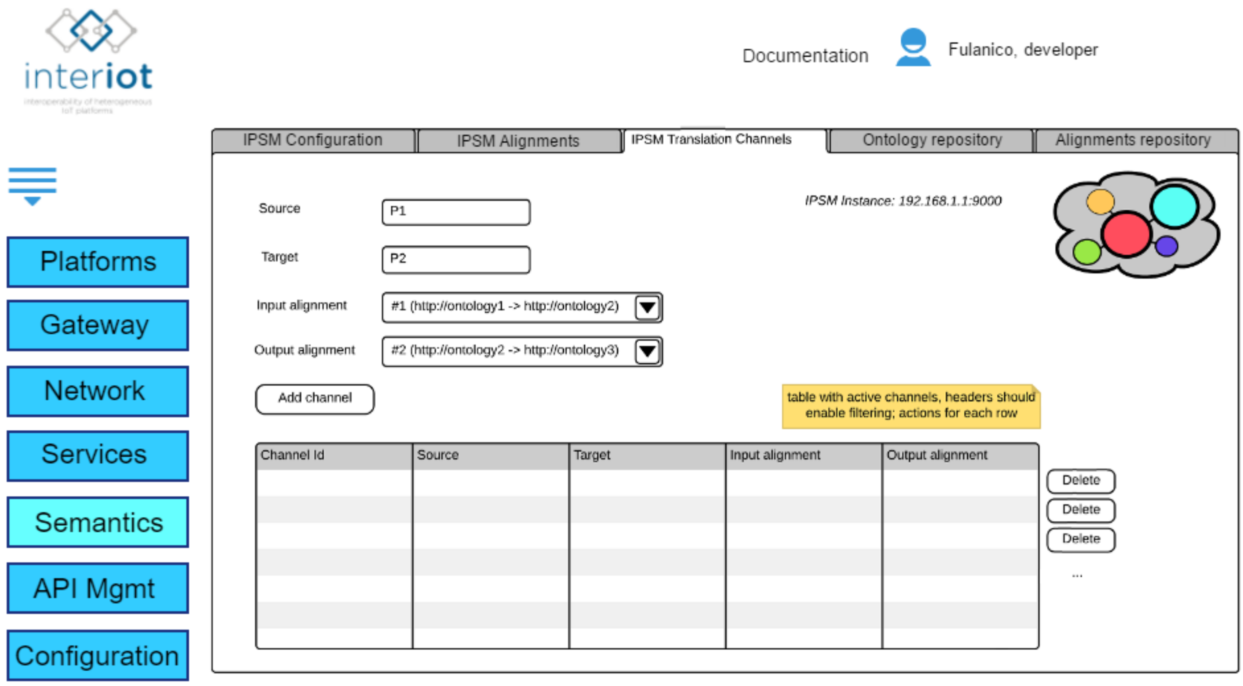
Name terminal2

Comment Ontology of Terminal 2 describing machinery positions

View option should open ontology viewer; central ontology should be highlighted; actions for each row

Ontology Id	Name	Base URI	Uploaded at	Comment
1	terminal1	http://ontology1/	2017-06-21 12:00:00	Ontology of Terminal 1...

Figure 59. Ontologies repository view



IPSM Instance: 192.168.1.1:9000

Source: P1

Target: P2

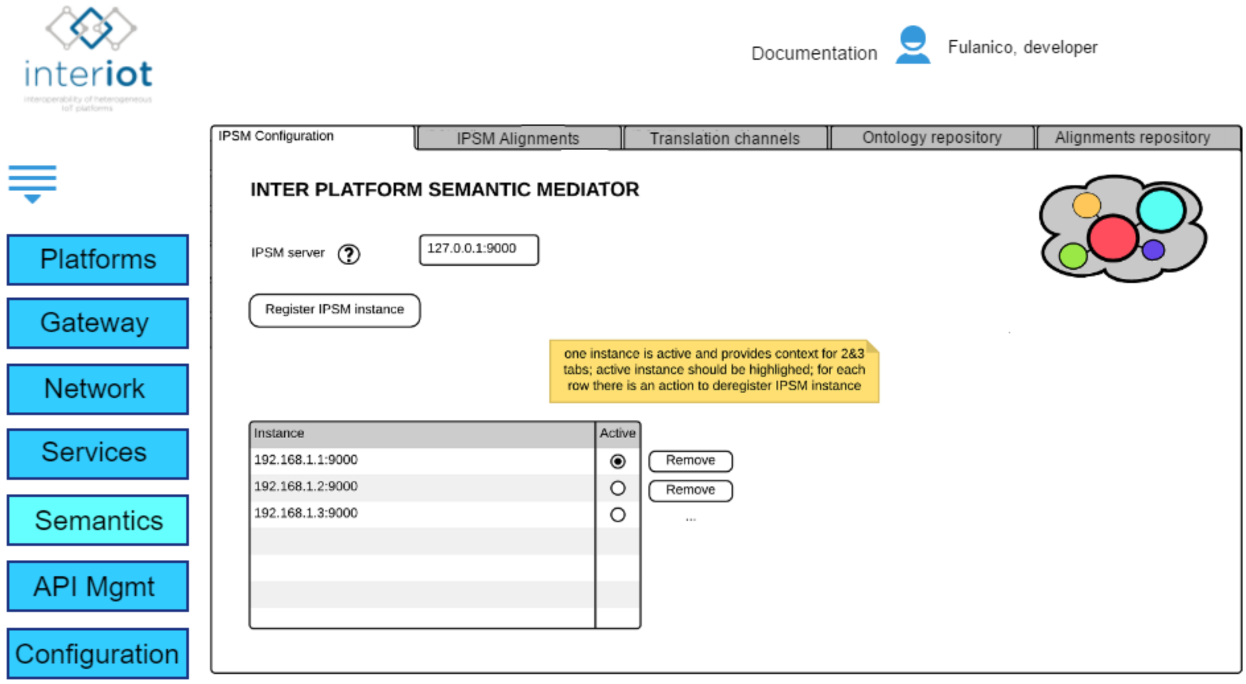
Input alignment: #1 (http://ontology1 -> http://ontology2)

Output alignment: #2 (http://ontology2 -> http://ontology3)

table with active channels, headers should enable filtering; actions for each row

Channel Id	Source	Target	Input alignment	Output alignment

Figure 60. IPSM Translation channels view



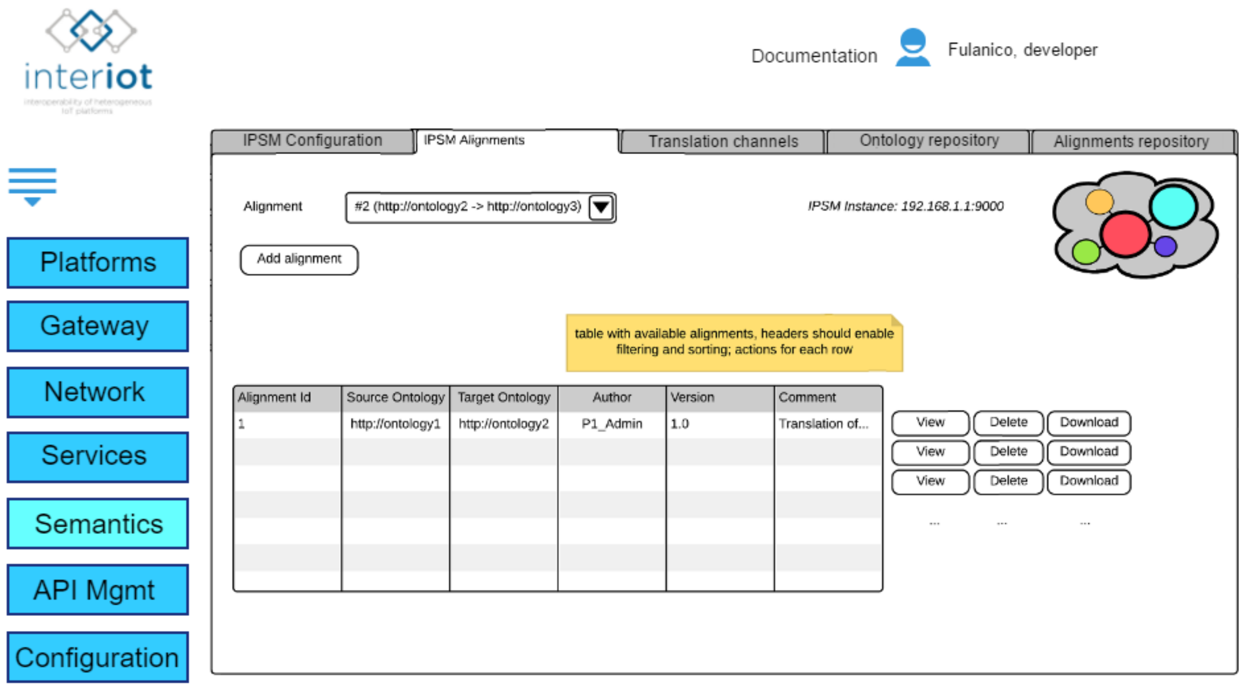
INTER PLATFORM SEMANTIC MEDIATOR

IPSM server ? 127.0.0.1:9000

one instance is active and provides context for 2&3 tabs; active instance should be highlighed; for each row there is an action to deregister IPSM instance

Instance	Active
192.168.1.1:9000	●
192.168.1.2:9000	○
192.168.1.3:9000	○

Figure 61. IPSM Configuration view



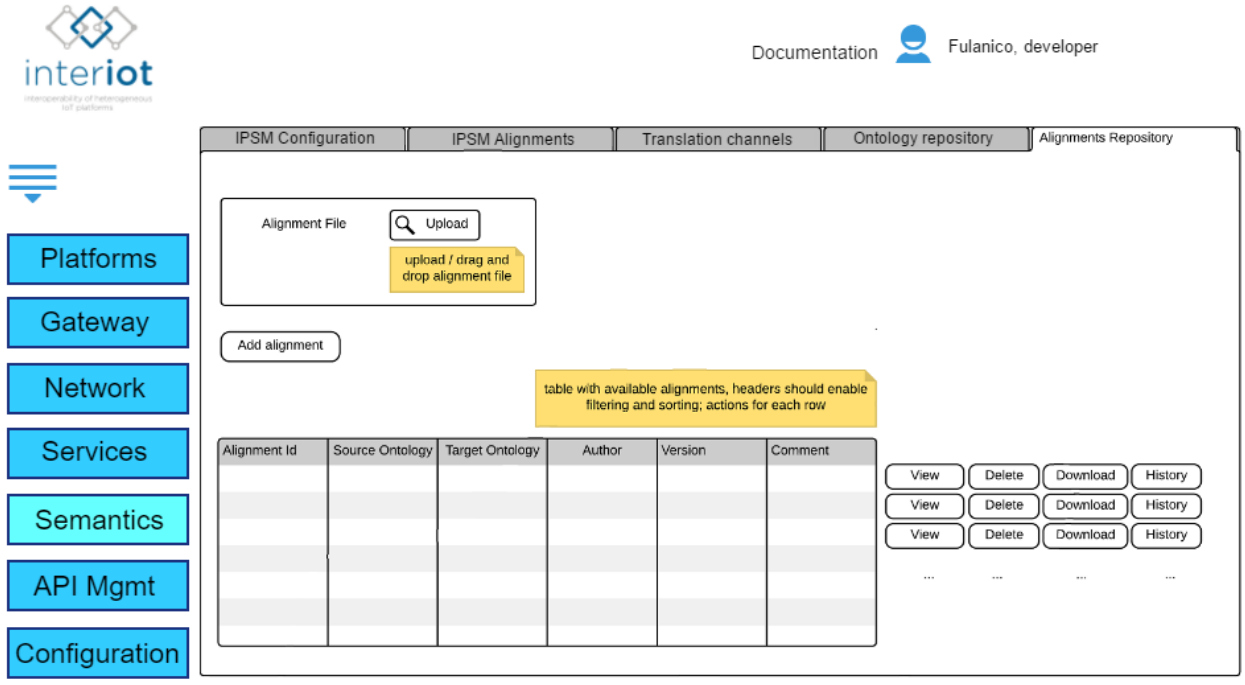
Alignment #2 (http://ontology2 -> http://ontology3)

IPSM Instance: 192.168.1.1:9000

table with available alignments, headers should enable filtering and sorting; actions for each row

Alignment Id	Source Ontology	Target Ontology	Author	Version	Comment
1	http://ontology1	http://ontology2	P1_Admin	1.0	Translation of...

Figure 62.IPSM Alignment configuration view



upload / drag and drop alignment file

table with available alignments, headers should enable filtering and sorting; actions for each row

Alignment Id	Source Ontology	Target Ontology	Author	Version	Comment

Figure 63. Alignments repository view

3.5.7 API management view

In the API main view (Figure 64), there is menu to manage all the APIs through an embedded version of the API Manager.

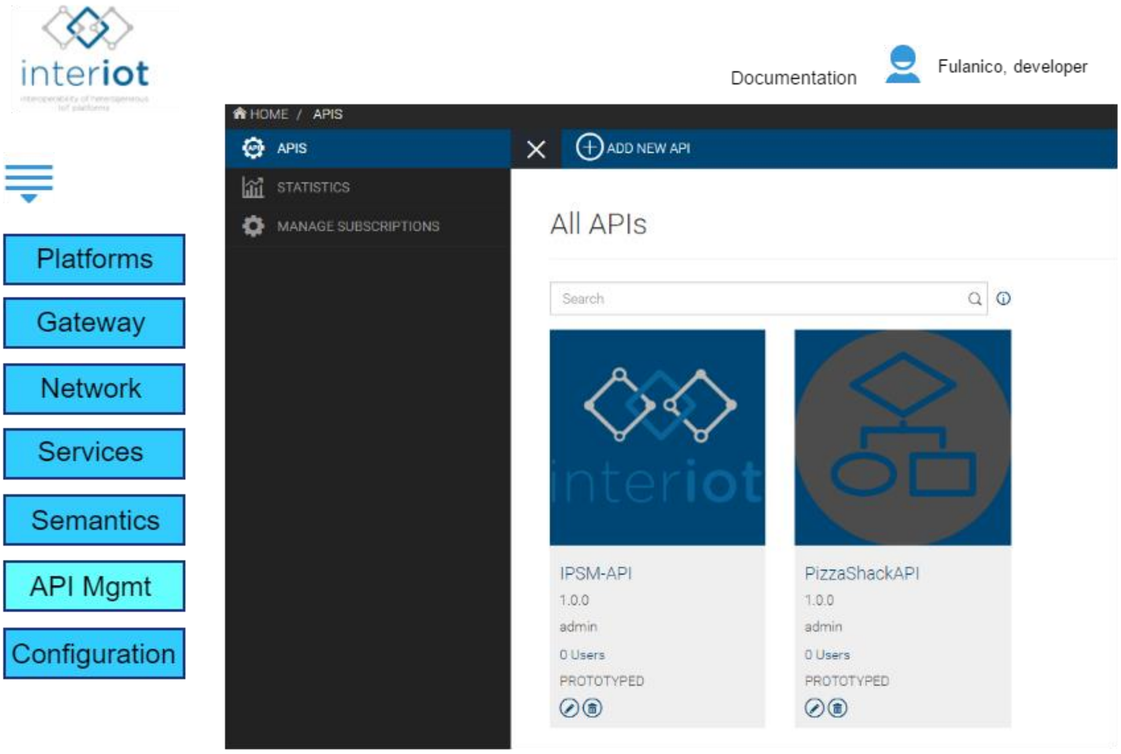


Figure 64. INTER-FW web application API management view

3.5.8 Configuration view

In the configuration main view (Figure 65), there are different general configuration issues such the log and the definition of new types of platforms.

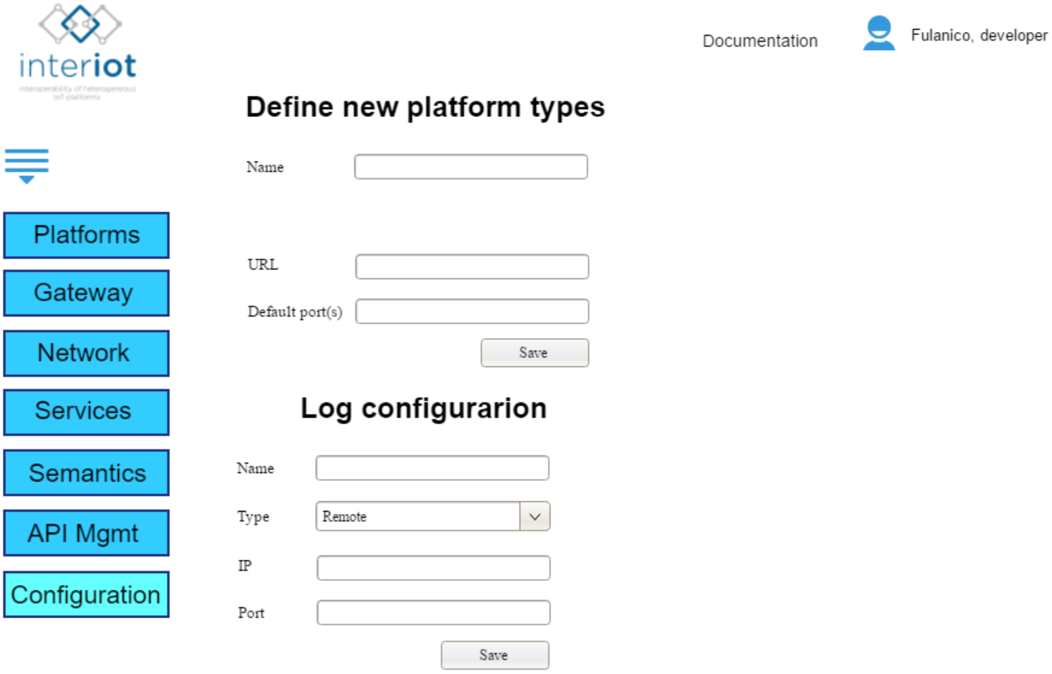


Figure 65. INTER-FW web application configuration view

3.6 INTER-FW relation with INTER-Layer

The solutions provided by INTER-Layer are focus on specific structures depending of the layer, so that, to manage these solution from a higher and more abstract global point of view, we propose INTER-FW.

INTER-FW then, aims to provide mechanisms, tools and helper content to properly manage the Layer Interoperability Infrastructure (LIIs) and Interoperability Layer Interfaces (ILIs). Each solution of INTER-Layer will communicate with INTER-FW for managing the instances and the configuration of them. We will see in more detail how every layer interacts with INTER-FW.

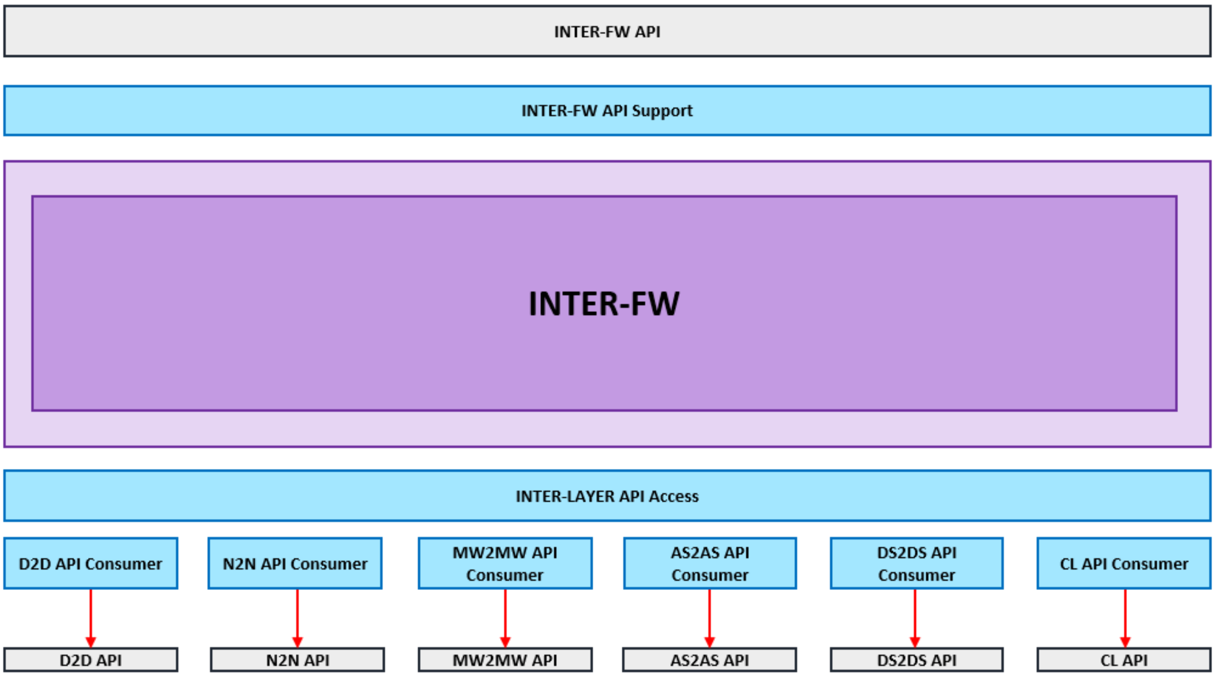


Figure 66 Relation with INTER-Layer diagram

Moreover, each INTER-Layer solution will provide an API to be consumed by INTER-FW in order to manage the content and configuration located within the solution, as we can see in Figure 66. Additionally, the INTER-FW API will expose together with its own extra value API calls, the APIs provided by each layer as a suite for future application to use not only the API provided by a single solution but several solutions combined.

Regarding how the APIs of each layer will be consumed by the framework, in some cases, as N2N and AS2AS, INTER-FW will facilitate graphic tools to visualize and manage the interoperability achieved in the layers and the user will utilize these tools to handle the elements within the solutions. In other cases, as in D2D and MW2MW the management, not of the element within the solution but of instances of the solution will be carried out by the framework working with Docker as a virtual container tool for this management.

We will explain a little bit more how this container tool works and how the solutions will be *dockerize* and handled by tools located within the framework.

Docker[37i] is software container platform that encapsulates applications to run and manage them side-by-side in isolated containers to obtain better performance and compute density. These containers

[37] https://www.docker.com/

can communicate with each other through a Docker network specifying the direction and port, and the Docker tool can handle the lifecycle of the containers in a way that this packages of software run isolated on a shared operating system being started, ran or stopped when needed.

Despite other virtualization methods or machines, containers do not build a full operating system, instead only libraries and settings required to make the software work as needed.

The solutions implemented in the layers should be wrapped then in Docker containers so that the lifecycle of them is managed in the framework.

Docker-swarm provides a tool to observe and manage a group of Docker nodes as if it were only one cluster so that you can start several Docker containers related between them at the same time and treat them as a whole.

A swarm is a cluster of Docker nodes where the solutions are deployed, with this tool you enable the initialization of the swarm or you can join a container to an existing swarm. Additionally, a CLI and an API is included to set commands to manage the swarm nodes.

This tool will be user to *clusterize* different solutions from the same layer (e.g several virtual gateways) and manage them as an ensemble. With this Dockerization of solutions INTER-FW also includes security, managing the Users/Roles that has access to the different containers or cluster of containers.

Additionally, INTER-FW will provide security mechanisms, as seen in Section 3.4, to provide exclusive access to the allowed client to each solution, and defining roles with different permits. This mechanism will be implemented with Cryptex tool as a secure storage and cryptographic key retrieval for Node.js. Thus, all access and role information will be secretly stored and ciphered in this tool.

Finally, the main relation between INTER-LAYER and INTER-FW is shown in the Web application of the frameworks that works as a portal so easy access the functionalities previously described. We can see this relationship in a more detail manner in the section 3.5 where we describe each of the tabs used to interact with the INTER-Layer solutions through its APIs. Hence, INTER-FW GUI structures each one of it modules (tabs) depending on the architecture and components of each layer solution, showing the structure of each layer and information about the components managed by them (devices, network elements, platforms, applications, etc). More specifically:

- D2D: the containers running the virtual gateways instances are managed with the GUI application.
- N2N: the topology of the network is visualized through the applications and the configuration of the switches, including QoS, are done within the GUI application.
- MW2MW: the different platforms connected to INTERMW are shown. The instance of INTERMW is unique.
- AS2AS: different containers run one instance of AS2AS for each user.
- DS2DS: the IPSM instance is fully manageable from the Web Console, including adding/removing channels, alignments and ontologies.
  Cross-Layer: All cross-layer functionalities related with the interaction of different layer solutions will be available through the FW (e.g utilization of the IPSM by the AS2AS environment).

4 INTER-API

4.1 API Management

API management is a process that encompasses publishing, documenting and overseeing application programming interfaces (APIs) in a secure, scalable environment. It allows organizations that publish their APIs to monitor the interfaces' lifecycles and also make sure that needs of both internal and external developers, as well as applications that are using the APIs, are being met. It thus provides core competencies for ensuring successful API usage in developer engagement, business insights, analytics, security and protection.

Software that implements API management typically provides the following functions:

- Automation and control of connections between the API and applications that use it.
- Ensuring consistency between multiple API implementations and versions.
- Traffic monitoring from individual applications.
- Wrapping of the API into security procedures and policies, thus protecting it from misuse.
- Memory management and caching mechanisms to improve application performance.
- Sandbox environment for developers. Coupled with good API documentation, it enables potential customers to try the API out.

Today, APIs are less dependent upon conventional service-oriented architectures (SOA) and more on lightweight JSON and REST services. It is however possible to convert existing SOAP, JMS or MQ interfaces into RESTful APIs with JSON content.

Typical components of API management systems are:

- Gateway; a server that acts as an API front-end, which receives API requests and passes requests to the back-end. It then passes the responses back to the requester. It can modify the requests and responses on the fly, and it can also provide the functionality to support authentication, authorization, security, audit and caching.
- Publishing tools; a collection of tools that API providers use to define APIs, for instance using the OpenAPI or RAML specifications. They also generate API documentation, manage access and usage policies, and can as well be used for testing and debug purposes.
- Developer portal/API store; a community site that enables user's access to documentation, tutorials, sample code, software development kits, and so forth. Here users can also manage subscription keys and obtain support from the API provider and the community.
- Reporting and analytics; a collection of tools that monitor API usage and load.
- Monetization service; a service that monetizes API usage.

An API management solution could be implemented in three different ways. The first one is as a **proxy**, where the API manager "sits" between the API and API's user, processing all the traffic between these two. In this way, API manager can implement caching and protection of customer's API from traffic spikes. However, a proxy also drives the cost up, as well as introduces additional privacy and latency issues. Apigee[38], Mashape[39] and Mashery[40] provide API management solutions that work as a proxy.

An API management solution could also work as an **agent**, that integrates itself with user's server and doesn't act as the middleman between the API and the user. API managers that work as agents typically don't suffer from latency issues and dependency on a 3rd-party, through which all the traffic passes. However, on the other end, features like caching are not easy to implement. 3scale offers a product of this type.

Some API managers work as **hybrids**, incorporating both a proxy and an agent. For example, the proxy could be used for caching, and agent for authentication. Apigee and 3scale are moving towards this solution.

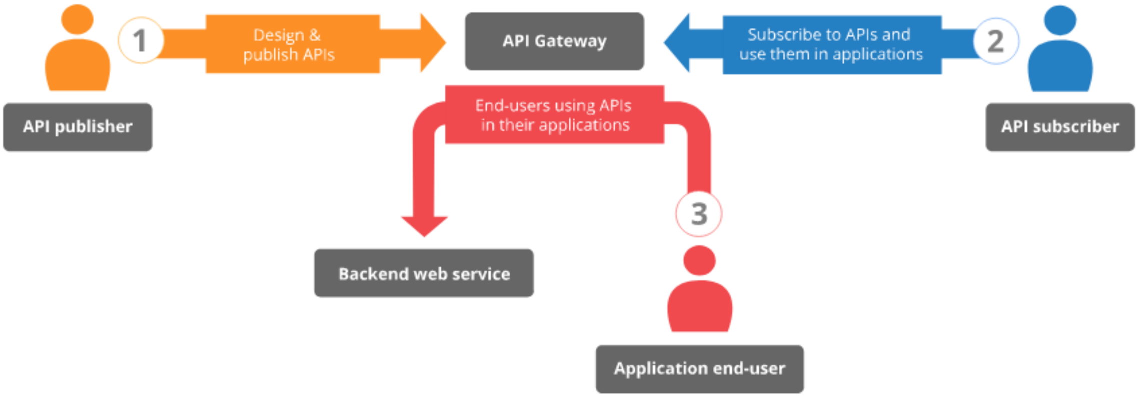


Figure 67. INTER-API manager structure

API management solutions

Many vendors offer their own API management solutions. Microsoft offers Azure API management[41], as well as do Oracle[42], IBM[43], WSO2[44] and Red Hat (3scale)[45] and others. Majority of solutions are implemented as proxies, while others are hybrid. Majority of them are directed towards SMEs, except for WSO2's API manager, Microsoft's Azure API manager, Mashape and 3scale, which are also suitable for enterprises. CA Layer7 API[46] manager is unique, as it has advanced support for mobile applications, while WSO2 API manager is open source. Next to WSO2, open source API managers are also Kong[47] and ApiAxle[48].

[38] https://apigee.com/api-management/#/homepage
[39] https://www.mashape.com/
[40] https://www.mashery.com/
[41] https://azure.microsoft.com/es-es/services/api-management/
[42] http://www.oracle.com/technetwork/middleware/api-manager/overview/index.html
[43] http://www-03.ibm.com/software/products/en/api-connect
[44] https://wso2.com/api-management/
[45] https://www.redhat.com/en/technologies/jboss-middleware/3scale
[46] https://www.ca.com/us/products/ca-api-management-saas.html
[47] https://getkong.org/
[48] http://apiaxle.com/

Different API managers cover different requirements. Some API managers, such as IBM's, Microsoft's and Oracle's, are part of a product or service, while others, such as WSO2's, are free. Some cannot even be deployed on premises, such as Microsoft's Azure.

For INTER-IoT would like to use an open source API manager, that is highly customizable and feature-full. Out of handful open source API managers presented above, WSO2 is the most popular and feature-complete, as well as highly customizable, and is thus an ideal choice for us.

4.2 API description

4.2.1 API Specification tools

API specification and documentation is a key factor for technology adoption, regardless whether the software is being used internally or by third-parties. With the growth of publicly available APIs, some de facto standards and tools have been emerging. The top specification formats used are OpenAPI Specification (Swagger), RAML and API Blueprint. This document will introduce briefly each of them and will show use examples and possible applications within the domain of INTER-IoT. In their most basic definition, API specification tools generate HTML and CSS code to display API methods, parameters, values, requests, responses, code samples, and more.

4.2.1.1 Swagger/OpenAPI Specification

The OpenAPI Specification (formerly known as Swagger[49]) is both a specification format and a framework for generating documentation, server and client API code. Because of its wide adoption, Swagger is becoming a standard for API specification.
Swagger-enabled API allows one to get interactive documentation, client SDK generation and discoverability. Swagger is language agnostic and uses JSON notation to describe the API documentation.
Swagger is open source and is widely supported by the developers' community. It has a strong ecosystem providing tools, code generators, etc.

4.2.1.2 RAML

RAML[50] is a specification that uses YAML notation and is specially intended for REST based APIs. It is open-sourced and has community support providing tooling and integration.

4.2.1.3 API Blueprint

API Blueprint[51] uses Markdown syntax to describe the APIs and the format is also open-sourced. APIARY, the promoter of API Blueprint format was recently bought by Oracle. The company offers a set of generators and tools to support the API design and implementation.

4.2.1.4 HYDRA-CG

A Hypermedia description language expressed via standards such as JSON-LD to make it easy to support Linked Data and interaction with other data sources[52].

[49] http://swagger.io/

[50] http://raml.org/

[51] https://github.com/apiaryio/api-blueprint

[52] http:// hydra-cg.com

4.2.2 Comparison

But by far the most mature tool is Swagger with its OpenAPI specification (originally Open Api Initiative) and the biggest community. Since all three tools use different data formats (JSON, YAML, Markdown), we can choose a tool that is suited to our other technological choices in INTER-IoT. One large difference between two primary contenders, Swagger and RAML, that might influence our decision as well, might be their underlying ideology. Swagger's is language-agnostic and thus might be too-constricted, while RAML's function methodology through usage of nested resources and sub-resources might appear more aesthetically intuitive, promoting deeper hierarchical understanding of the API instead of simpler cause-and-effect understanding, as promoted by Swagger.
All four above mentioned API Specification tools encompass the full API life-cycle, though RAML lacks code-level tooling. All three API Specification tools provide stub generators.
Both Swagger and RAML have rich ecosystems and large communities, while API Blueprint is still relatively small. By far the biggest community belongs to Swagger, though, due to its versatility and it being a de-facto standard for API specification. HYDRA, on the other hand, is a completely different approach since it uses semantic description approach, and can complement the other three.

4.2.3 Other API description languages and solutions

There are other API DL projects, but most of them are still in preliminary phases in terms of adoptions or functionalities offered. Some of them are:

- **Slate**[53]: A tool for documenting the APIs in a neat, attractive way. It uses markdown format and is open source. It provides a way to document the API but however it fails offering advanced features such as interactive tools, design tools, code generation or validation.
- **I/O Docs**[54]: Interactive documentation tools for REST APIs, it provides documentation specification based on JSON Schema format.

Several services act as API Directories, offering a unified repository to search API-related documentation. Some of the most known are:

- **ProgrammableWeb**[55]: Biggest, manually curated API directory on the Internet.
- **APIHound**[56]: APIHound is continuously examining billions of pages to locate the best API resources. It currently lists more than 50.000 APIs.
- **{API}Search**[57]: An experimental API Search service to help discover APIs on the web. The service uses the APIs.json proposed discovery format.

There are plenty of tools to test and building REST APIs, available in all platforms and formats.Three very popular tools are:

- **CURL**: One of the more popular libraries and command line tools, curl[58] allows you to invoke various protocols on various resources.
- **HTTPIE**: httpie[59] is a very flexible and easy to use client for interacting with

[53] https://lord.github.io/slate/

[54] https://github.com/mashery/iodocs

[55] https://www.programmableweb.com/

[56] http://apihound.com/apifinder

[57] http://apis.io/

[58] https://curl.haxx.se/

[59] https://httpie.org/

- **Postman**[60]: Full-blown API testing requires the ability to capture and replay requests, use various authentication and authorization schemes and more. The previous command line tools allow for much of this but Postman is a newer desktop application that makes these activities easier for teams of developers.

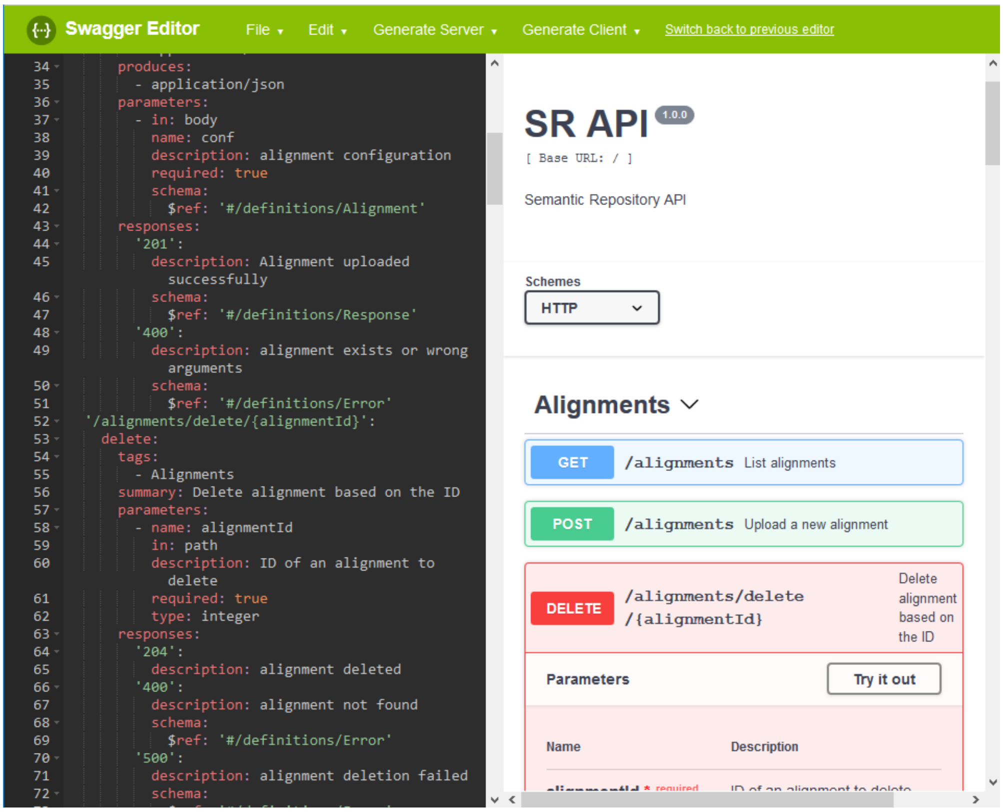
4.2.4 Conclusions

It seems that currently the three major players on the API description scene are Swagger, RAML and API Blueprint. Of these three, RAML and Swagger present more features such as interactive or design tools, whereas API Blueprint competes to be the easiest to deploy for fast prototyping. For INTER-IoT, any of them could be used, since they provide sufficient support API creation, documentation and testing.

4.3 API design

Following the reasoning in section 4.2, all Inter-Layer components listed in section 2.4 and INTER-API will document their API in the Swagger-JSON format. Single layer APIs will be accessed through an instance of the WSO2 API manager that acts as the main entry point for API management, access and usage.
The reader interested in a detailed documentation about each API layer is referred to the documentation in swagger format published as part of deliverable D4.5, where can find a detailed description if each API REST layer.

An example of the Swagger visualisation and test environment provided through the Swagger Editor for the IPSM repository component is provided in the following figure:



```
    produces:
      - application/json
    parameters:
      - in: body
        name: conf
        description: alignment configuration
        required: true
        schema:
          $ref: '#/definitions/Alignment'
    responses:
      '201':
        description: Alignment uploaded successfully
        schema:
          $ref: '#/definitions/Response'
      '400':
        description: alignment exists or wrong arguments
        schema:
          $ref: '#/definitions/Error'
  '/alignments/delete/{alignmentId}':
    delete:
      tags:
        - Alignments
      summary: Delete alignment based on the ID
      parameters:
        - name: alignmentId
          in: path
          description: ID of an alignment to delete
          required: true
          type: integer
      responses:
        '204':
          description: alignment deleted
        '400':
          description: alignment not found
          schema:
            $ref: '#/definitions/Error'
        '500':
          description: alignment deletion failed
          schema:
```

SR API 1.0.0

[Base URL: /]

Semantic Repository API

Schemes: HTTP

Alignments

GET /alignments List alignments

POST /alignments Upload a new alignment

DELETE /alignments/delete/{alignmentId} Delete alignment based on the ID

Parameters

Name	Description
alignmentId * required	ID of an alignment to delete

Figure 68. Swagger editor IPDM component

[60] https://www.getpostman.com/

Below we provide a list of REST methods and model definitions exposed by single layers and the main functionalities provided.

4.3.1 Inter-Layer REST API Endpoints

Following the identified needs for the exposure of Inter-layer components in 2.5.2 a set of OpenApi definitions has been delivered for each Inter-Layer component. Those have been integrated in the WSO2 API manager and published to the INTER-IoT API Store[61]. There, users can register and gain access to the Inter-IoT cloud deployment.

In addition to controlled (secure) and harmonised access to all Inter Layer components, the API Manager solution supports composing of several operation in higher level APIs and possibly adaptation of endpoints according to user needs.

4.3.1.1 Device Layer Interoperability (D2D) – Gateway API summary

This API is to manage a single instance of the Gateway (understanding instance as a couple of connected virtual/physical parts of the gateway) and supports the core functionalities of the Gateway.

The current organisation of exposed APIs is Inter-Layer centric, i.e. they directly expose the functionality of single layers. In the second phase, these will be combined and restructured in a way to provide a more meaningful set of hierarchical resources from the INTER FW point of View. The user will thus have a possibility to choose between the two approaches in accordance to the specific interoperability scenario to be solved.

More functionalities can be added by the means of extensions, these extensions can dynamically export special API java classes that will be collected and published in the API as new endpoints in order to expand the interaction with the Gateway.

Tag: virtual

Operation	Description
GET /virtual/platform	Show IoT Platform Configuration
GET /virtual/config	Get the Virtual Gateway configuration
POST /virtual/config	Set the Virtual Gateway configuration
GET /virtual/extensions	List the available extensions for the virtual gateway

Tag: physical

Operation	Description
GET /physical/config	Get the Physical Gateway configuration
POST /physical/config	Set the Virtual Gateway configuration
GET /physical/extensions	List the available extensions for the physical gateway
GET /physical/devices	Get a list of devices connected to this Gateway

Tag: device

Operation	Description
GET /device/{deviceUid}	Get Device Description
GET /device/{deviceUid}/start	Connect device
GET /device/{deviceUid}/stop	Disconnect device
GET /device/{deviceUid}/read	Read last device state
POST /device/{deviceUid}/write	Write device state

[61] https://vmplsp02.westeurope.cloudapp.azure.com:9443/store/

GET /device/{deviceUid}/status	Get device status

4.3.1.2 Network Layer Interoperability (D2D)

Tag: Switches

Operation	Description
GET /stats/switches	List Switches
GET /stats/desc/{id}	Switch Basic Description

Tag: Flows

Operation	Description
GET /stats/flow/{id}	Flow information
POST /stats/flowentry/add	Addition of a new flow entry
POST /stats/flowentry/modify	Modify flow entry in a switch
POST /stats/flowentry/delete	

Tag: Tables

Operation	Description
GET /stats/table/{id}	Table information
GET /stats/tablefeatures/{id}	Table features information

Tag: Ports

Operation	Description
GET /stats/port/{id}	Port information
GET /stats/port/{id}/{port}	Port information
POST /stats/portdesc/modify	Modify port description

Tag: Roles

Operation	Description
POST /stats/role	Set a role

Tag: Queues

Operation	Description
GET /qos/queue/status/{id}	Get status of the queue
GET /qos/queue/{id}	Retrieve the queue configuration
POST /qos/queue/{id}	Set a queue into a port
DELETE /qos/queue/{id}	Delete the configuration of a queue

Tag: Rules

Operation	Description
GET /qos/rule/{id}	Retrieve the rules configuration
POST /qos/rule/{id}	
DELETE /qos/rule/{id}	Delete a Qos Rule

Tag: Meters

Operation	Description
GET /qos/meter/{id}	Retrieve the meters configuration
POST /qos/meter/{id}	
DELETE /qos/meter/{id}	Delete a Qos Meter

4.3.1.3 Middleware Layer Interoperability (INTERMW)

Tag: client

Operation	Description
POST /intermw/client/{clientId}	Register client
DELETE /intermw/client/{clientId}	Unregister client
POST /intermw/pull/{clientId}/{numberOfMessages}	Message pull request

Tag: platform

Operation	Description
DELETE /intermw/platform/{clientId}/{platformId}	Remove platform instance
POST /intermw/platform/{clientId}	Register a new platform instance.

Tag: thing

Operation	Description
DELETE /intermw/thing/{clientId}/{thingId}	Unregister the thing
DELETE /intermw/subscribe/{clientId}/{thingId}	Unsubscribe from the thing
POST /intermw/subscribe/{clientId}	Subscribes to the thing
POST /intermw/thing/{clientId}	Register the thing

4.3.1.4 Application Service Layer Interoperability (AS2AS)

Tag: auth

Authentication and Credentials

Operation	Description
GET /auth/login	Get the active authentication scheme
POST /auth/token	Exchange credentials for access token
POST /auth/revoke	Revoke an access token

Tag: settings

Runtime Settings

Operation	Description
GET /settings	Get the runtime settings

Tag: flows

Active Flow Configuration

Operation	Description
GET /flows	Get the active flow configuration.
POST /flows	Set the active flow configuration

4.3.1.5 Data and Semantics Layer Interoperability

IPSM API

Tag: Alignments

Operation	Description
GET /alignments	List alignments
POST /alignments	Upload new alignment
DELETE /alignments/{alignmentId}	Delete alignment based on the ID
GET /alignments/{alignmentId}	Get an alignment based on the ID

Tag: Channels

Operation	Description
GET /channels	List active IPSM channels
POST /channels	Create new channel
DELETE /channels/{channelId}	Delete channel based on the ID

Semantics Repository API

Tag: Alignments

Operation	Description
GET /alignments	List alignments
POST /alignments	Upload a new alignment
DELETE /alignments/delete/{alignmentId}	Delete alignment based on the ID
GET /alignments/get/{alignmentId}	Get an alignment based on the ID

Tag: Ontologies

Operation	Description
GET /ontologies	List ontologies
POST /ontologies	Upload a new ontology
DELETE /ontologies/delete/{ontologyId}	Delete ontology based on the ID
GET /ontologies/get/{ontologyId}	Get ontology based on the ID

5 Ethical issues

5.1 Introduction

Ethics is a central consideration to all INTER-IoT planning and development. As requested at the interim review, an ethical advisory board has been established. This board, within INTER-IoT, continuously reviews ethical issues. The aim of the committee is to ensure that ethical considerations and issues are addressed in the conduct of the research and development work undertaken within the project. The committee seeks to support and encourage the process of ethically conducted research to maintain the safety and well-being of participants and researchers to promote ethical values.

5.2 Ethics and INTER-FRAMEWORK

As stated above, INTER-FRAMEWORK is designed to configure, monitor, and manage different layers within INTER-IoT while providing a REST-like API to enable the use of the interoperable platform and INTER-IoT features. To provide this type of functionality, ethical consideration must be made of the types of data being utilised. The backend design must be guided by end user needs when handling each type of data and reflected in the requirements. The requirements are discussed in section 2.3. Additionally, the possible uses of INTER-IoT are purposefully wide ranging. This makes it difficult to exhaustively define all possible requirements. The goal then becomes to provide the end user with a customisable system capable of handling all sensitivity levels of data.

5.2.1 Data types

Primary focus of the ethical review of data management focuses on personal data and sensitive personal data.

Personal data means data which relate to a living individual who can be identified –

a) from those data, or
b) from those data and other information which is in the possession of, or is likely to come into the possession of, the data controller,

and includes any expression of opinion about the individual and any indication of the intentions of the data controller or any other person in respect of the individual.

Sensitive personal data means personal data consisting of information as to -

a) the racial or ethnic origin of the data subject,
b) his political opinions,
c) his religious beliefs or other beliefs of a similar nature,
d) whether he is a member of a trade union (within the meaning of the Trade Union and Labour Relations (Consolidation) Act 1992),
e) his physical or mental health or condition,
f) his sexual life,
g) the commission or alleged commission by him of any offence, or
h) any proceedings for any offence committed or alleged to have been committed by him, the disposal of such proceedings or the sentence of any court in such proceedings.

It is possible that INTER-IoT and INTER-FW will be used during processing of these types of data, so appropriate controls have been built into the back end to enable users to do this ethically by conforming to the data protection act.

5.2.2 Requirements for ethical data processing

The data protection act requires adherence to 8 principles:

1. Personal data shall be fairly and lawfully processed as defined in the data protection act.
2. Personal data shall be obtained only for one or more specified and lawful purposes, and shall not be further processed in any manner incompatible with that purpose or those purposes.
3. Personal data shall be adequate, relevant and not excessive in relation to the purpose or purposes for which they are processed.
4. Personal data shall be accurate and, where necessary, kept up to date.
5. Personal data processed for any purpose or purposes shall not be kept for longer than is necessary for that purpose or those purposes.
6. Personal data shall be processed in accordance with the rights of data subjects under the data protection act.
7. Appropriate technical and organisational measures shall be taken against unauthorised or unlawful processing of personal data and against accidental loss or destruction of, or damage to, personal data.
8. Personal data shall not be transferred to a country or territory outside the European Economic Area unless that country or territory ensures an adequate level of protection for the rights and freedoms of data subjects in relation to the processing of personal data.

Principle 1, 2, 3, 4, 5, 6 and 8 require the active management of end users. Principle 7 is our key focus with the use cases being the key driver in identifying requirements necessary.

Requirement	**Principle**	**Actions**
System security and privacy [27, 28]	1, 2, 6, 7	Controlling access to the INTER-FW and other systems connected to INTER-FW is necessary to allow users to determine the who can access what data and how it can be utilized.
Auditability [58]	1, 2, 5, 7, 8	A history including the source of data, modifications to data, and viewing of data is necessary to ensure that proper usage is controlled.
Constraints on processing of personal and health data [61, 62]	1, 2, 6, 7	Providing the tools so a user can insure that they are collecting the correct data and the data is being used only for the purposes the deem appropriate is necessary to allow appropriate data management for processing of health and personal data.
Confidentiality [69]	1, 7	Providing the tools to ensure data can be isolate within the system so no other users will know about it or a description of it is necessary.
Users manage how their public data is seen [77]	1, 7	Allowing users to manage how their data is seen, provides a way to ensure data is processed appropriately and ethically.
Manage user permission [260, 262]	1, 2, 7	Access to data needs to be controlled on a per user basis. Tools for controlling access to specific parts of INTER-IoT are provided by INTER-FW.

Access to personal data needs to be previously authorized [263]	1, 2, 7	Access to personal and sensitive data needs to be controlled on a per user basis. Tools for controlling access to specific parts of INTER-IoT are provided by INTER-FW.
Provision of authentication credentials [63]	1, 2, 6, 7	Providing effective methods to prevent unauthorised access to INTER-FW is necessary and tied closely to system security.

Table 21 Ethical issues related requirements

INTER-FW cannot guarantee that it's users will act in accordance with the data protection act and ethically, however we endeavour to provide them the tools to do so. In addition, to prevent improper usage of the tooling and the software, a set of rules of conduct or policies will be provided to let the user act properly. INTER-IoT demonstrators, pilots and proof of concepts will always follow this code of conduct, avoiding any bad use of the technology.

6 Conclusions and future work

In this document, a complete analysis and design of the INTER-FW and INTER-API is presented. This work, which will be reviewed and completed in M30, gives the main features to configure, administer and manage heterogeneous IoT platforms in scenarios where interoperability is a key factor, as it will be tested and validated in the three INTER-IoT pilots (INTER-LogP, INTER-Health and Cross-domain Pilot) and also the Open Calls who joined the project in mid-2017.

The features designed and reported are extracted from an exhaustive process of gathering requirements and feedback from the project consortium expertise and key stakeholders interviewed during the first phases of the execution.

As these requirements are also addressed in other technical tasks of this projects, it is worthy to show the process of fulfilment of these requisites and how they are addresses in different aspects in the parallel activities of the INTER-IoT project. Annex II show how identified requirements fulfil the identified use cases. Some of the use cases listed are expected to be analysed in next releases of the framework because they belong to different functional groups (extension framework, documentation…).

The identified use cases have been used as a basis to create a full design of the application, analysing the data model needed, entities relations and components needed to the INTER-FW managing and configuration tools implementation. The previous sections describe this engineering process, including the main security mechanisms.

The INTER-IoT project relies in a reference architecture and meta-data model to create IoT interoperability solutions. These results have been a fundamental baseline to instantiate the software architecture proposed in this document, as shown in section 3.2.

For the future version of the document, it is expected a full coverage of the use cases derivated of the software development framework for expansion, which will include the mechanisms to develop and deploy in runtime new adapters for the different interoperability layers such as D2D, MW2MW or AS2AS. Furthermore, a complete description of the deployments performed in the pilot scenarios will also be included, paying special attention to scalability and resilience to failures aspects, critical in industrial, logistics and health IoT scenarios.

7 References

[1] D. G. Gavin Mulligan, «What are the advantages/disadvantages of using REST API over native libraries,» 2003. [En línea]. Available: https://techbeacon.com/how-wrap-legacy-systems-rest-apis.

[2] V. G. K. Pammi, «Agile User Stories: The Building Blocks for Software Project Development Success,» Scrum Alliance, 20 September 2013. [En línea]. Available: https://www.scrumalliance.org/community/articles/2013/september/agile-user-stories. [Último acceso: 30 August 2017].

[3] Daubert, Jörg; Wiesmaier, Alexander and Kikiras, Panayotis;, «A view on privacy and trust in IoT».

[4] M. Rouse, «API Management,» 2016. [En línea]. Available: http://searchmicroservices.techtarget.com/definition/API-management. [Último acceso: June 2017].

[5] R. Johnson & Brian, «Designing Reusable Classes,» *Journal of Object-Oriented Programming,* vol. 1(2), pp. 22-35, 1988.

[6] M. W. S. R. E. R. Steve Danielson, «What is API Management?,» 23 01 2017. [En línea]. Available: https://docs.microsoft.com/en-us/azure/api-management/api-management-key-concepts. [Último acceso: June 2017].

[7] G. Psitakis, «API Management tools: How to find one for you,» 23 March 2015. [En línea]. Available: https://www.developereconomics.com/api-management-tools-how-to-find-the-one-for-you. [Último acceso: June 2017].

[8] Z. Flower, «A roundup of the leading API management tools,» [En línea]. Available: http://searchmicroservices.techtarget.com/feature/A-roundup-of-the-leading-API-management-tools-available-today. [Último acceso: June 2017].

[9] Mulesoft, Inc., «ProgrammableWeb,» [En línea]. Available: https://www.programmableweb.com. [Último acceso: 2017].

[10] A. Anthony, «Tracking the growth of the API Economy,» [En línea]. Available: https://nordicapis.com/tracking-the-growth-of-the-api-economy/. [Último acceso: September 2017].

[11] B. Doerrfield, «Ultimate Guide to 30+ API Documentation Solutions,» 30 November 2016. [En línea]. Available: https://nordicapis.com/ultimate-guide-to-30-api-documentation-solutions/. [Último acceso: August 2017].

[12] Y. Srikrishnan, «REST API Documentation - Part 1,» 29 December 2015. [En línea]. Available: https://dzone.com/articles/rest-api-documentation-part-1. [Último acceso: June 2017].

[13] Y. Srikrishnan, «REST API Documentation Part 2 — Swagger, RAML, and Open API,» 2 February 2016. [En línea]. Available: https://dzone.com/articles/rest-api-documentation-part-2-1. [Último acceso: June 2017].

[14] M. Stowe, «API Spec Comparison Tool,» 4 November 2016. [En línea]. Available: http://www.mikestowe.com/2014/12/api-spec-comparison-tool.php. [Último acceso: July 2017].

[15] S. Judd, «Documenting REST APIs – a tooling review,» 28 July 2015. [En línea]. Available: https://opencredo.com/rest-api-tooling-review/. [Último acceso: July 2017].

[16] K. Sandoval, «Top Specification Formats for REST APIs,» 19 March 2016. [En línea]. Available: https://nordicapis.com/top-specification-formats-for-rest-apis/. [Último acceso: June 2017].

[17] R. T. Fielding, «Architectural Styles and the Design of Network-based Software Architectures,» University of California, Irvine, CA, 2000.

[18] M. B. A. v. B. A. C. W. C. M. F. J. G. J. H. A. H. R. J. J. K. B. M. R. C. M. S. M. K. S. J. S. D. T. Kent Beck, «Manifesto for Agile Software Development,» 2001. [En línea]. Available: http://agilemanifesto.org/. [Último acceso: 2017].

[19] R. T. Fielding, «REST APIs must be hypertext-driven,» 28 October 2008. [En línea]. Available: http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven. [Último acceso: June 2017].

8 Annex I User requirements

Scalability. Computing resources [3]
INTER-FW should be scalable. Scalability relates to the ability of systems to seamlessly cater for higher demand in computing resources of data, devices, people and applications. The implemented mechanisms should ensure low communication overhead, ensuring fast decision making and high performance. Scalability has two approaches that must be targeted: - Allowing an increasing number of devices/platforms in a specific deployment. - Allowing a flexible deployment strategy to support from a small number of devices/platforms to a large number of them.
Acceptance criteria: A scalable system with 500 devices sending real-time data from one or more platforms should be managed by INTER-FW.
MoSCoW priority: Should have

Extensibility [10]
INTER-FW should be extensible with new functionalities. The system should expose functionality to the infrastructure maintenance to update the functionality when needed with new INTER-FW versions, without affecting existing clients.
Acceptance criteria: INTER-FW must be able to incorporate new functionalities without affecting an existing client that uses the previous functionality.
MoSCoW priority: Should have

Platform independency [14]
INTER-IoT must be platform independent. The INTER-IoT products must be able to connect or communicate to any platform by means of a common procedure which can be supported by any platform.
Acceptance criteria: At least Linux, Windows, IOs and Android must run as platforms or devices.
MoSCoW priority: Must have

Remote device control [26]
INTER-LAYER could be able to control devices remotely.

Manufacturer apps must be supported and allowed to take control over their devices.
Acceptance criteria: A manufacturer app remains functional once the system is connected to INTER-IoT. Compatible devices must be able to be connected and controlled together.
MoSCoW priority: Could have

System security [27]
INTER-IoT must have system security. Certain sections or sensors/actuators must be securely shielded from the public. It is obtained by relevant control functions such as access and transport resource control functions: authentication, authorization and accounting (AAA).
Acceptance criteria: Devices cannot be accessible by everyone. Access must be controllable for sensors. The network layer must provide reliable and secure connectivity as required by the pilots. Secured environments can be created for designated users, any sensitive data communication must be secure. Security levels do not allow third parties to take over control of a private system that is working over the IoT.
MoSCoW priority: Must have

System privacy [28]
INTER-IoT must provide privacy protection. Provide privacy protection for accessing information about physical entities, services or platforms connected to or integrated into INTER-IoT. To maintain this privacy, third party access to the private data or into the system is not possible. Also, the identifier or other critical information of a device (e.g., ID of an RFID tag or MAC address of Wireless Sensor) must not be tracked by unauthorized entities. Additionally, the avoidance of the integration and interaction of false nodes/sensors, or unauthorized smart objects must be ensured. Optional cryptographic mechanisms for information could be conveniently used.
Acceptance criteria: INTER-IoT provides higher security in device integration and system-device interaction. Security levels do not allow third parties to take over control of a private system that is working over the IoT. The IDs are only sent (and maybe stored) to/in other authorized entities, typically in the same subsystem, without any tracking purposes.
MoSCoW priority: Must have

Heterogeneous information representation [42]
INTER-FW must provide information from different sources.

The method of integration of multiple information and knowledge representing the same real-world sensing object into a consistent, accurate, and useful representation. It helps to fully take the usage of the IoT information resources for different application and service within an IoT system or between different information systems.
Acceptance criteria: Different information representation at different actors of the pilots is able to share information consistently. A cross-pilot with different information semantics and representation demonstrates the correct implementation of this requirement.
MoSCoW priority: Must have

API for third-party developers [47]
INTER-FW must provide an API for third-party developers. INTER-FW shall expose a public API for third-party developers and any potential INTER-IoT user. This API is the only way to access the functionality offered by all the components in the system.
Acceptance criteria: API resources can be accessed from external parties (different from the project consortium or the Open Call participants).
MoSCoW priority: Must have

API for data publication [51]
INTER-FW must provide an API for data publication. To have access at the application & services layers and at the semantic level from external IoT platforms. The API should be used for publishing sensor data into an existing IoT platform through an API without knowing anything about protocols.
Acceptance criteria: To have an API available for publishing sensor data into an existing IoT platform, specifically into a previously created sensor registered into the platform.
MoSCoW priority: Must have

API REST [52]
INTER-FW must provide an Rest API. The public API of the framework follows the RESTful design principles.
Acceptance criteria: To have a REST API implemented and documented in a swagger-like tool.
MoSCoW priority: Must have

Auditability and Accountability [58]
INTER-FW could be auditable and accountable.

Configured operations performed in the system must be tracked uniquely to the entity that generated it. The platform should allow: - To retrieve users and/or devices that carried out or are in charge of the activities in the system and their logged operations. - Producing an Audit log with trace of the most important data and their values before and after changes; - Maintain records for a period not less than six months; - Provide synchronization technologies in order to keep aligned the date and time recorded in the logs associated with the access. The criteria for registration of the aforesaid Log (products so as to not be editable) must at least enable the identification: - The event that triggered the log (login, logout, login failure); - The user, the date and the start / end connection. - The sensitive data updates (before and after).
Acceptance criteria: Allow auditability and accountability anytime, to ensure security operational aspects.
MoSCoW priority: Could have

Constraints on processing of personal data [61]
INTER-IoT must comply with the constraints to process personal data. In the case of processing of personal data, the processing must conform to the rules and criteria laid down in Directives 95/46/EC and 2002/58/EC and Nationals regulations. These conditions fall into three categories: • Transparency (the natural person has the right to be informed when his personal data is being processed). What we are required to provide, the purpose for which data is treated, all the information that characterizes the treatment of his data and the rights they enjoy and need to collect its approval in this regard. • Legitimate purpose (personal data can only be processed for specified explicit and legitimate purposes and may not be processed further in a way incompatible with those purposes). • Proportionality (personal data may be processed only insofar as it is adequate, relevant and not excessive in relation to the purposes for which they are collected and/or further processed; when sensitive personal data such as religious beliefs, political opinions, health, sexual orientation, race, membership of past organisations are being processed, extra restrictions apply.
Acceptance criteria: Presence of requirements and functions to ensure the rules and criteria of EC and Nationals regulations.
MoSCoW priority: Must have

Constraints on processing of personal and health data [62] (including 143, 145, 146, 152)

INTER-Health should conform to personal and health data processing rules.

The processing of Personal and Health data must conform to the rules and criteria laid down in “Personal Data Protection Code - Legislat. Decree no.196 of 30 June 2003” and in “Measures and arrangements applying to the controllers of processing operations performed with the help of electronic tools -27 November 2008”.

For Italian privacy regulation some specific instances are required:

Informed Consent: the person in charge of data processing must collect the informed consent of the involved person to the processing of data disclosing health status (for more details see requirement 145).

Information sheet: Art. 13 of the Privacy Codex provides that the people in charge of data processing or collecting need to be informed orally or in writing about the processing of data (for more details see requirement 146).

Privacy Codex: The behaviour of healthcare operators must be respectful of the right of personal dignity and confidentiality of every citizen and it has to be appropriate to various situations in which benefits are provided according to the Legislative Decree 196/2003 (for more details see requirement 143).

For United Kingdom regulation some specific instances are required:

Compliance with the Data protection act: Compliance with the data protection act is required in all cases of personal data collection, usage and storage. Sensitive data (such as health data) require a higher level of protection (for more details see requirement 151).

Information Security and Information Governance good practice guidelines: The health and social care information center implements good practice guidelines for use within the NHS. Adoption of any system by NHS funded organizations requires compliance with best practice guidelines for information security and governance (for more details see requirement 152).

Acceptant criteria:

Specific functions to ensure the rules and criteria of the Nationals regulations for the Personal Data Protection (EU Regulation 2016/679 and EU Directive 2016/680).

MoSCoW priority: Must have

Provision of authentication credentials [63]

INTER-FW must provide an authentication access with credentials.

As regards the management of the User credentials, the platform has to:

- allow access only through individual authentication credentials (consisting of a User ID and an authentication device, e.g. Password);

- prevent the reassignment of User ID to another user;

- allow the definition of access profiles sets that guarantee the principles of "need to know" and “segregation of duties”;

- allow the extraction of the information required to verify the correct allocation of authentication credentials and their authorization profiles;

- carry out automatic checks at least monthly of the users inactive for more than six months in order to suspend, unless the users for which it has been required and authorized a derogation on the basis of an operational need.

Acceptance criteria:

Have a secure and efficient authentication system.

MoSCoW priority: Must have

Confidentiality [69]

INTER-FW access should be confidential.

Avoid data falsification or disclosure.

- If the need of data processing ended, such data must be deleted permanently and irreversibly in order to prevent unauthorized treatment.

- It must be guaranteed the logical isolation of data belonging to different customers on a single platform. In particular, it must be guaranteed the segregation of single customer views, in order to allow processing of data only to persons in charge of the processing (preventing access / views by unauthorized persons).

- Special procedures for extraction and transmission of the data processed by the platform must be available.

- In order to ensure the confidentiality of data stored in the platform encryption must be provide of identification codes or other solutions that make health data unintelligible to those who are authorized to access (i.e. identification data decoupled from health / sensitive ones).

Acceptance criteria:

The different actors can access the objects for which access was granted, and not to those it hasn't. Besides the normal positive accessibility tests, tests including negative ones, particularly with colliding device names and other potential issues.

MoSCoW priority: Should have

Easy-to-use user interface [70]

INTER-Health could provide different interface features for different category of end users.

All end user interfaces should be easy to use.

For older people the confidence with the use of smart phones, Internet applications and information technology in general is very limited.

For younger people the confidence on the use of smartphones Internet technologies and applications increases.

As services are offered to both types of users, this requirement needs to be considered as a reference target for the design of web and mobile interfaces

It is assumed that health care providers (doctors, nurses, technicians) have experience with Internet applications. Access to the management, monitoring and consultation could be done through a web interface.

Acceptant criteria:

All the offered features should be easy to use by each category of end users.

MoSCoW priority: Could have

Application response time [71]

INTER-Health must guarantee adequate response times.

The resulting integrated platform should guarantee the same performances of the original platforms in terms of response time.
Acceptant criteria: The "navigation" functionalities on different contents by using either Smartphone or Personal Computer to access to the platform, have to guarantee a response time of a few seconds.
MoSCoW priority: Must have

Users manage how their public data is seen [77]
INTER-FW should allow users to select which of his public data can be seen. Devices/IoT platforms as data sources are owned by different third parties. The owner of the object should be able to manage who and when other users have access to their information. IoT platforms should support data ownership management, data-flow monitoring, and access management. Data visibility is managed according to owning entities policies. This is managed globally (platform independent) At the configuration of an IoT platform registered into INTER-IoT, the software integrator may be able to specify a list of devices and/or operations which are accessible from external agents through INTER-IoT, how long, with whom, etc.
Acceptance criteria: The user has tools to manage the access to their devices and platforms capabilities.
MoSCoW priority: Should have

API for proprietary systems interoperate with other systems [86]
INTER-FW should have an API for interoperability with proprietary systems. An upgrade or import-export process is necessary to integrate proprietary systems with the new systems. Could be done with an API from platform to platform or with a firmware upgrade of the devices.
Acceptance criteria: API that allow interoperability among platforms.
MoSCoW priority: Should have

Trust management [100]
INTER-IoT could support trust issues. Trust issues are related to the scenario in which devices and platforms cooperate without previous collaboration history. Trust management enables to make sure that the shared data are real and trustworthy, especially with crowdsourced and user generated data.
Acceptance criteria: Solution proposes mechanism to achieve trust management.
MoSCoW priority: Could have

Use of standards [123]
INTER-IoT should make use of standards. Solution should make use of established standards wherever suitable. The use of existing standards should be envisioned. The extension/adaptation of existing standards should be preferred for the definition of a proprietary solution.
Acceptance criteria: Existing standards are used when possible.
MoSCoW priority: Should have

Adaptability [125]
INTER-IoT could be adaptable to changing requirements. INTER-IoT must be adaptable to changing requirements on availability, performance and throughput. It must be suited to be continuously improved with respect to changing user behaviour. Usage processes, service usage patterns and user feedback should be tracked continuously in order to detect architectural weaknesses, inappropriate design decisions and inappropriate technologies.
Acceptance criteria: INTER-IoT provides data for the continuous evaluation of usage patterns and their impact on the running solution.
MoSCoW priority: Could have

Portability [132]
INTER-FW should be portable to different systems. Service providers must be able to switch between customers / users.
Acceptance criteria: Different systems recognize the same user.
MoSCoW priority: Should have

QoS Integration [142]
INTER-FW could provide QoS.

When appropriate, the IoT architecture should use the Quality of Service (QoS) features supported by underlying networks. This allows for all underlying networks to offer the same level of quality thus enhancing homogeneity.
Acceptance criteria: Identify QoS of all underlying networks and integrate all corresponding QoS features to the entire system. A checklist of all the QoS features of the system should be drawn.
MoSCoW priority: Could have

Design of required ontologies [186]
INTER-FW must use a common ontology. Use of required ontologies - a generic ontology of the Internet of Things. Creation of GOIoTP, a global IoT ontology, providing common understanding of the IoT (generic) meta-structure, and enabling semantic interoperability. It is required to be designed or chosen from available ones in order to produce semantic alignment. GOIoTP is based on current main IoT ontologies, such as W3C SSN, SAREF, etc.
Acceptance criteria: Within INTER-IoT at least generic ontology of the Internet of Things is available. INTER-LAYER must offer semantic interoperability through the creation and use of a Central Ontology.
MoSCoW priority: Must have

Integration with legacy systems [188]
INTER-IoT must integrate with legacy systems. Interfaces toward existing systems must be developed. In order to allow interaction between new and old developments. It is necessary connect legacy systems with new services through standard based protocol gateways to free data from proprietary constraints.
Acceptant criteria: When legacy systems older than 2010 are used, these systems must be able to communicate with the new INTER-IoT gateways and the rest of the system.
MoSCoW priority: Must have

IDEs and APIs for rapid new applications development [199]

INTER-IoT must be supported by APIs and IDEs. The port environment is constantly changing and therefore the need to create new applications. The products created must have APIs and IDEs that allow easily create new applications. They also have to be well documented to facilitate an understanding.
Acceptance criteria: Have simple APIs and IDEs to develop services with complete documentation.
MoSCoW priority: Must have

API for network services [226]
INTER-LAYER must provide an API for network services. The network layer should provide an API to guarantee that the access to the gateways and devices could be accomplished in a transparent manner from the upper layers point of view.
Acceptance criteria: INTER-LAYER must provide the specific available API for the access.
MoSCoW priority: Must have

API Middleware for interoperability between different platforms [237]
Middleware must provide an API for interoperability between platforms. An exported API by the middleware provides access to the different devices of the platforms connected to Inter-Layer.
Acceptance criteria: The correct access and utilization of the functionalities provided by the middleware layer, exposed through and public API.
MoSCoW priority: Must have

Gateway access API [243]
INTER-LAYER must provide an API for accessing the gateway. The API is obligatory in order to retrieve virtual object/device data, control of actuators, etc. from another layer.
Acceptance criteria: All exposed functions of the gateway are reachable from the API.
MoSCoW priority: Must have

Each data unit is identified univocally [254]
INTER-FW must have unique data units.

Each minimal unit of meaningful data transmission (e.g. a heart rate measurement or a truck location event) must contain an identifier allowing retrieve the source of data and the network/platform for traceability.
Acceptance criteria: Data transmissions have an identifier for traceability.
MoSCoW priority: Must have

Each sensor has a unique INTER-IoT identifier [256]
INTER-FW devices must have a unique identifier. An identifier system must be developed to be able to identify each device. Granularity in identification must reach the device level. Device-level services are provided at INTER-FW level [22] [33] [34] [35].
Acceptance criteria: Each device can be uniquely identified. Furthermore, there should not be a limitation to the number of devices that can be connected due to the lack of identifiers. The amount of identifier codes must be sufficiently large to accompany all current and future devices.
MoSCoW priority: Must have

The INTER-IoT unique ID is used to find the platform-specific ID of the device [257]
INTER-FW should find a platform-specific device ID from the INTER-IoT unique ID. The platform specific ID of each element needs to be retrieved from a unique ID assigned in INTER-IoT. This ensures traceability.
Acceptance criteria: A platform-specific ID can be retrieved from an INTER-IoT ID.
MoSCoW priority: Should have

Provides a sensor-level interface [259]
INTER-FW must provide functions for interoperability of devices. Device interoperability can be managed by the INTER-FW with tools/APIs to do it. These APIs enable to access single devices and perform the different operations about it.
Acceptance criteria: The device interoperability functions are available at INTER-FW level.

MoSCoW priority: Must have

Manage user permission [260]
INTER-FW must manage user permission. User permissions are managed at framework level.
Acceptance criteria: There is a field for user permission in the user configuration.
MoSCoW priority: Must have

A user knows its permissions [261]
INTER-FW could allow a user to know its permissions. The permission to access system resources are known by the user at INTER-FW level.
Acceptance criteria: There is a method to ask the framework about the permissions of the current user.
MoSCoW priority: Could have

Manages group-based permissions [262]
INTER-FW could manage permissions for groups. Permissions can be managed at group level in order to simplify business processes.
Acceptance criteria: Users can be managed in groups.
MoSCoW priority: Could have

Access to personal data needs to be previously authorized [263]
INTER-FW must check permission for accessing to personal data. Access to personal information must be previously authorized by the owner.
Acceptance criteria: System does not allow access to personal data without previous permission of the owner.
MoSCoW priority: Must have

API allows create/update/remove users [264]
INTER-FW must provide an API to create/update/remove users. API to allow applications to implement user management tools.
Acceptance criteria: Allow methods for CRUD users by using the API.
MoSCoW priority: Must have

API allows device declaration and configuration [265]
INTER-FW must provide an API to declare and configure devices. API to allow applications to implement device management tools.
Acceptance criteria: Allow methods for CRUD devices by using the API.
MoSCoW priority: Must have

API allows resources/capabilities discovery [266]
INTER-FW must provide an API to discovery resources. API allow applications to discover resources and capabilities of the platforms.
Acceptance criteria: Allow methods for discover devices, platforms and capabilities of them.
MoSCoW priority: Must have

API allows historic data query [268]
INTER-FW should provide an API to query historic data. API allows to query historic data for reports, charts, etc.
Acceptance criteria: Allow methods for database-like querying.
MoSCoW priority: Should have

API allows subscription to data streams/queues [270]
INTER-FW must provide an API to subscribe to data streams. If data streams or queues are available, third parties can discover and connect to them to develop their own applications.
Acceptance criteria: Allow methods for subscription to synchronous data sources.
MoSCoW priority: Must have

Stores recent data for recovery [272]
INTER-FW should store recent data for recovery. A copy of the recent data shared among platforms is stored in a given time window.
Acceptance criteria:

Shared data must be able to be recovered after a connectivity failure.
MoSCoW priority: Should have

Stores system status for recovery [273]
INTER-FW should store system status for recovery. The system state and configuration is persisted allowing to recover it after an unexpected failure.
Acceptance criteria: Status data must be able to be recovered after a platform failure.
MoSCoW priority: Should have

Time triggers [276]
INTER-FW should support time triggers for improving the communications. It must be possible to provision timers in the protocol stack to indicate when to re-send a packet that has not been confirmed. These timers should be individually provision according to the destination address in the packet.
Acceptance criteria: Time triggers are implemented and working in the prototype, as validated by the pilots. A mock-up component can be set to not return ACKs and test the functionality.
MoSCoW priority: Should have

Indivisibility [277]
INTER-FW could transfer a sequence of packets as a logical unit. The protocol must have a means to transfer a sequence of packets as a logical unit, like a firmware upgrade, a data log, or provisioning information.
Acceptance criteria: Features are implemented and working in the prototype, as validated by the pilots.
MoSCoW priority: Could have

Future-proof [278]
INTER-FW should work with previous versions. Future-proof: Future versions of the protocol must work with prior versions and provide all the same capabilities as prior versions.
Acceptance criteria: Features are implemented and working in the prototype, as validated by the pilots. In particular, the first and second release of the implementations should be backwards compatible, as old use-cases are tested in the new test-bed.
MoSCoW priority: Should have

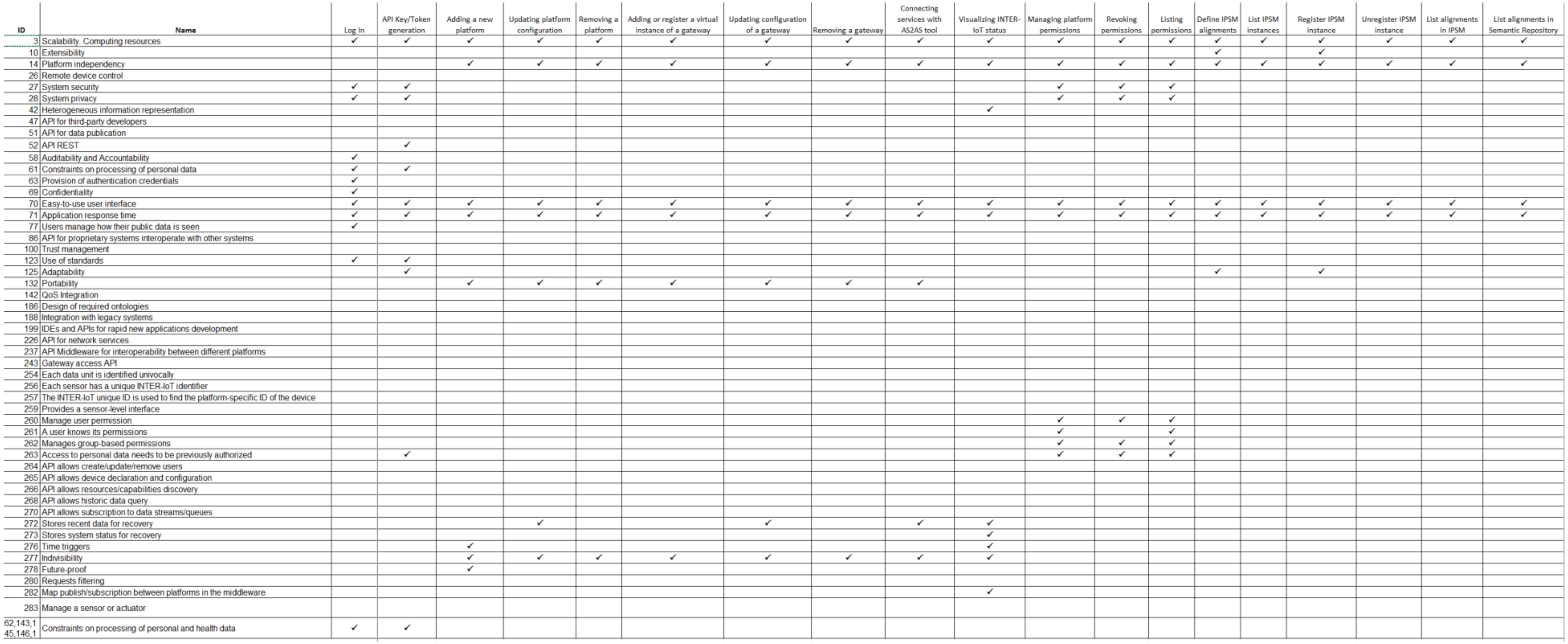
Requests filtering [280]
INTER-FW must provide Requests filtering in the API. When sending a requests to INTER-FW, it is possible to specify filtering: The system shares a common filter format when possible. This filtering allows: - Selection of platform(s). - Selection of device(s). - Selection of property type(s). - Selection of property filtering(s). - Selection of geo-queries (if allowed by the IoT platform).
Acceptance criteria: All the previous filtering types are accessible in the API and performed to the underlying connected IoT platforms.
MoSCoW priority: Must have

Map publish/subscription between platforms in the middleware [282]
Middleware could provide publication and subscription between platforms. The user can configure to map publish/subscription between platforms, by performing a specific subscription from a source IoT platform and mapping it to be directly published in the target IoT platform.
Acceptance criteria: An API allows to set up a subscription from a source IoT platform to the target IoT platform.
MoSCoW priority: Could have

Manage a sensor or actuator [283]
INTER-LAYER should be able to manage actuators. In addition to receiving information from sensors and actuators, it is necessary to be able to send configuration changes or specific actions.

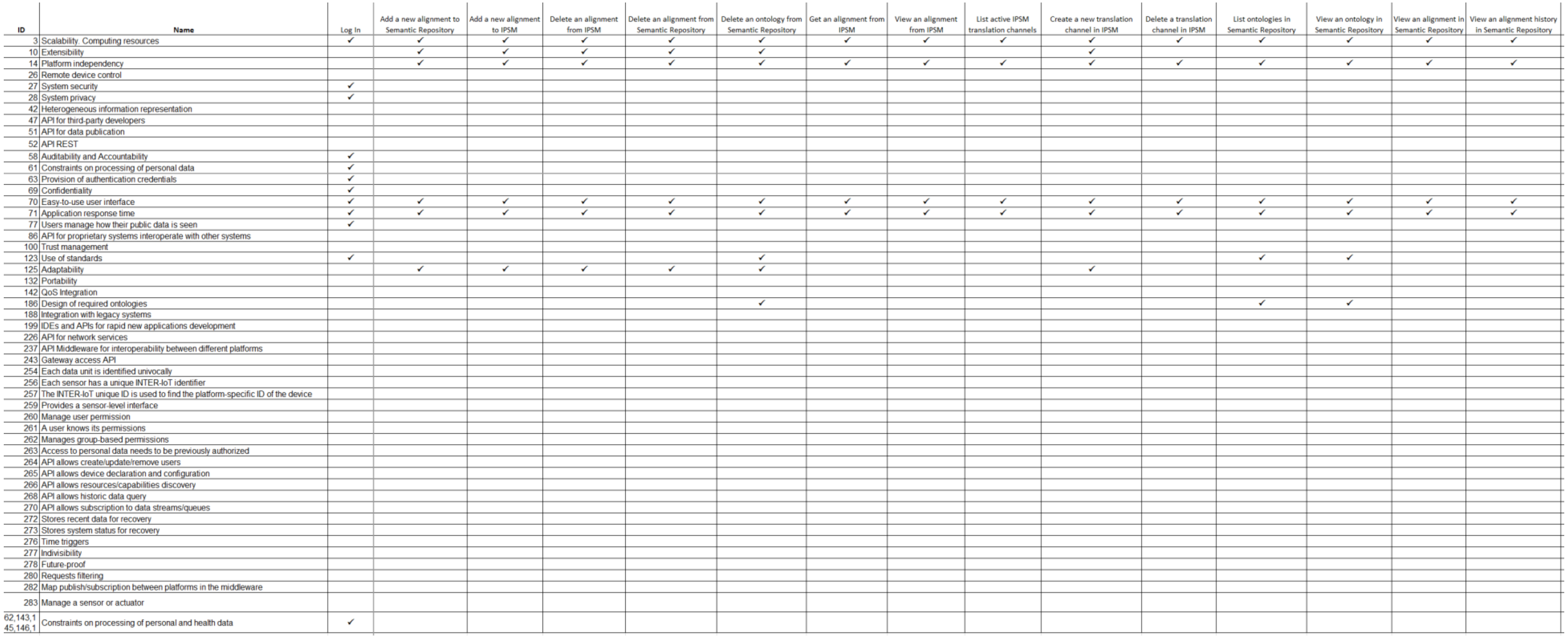
Acceptance criteria: Development of a method to send an action to a specific actuator.
MoSCoW priority: Should have

9 Annex II Requirements – use cases mapping



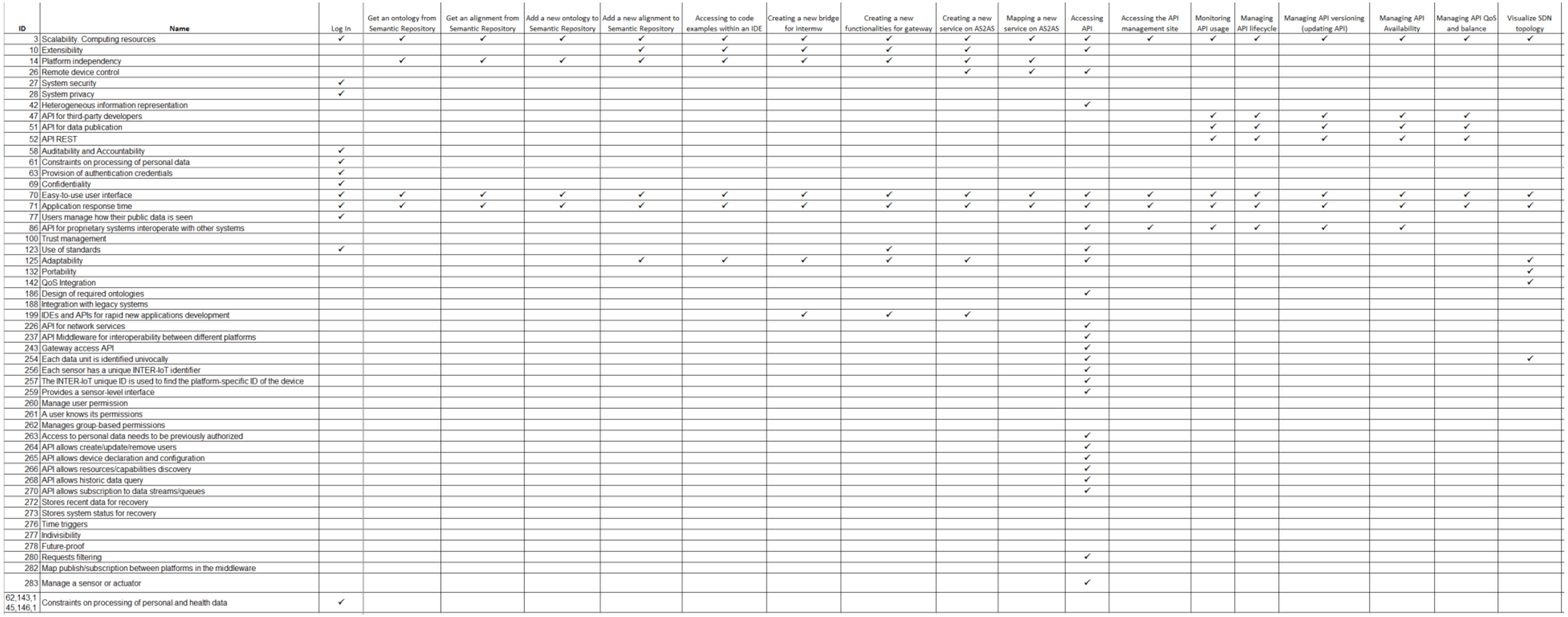
ID	Name	Log In	API Key/Token generation	Adding a new platform	Updating platform configuration	Removing a platform	Adding or register a virtual instance of a gateway	Updating configuration of a gateway	Removing a gateway	Connecting services with AS2AS tool	Visualizing INTER-IoT status	Managing platform permissions	Revoking permissions	Listing permissions	Define IPSM alignments	List IPSM instances	Register IPSM instance	Unregister IPSM instance	List alignments in IPSM	List alignments in Semantic Repository
3	Scalability. Computing resources	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
10	Extensibility														✓		✓			
14	Platform independency			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
26	Remote device control																			
27	System security	✓	✓									✓	✓	✓						
28	System privacy	✓	✓									✓	✓	✓						
42	Heterogeneous information representation										✓									
47	API for third-party developers																			
51	API for data publication																			
52	API REST		✓																	
58	Auditability and Accountability	✓																		
61	Constraints on processing of personal data	✓	✓																	
63	Provision of authentication credentials	✓																		
69	Confidentiality	✓																		
70	Easy-to-use user interface	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
71	Application response time	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
77	Users manage how their public data is seen	✓																		
86	API for proprietary systems interoperate with other systems																			
100	Trust management																			
123	Use of standards	✓	✓																	
125	Adaptability		✓												✓		✓			
132	Portability			✓	✓	✓	✓	✓	✓	✓										
142	QoS Integration																			
186	Design of required ontologies																			
188	Integration with legacy systems																			
199	IDEs and APIs for rapid new applications development																			
226	API for network services																			
237	API Middleware for interoperability between different platforms																			
243	Gateway access API																			
254	Each data unit is identified univocally																			
256	Each sensor has a unique INTER-IoT identifier																			
257	The INTER-IoT unique ID is used to find the platform-specific ID of the device																			
259	Provides a sensor-level interface																			
260	Manage user permission											✓	✓	✓						
261	A user knows its permissions											✓		✓						
262	Manages group-based permissions											✓	✓	✓						
263	Access to personal data needs to be previously authorized		✓									✓	✓	✓						
264	API allows create/update/remove users																			
265	API allows device declaration and configuration																			
266	API allows resources/capabilities discovery																			
268	API allows historic data query																			
270	API allows subscription to data streams/queues																			
272	Stores recent data for recovery				✓			✓		✓	✓									
273	Stores system status for recovery										✓									
276	Time triggers			✓							✓									
277	Indivisibility			✓	✓	✓	✓	✓	✓	✓	✓									
278	Future-proof			✓																
280	Requests filtering																			
282	Map publish/subscription between platforms in the middleware										✓									
283	Manage a sensor or actuator																			
62,143,1 45,146,1	Constraints on processing of personal and health data	✓	✓																	

Table 22 Requirements - use cases mapping part I



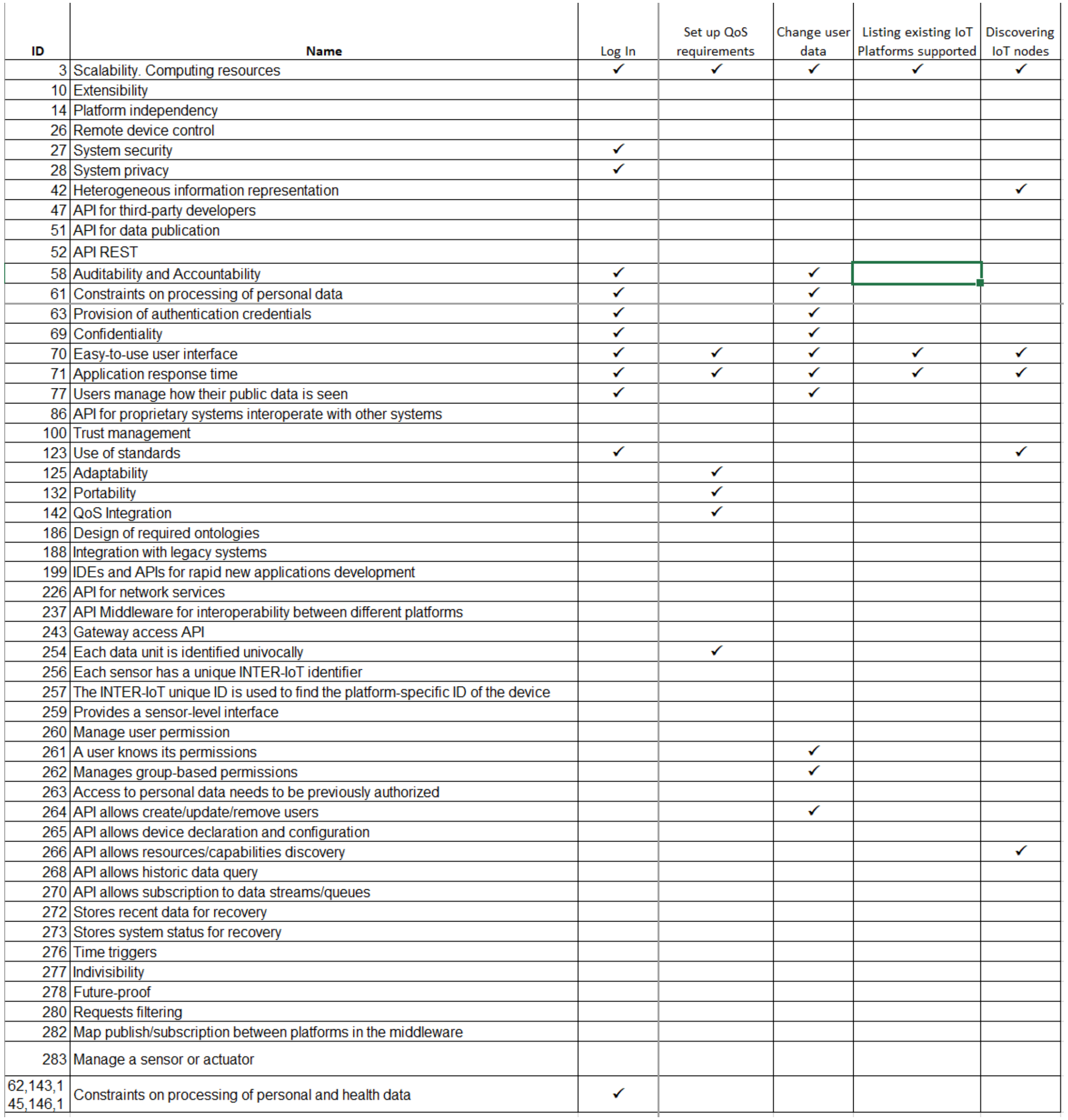
ID	Name	Log In	Add a new alignment to Semantic Repository	Add a new alignment to IPSM	Delete an alignment from IPSM	Delete an alignment from Semantic Repository	Delete an ontology from Semantic Repository	Get an alignment from IPSM	View an alignment from IPSM	List active IPSM translation channels	Create a new translation channel in IPSM	Delete a translation channel in IPSM	List ontologies in Semantic Repository	View an ontology in Semantic Repository	View an alignment in Semantic Repository	View an alignment history in Semantic Repository
3	Scalability. Computing resources	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
10	Extensibility		✓	✓	✓	✓	✓				✓					
14	Platform independency		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
26	Remote device control															
27	System security	✓														
28	System privacy	✓														
42	Heterogeneous information representation															
47	API for third-party developers															
51	API for data publication															
52	API REST															
58	Auditability and Accountability	✓														
61	Constraints on processing of personal data	✓														
63	Provision of authentication credentials	✓														
69	Confidentiality	✓														
70	Easy-to-use user interface	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
71	Application response time	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
77	Users manage how their public data is seen	✓														
86	API for proprietary systems interoperate with other systems															
100	Trust management															
123	Use of standards	✓					✓						✓	✓		
125	Adaptability		✓	✓	✓	✓	✓				✓					
132	Portability															
142	QoS Integration															
186	Design of required ontologies						✓						✓	✓		
188	Integration with legacy systems															
199	IDEs and APIs for rapid new applications development															
226	API for network services															
237	API Middleware for interoperability between different platforms															
243	Gateway access API															
254	Each data unit is identified univocally															
256	Each sensor has a unique INTER-IoT identifier															
257	The INTER-IoT unique ID is used to find the platform-specific ID of the device															
259	Provides a sensor-level interface															
260	Manage user permission															
261	A user knows its permissions															
262	Manages group-based permissions															
263	Access to personal data needs to be previously authorized															
264	API allows create/update/remove users															
265	API allows device declaration and configuration															
266	API allows resources/capabilities discovery															
268	API allows historic data query															
270	API allows subscription to data streams/queues															
272	Stores recent data for recovery															
273	Stores system status for recovery															
276	Time triggers															
277	Indivisibility															
278	Future-proof															
280	Requests filtering															
282	Map publish/subscription between platforms in the middleware															
283	Manage a sensor or actuator															
62,143,1 45,146,1	Constraints on processing of personal and health data	✓														

Table 23 Requirements - use cases mapping part II



ID	Name	Log In	Get an ontology from Semantic Repository	Get an alignment from Semantic Repository	Add a new ontology to Semantic Repository	Add a new alignment to Semantic Repository	Accessing to code examples within an IDE	Creating a new bridge for intermw	Creating a new functionalities for gateway	Creating a new service on AS2AS	Mapping a new service on AS2AS	Accessing API	Accessing the API management site	Monitoring API usage	Managing API lifecycle	Managing API versioning (updating API)	Managing API Availability	Managing API QoS and balance	Visualize SDN topology
3	Scalability. Computing resources	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
10	Extensibility					✓	✓	✓	✓	✓		✓							
14	Platform independency		✓	✓	✓	✓	✓	✓	✓	✓	✓								
26	Remote device control									✓	✓	✓							
27	System security	✓																	
28	System privacy	✓																	
42	Heterogeneous information representation											✓							
47	API for third-party developers													✓	✓	✓	✓	✓	
51	API for data publication													✓	✓	✓	✓	✓	
52	API REST													✓	✓	✓	✓	✓	
58	Auditability and Accountability	✓																	
61	Constraints on processing of personal data	✓																	
63	Provision of authentication credentials	✓																	
69	Confidentiality	✓																	
70	Easy-to-use user interface	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
71	Application response time	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
77	Users manage how their public data is seen	✓																	
86	API for proprietary systems interoperate with other systems											✓	✓	✓	✓	✓	✓		
100	Trust management																		
123	Use of standards	✓							✓			✓							
125	Adaptability					✓	✓	✓	✓	✓		✓							✓
132	Portability																		✓
142	QoS Integration																		✓
186	Design of required ontologies											✓							
188	Integration with legacy systems																		
199	IDEs and APIs for rapid new applications development							✓	✓	✓									
226	API for network services											✓							
237	API Middleware for interoperability between different platforms											✓							
243	Gateway access API											✓							
254	Each data unit is identified univocally											✓							✓
256	Each sensor has a unique INTER-IoT identifier											✓							
257	The INTER-IoT unique ID is used to find the platform-specific ID of the device											✓							
259	Provides a sensor-level interface											✓							
260	Manage user permission																		
261	A user knows its permissions																		
262	Manages group-based permissions																		
263	Access to personal data needs to be previously authorized											✓							
264	API allows create/update/remove users											✓							
265	API allows device declaration and configuration											✓							
266	API allows resources/capabilities discovery											✓							
268	API allows historic data query											✓							
270	API allows subscription to data streams/queues											✓							
272	Stores recent data for recovery																		
273	Stores system status for recovery																		
276	Time triggers																		
277	Indivisibility																		
278	Future-proof																		
280	Requests filtering											✓							
282	Map publish/subscription between platforms in the middleware																		
283	Manage a sensor or actuator											✓							
62,143,1 45,146,1	Constraints on processing of personal and health data	✓																	

Table 24 Requirements - use cases mapping part III



ID	Name	Log In	Set up QoS requirements	Change user data	Listing existing IoT Platforms supported	Discovering IoT nodes
3	Scalability. Computing resources	✓	✓	✓	✓	✓
10	Extensibility					
14	Platform independency					
26	Remote device control					
27	System security	✓				
28	System privacy	✓				
42	Heterogeneous information representation					✓
47	API for third-party developers					
51	API for data publication					
52	API REST					
58	Auditability and Accountability	✓		✓		
61	Constraints on processing of personal data	✓		✓		
63	Provision of authentication credentials	✓		✓		
69	Confidentiality	✓		✓		
70	Easy-to-use user interface	✓	✓	✓	✓	✓
71	Application response time	✓	✓	✓	✓	✓
77	Users manage how their public data is seen	✓		✓		
86	API for proprietary systems interoperate with other systems					
100	Trust management					
123	Use of standards	✓				✓
125	Adaptability		✓			
132	Portability		✓			
142	QoS Integration		✓			
186	Design of required ontologies					
188	Integration with legacy systems					
199	IDEs and APIs for rapid new applications development					
226	API for network services					
237	API Middleware for interoperability between different platforms					
243	Gateway access API					
254	Each data unit is identified univocally		✓			
256	Each sensor has a unique INTER-IoT identifier					
257	The INTER-IoT unique ID is used to find the platform-specific ID of the device					
259	Provides a sensor-level interface					
260	Manage user permission					
261	A user knows its permissions			✓		
262	Manages group-based permissions			✓		
263	Access to personal data needs to be previously authorized					
264	API allows create/update/remove users			✓		
265	API allows device declaration and configuration					
266	API allows resources/capabilities discovery					✓
268	API allows historic data query					
270	API allows subscription to data streams/queues					
272	Stores recent data for recovery					
273	Stores system status for recovery					
276	Time triggers					
277	Indivisibility					
278	Future-proof					
280	Requests filtering					
282	Map publish/subscription between platforms in the middleware					
283	Manage a sensor or actuator					
62,143,1 45,146,1	Constraints on processing of personal and health data	✓				

Table 25 Requirements - use cases mapping part IV