SEMIoTICS

Deliverable D4.11
Semantic Interoperability Mechanisms for IoT (final)

Deliverable release date	30.04.2020
Authors	1. Eftychia Lakka, George Hatzivasilis, Nikolaos Petroulakis (FORTH), 2. Darko Anicic, Arne Broering (SAG) 3. Mirko Falchetto (ST) 4. Lukasz Ciechomski (BS) 5. Kostas Ramantas (IQU)
Responsible person	Eftychia Lakka (FORTH)
Reviewed by	Darko Anicic (SAG), Konstantinos Fysarakis (STS), Jordi Serra (CTTC), Mirko Falchetto (ST), Kostas Ramantas (IQU), Urszula Rak (BS), Nikolaos Petroulakis (FORTH)
Approved by	PTC Members (Vivek Kulkarni, Nikolaos Petroulakis, Ermin Sakic, Mirko Falchetto, Domenico Presenza, Verikoukis Christos) PCC Members (Vivek Kulkarni, Nikolaos Petroulakis, Verikoukis Christos, Georgios Spanoudakis, Domenico Presenza, Danilo Pau, Joachim Posegga, Darek Dober, Kostas Ramantas, Ulrich Hansen)
Status of the Document	Final
Version	1.0
Dissemination level	Public

Table of Contents

1 Introduction ... 6
1.1 PERT chart of SEMIoTICS ... 8
2 Motivation ... 9
2.1 Challenges for Semantic Interoperability ... 9
2.1.1 Overview ... 9
2.1.2 SEMIoTICS Challenges for Semantic Interoperability ... 10
2.2 Motivating Scenario – Smart Sensing ... 11
2.2.1 Horizontal Scenario – Field Layer ... 13
2.2.2 Vertical Scenario – Backend Layer ... 13
3 Background and Related Work ... 15
3.1 The Basis for Semantics ... 15
3.2 Semantic Models for Smart Objects ... 17
3.3 Ontologies ... 19
3.3.1 Overview ... 19
3.3.2 Ontologies in Healthcare Domain ... 19
4 SEMIoTICS Semantics ... 21
4.1 Data Mappings in SEMIoTICS ... 21
4.2 Ontologies in SEMIoTICS ... 21
4.3 TDs for all the Types of Smart Objects in SEMIoTICS ... 21
4.3.1 TDs of UC1 Sensors ... 21
4.3.2 TDs of UC2 Sensors ... 25
4.3.3 TDs Of UC3 Sensors ... 26
5 Semantic Mediator Mechanisms ... 30
5.1 Ontology Alignments & Data Transformation Techniques ... 30
5.1.1 Data Mapping ... 30
5.1.2 Ontology Alignment ... 30
5.1.3 Full set of Adaptor Nodes ... 33
5.2 Semantic Reasoning ... 36
6 SEMIoTICS Integration Approach with Other IoT Platforms ... 39
6.1 Introduction ... 39
6.2 Interaction with FIWARE ... 39
6.2.1 Overview ... 39
6.2.2 Scenario of Integration Between SEMIoTICS and FIWARE ... 44
6.2.3 FIWARE – SEMIoTICS Interoperability Layer Bridge ... 47

6.3 Interaction with MindSphere 48
6.4 Interaction with openHAB 51
7 End-To-end Interoperability Verification Mechanisms 54
7.1 Translation of Recipes into SPDI Patterns 54
7.2 Interoperability Patterns....58
7.2.1 Introduction 58
7.2.2 Semantic Interoperability Pattern Definition 59
7.2.3 Semantic Interoperability Pattern Specification Rule 59
8 Semantic Interoperability mechanisms - Implementation 61
8.1 Role of Backend Semantic Validator Component 61
8.2 Proof-of-concept Implementation 61
8.3 Final Implementation 64
8.3.1 Validation Mechanisms – Implementation 64
8.3.2 Connection with External IoT Platforms – Implementation....68
8.3.3 Interoperability Adaptation – Implementation 71
9 Semantic Interoperability mechanisms in the SEMIoTICS Use Cases 72
9.1 Use Case 1 72
9.2 Use Case 2 73
9.3 Use Case 3 74
10 Validation....77
10.1 Related Project Objectives and Key Performance Indicators (KPIs) 77
10.2 SEMIoTICS Interoperability Requirements....78
11 Conclusion....79
References....80

Acronyms Table	
Acronym	**Definition**
API	Application Programming Interface
BSV	Backend Semantic Validator
CEN	Committee European Normalization
CRUD	Create, Read, Update and Delete
DAWG	Data Access Working Group
EBNF	Extended Backus-Naur Form
ECA	Event-Condition-Action
EHRs	Electronic Health Records
EU	European Union
GEs	Generic Enablers
GLIF	Guidelines Interchange Format
GW	Gateway
GWSM	GW Semantic Mediator
HL7	Health Level 7
HTTP	Hypertext Transfer Protocol
IIoT	Industrial Internet of Things
IoT	Internet of Things
ISO	Organization for Standardization
JSON	JavaScript Object Notation
JSON-LD	JSON for Linking Data
KPI	Key Performance Indicator
LD	Linked Data
LOD	Linked Open Data
NGSI	Next Generation Service Interface
PEB	Pattern Engine - Backend
PO	Pattern Orchestrator
OWL	Web Ontology Language
P2P	Peer to Peer
QoS	Quality of Service
RC	Recipe Cooker
RDF	Resource Description Framework

RDFS	RDF Scheme
RE	Regular Expression
REST	Representational State Transfer
RPCs	Remote Procedure Calls
SAPB	Semantic API & Protocol Binding
SAREF	Smart Appliance REFerence
SDN	Software Defined Networking
SEMIoTICS	Smart End-to-end Massive IoT Interoperability, Connectivity and Security
SM	Semantic Mediator
SOSA	Sensor, Observation, Sample, and Actuator
SPDI	Security, Privacy, Dependability, and Interoperability
SSC	SDN controller of SEMIoTICS
SSN	Semantic Sensor Network
SWE	Sensor Web Enablement
TD	W3C Things Description
TDB	Thing Directory - Backend
UC	Use Case
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	eXtensible Markup Language
W3C	World Wide Web Consortium
WoT	Web of Things
WP	Work Package
WSDL	Web Services Description Language

1 INTRODUCTION

This deliverable is the final output of Task 4.4 and tackles the semantic interoperability issues that arise in the Internet of Things (IoT) domain [1]. Semantic interoperability is the designed property where various systems can interact with each other and exchange data with unambiguous, shared meaning. This enables knowledge discovery, machine computable reasoning and federation of different information systems.

Interoperability is materialized by including information regarding the data (metadata) and linking each element to a commonly agreed and shared vocabulary. Thus, the meaning of the data is exchanged along with the data itself in a self-describing information package. The shared vocabulary and the association to an ontology enable machine interoperation, logic, and inference. Ontology is an explicit specification of a conceptualisation and includes a formal representation of the properties and relations between the entities, concepts and data of a specific application domain.

In general, technologies from the Semantic Web are adapted in order to capture the inherited properties of the IoT domain. They are widely-used and well-studied eXtensible Markup Language (XML) schemes, like the Resource Description Framework (RDF), RDF Schema (RDFS), and Web Ontology Language (OWL) for ontologies, and the Web Services Description Language (WSDL) for services [2]. Such technologies offer common description and representation of data and services, characterize things and their capabilities, and deal with the semantic annotation, resource discovery, access management, and knowledge extraction in a machine-readable and interoperable manner.

More recently, World Wide Web Consortium (W3C) has launched a working group called Web of Things (WoT)[1] with the goal to counter the IoT fragmentation and enable interoperable IoT devices and services, thereby reducing the costs of their development. A notable feature of W3C WoT approach is Thing Description (TD)[2], used to describe the metadata and interfaces of (physical) Things in a machine interpretable format. TD has been built upon the W3C's extensive work on RDF, Linked Data (LD)[3] and JavaScript Object Notation (JSON) for Linking Data (JSON-LD)[4]. TD defines a domain agnostic vocabulary to describe any Thing in terms of its properties, events and actions. In order to give a semantic meaning to a set of properties, events and actions for a Thing, various semantic models can be used. One notable community effort to create a semantic schema for IoT applications is iot.schema.org[5]. Together, W3C WoT and iot.schema.org, provide a semantic interoperability layer that enables software to interact with the physical world. The interaction is abstracted in such a way that it simplifies the development of applications across diverse domains and IoT ecosystems. SEMIoTICS adopts this approach to establish its core semantics (see Chapter 4).

The common interpretation of semantic information in a globally shared ontology could be quite useful. However, this is not always the case. Although several local systems may utilize popular or standardized ontologies, eventually they extend them and establish their own semantics and interfaces. The direct interaction between these systems is not feasible. Thus, the use of semantic interoperability mechanisms is proposed in this deliverable (see Chapter 5), which correlate the required information and enable the interoperability of systems with different semantics or cross-domain interaction. The integration of SEMIoTICS with the other IoT Platforms including the FIWARE is also presented in Chapter 6.

Then, a common and generic Application Programming Interface (API) will be established in Task 4.6 between the different IoT middleware platforms. The API will ease the development of software services and applications for different platforms according to a well-defined architecture. Moreover, SEMIoTICS focuses on semantic interoperability in an attempt to establish interoperability patterns that will facilitate the modelling and

[1] https://www.w3.org/WoT/

[2] https://w3c.github.io/wot-thing-description/

[3] https://www.w3.org/standards/semanticweb/data

[4] https://www.w3.org/TR/2014/REC-json-ld-20140116/

[5] https://github.com/iot-schema-collab & http://iotschema.org/

real-time management of the underlying IoT ecosystem (see Deliverable 4.1 and its follow-up, D4.8) by incorporating the abovementioned mechanisms - i.e. semantics, Semantic Mediators (SMs) and common APIs. This will be based on the formal analysis of the RECIPE tool [3] and the five main interoperability settings suggested by the European Union (EU) funded project BigIoT [4] in order to address interoperability and compatibility issues for composing services from inter- to cross-domain topologies (see Chapter 7). In Chapter 8, the first and the final proposed implementation for of semantic interoperability mechanisms is presented. Chapter 9 provides specific scenarios of the use of semantic interoperability mechanisms in each of the SEMIoTICS use cases. Finally, in Chapter 10 the validation of the project objectives, key performance and interoperability mechanisms are also presented.

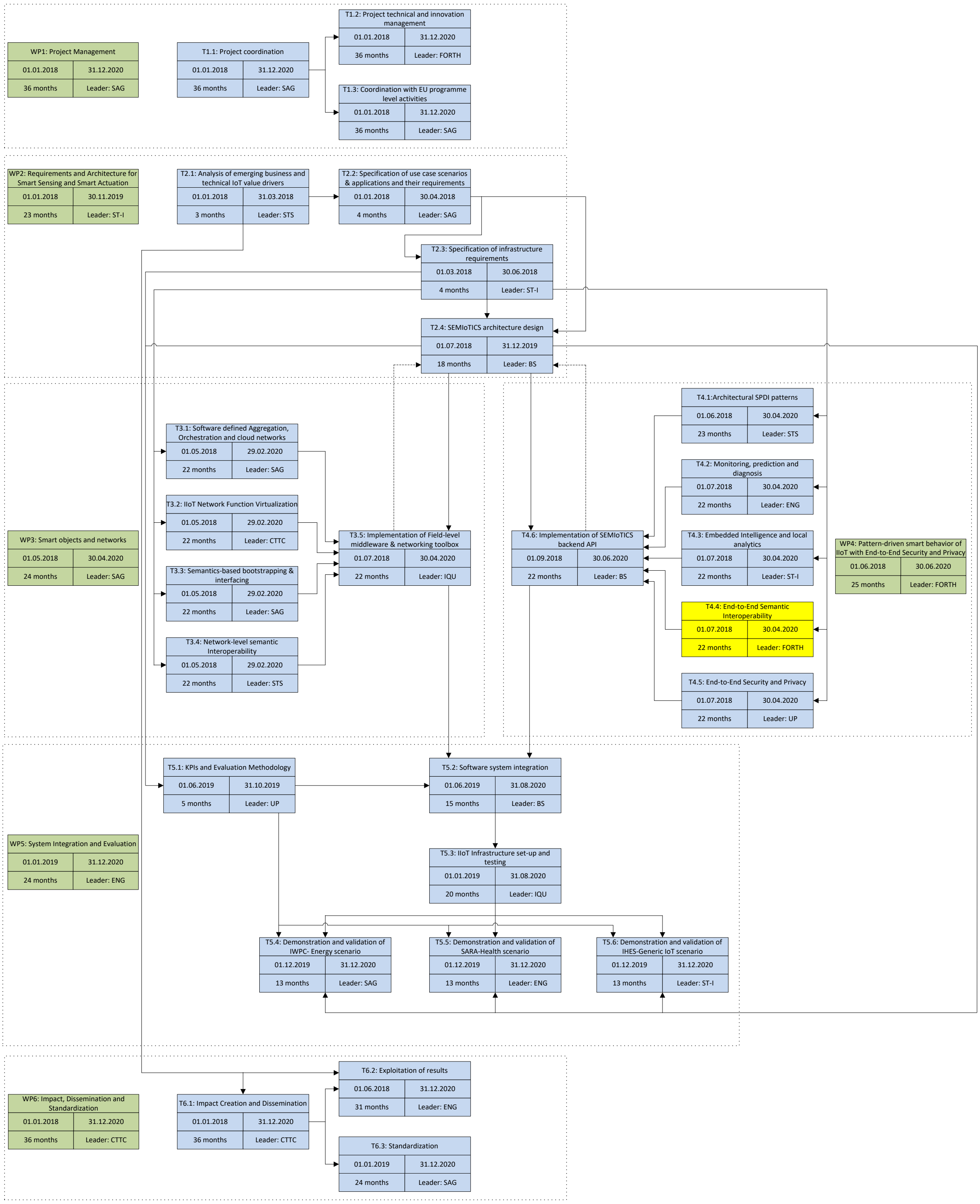
In this context and considering the delta to the previous version of the deliverable, i.e. D4.4 - "Semantic Interoperability Mechanisms for IoT (first draft)", the latest developments presented within this final Task 4.4 deliverable include:

- A brief overview of the TDs for the main types of smart objects in SEMIoTICS (see subsection 4.3)
- A full set of Adaptor Nodes in semantic validation mechanisms (see subsection 5.1.3)
- The final SEMIoTICS integration approach with other IoT platforms (FIWARE, MindSphere, openHAB) (see section 6)
- Final definition and specification of the semantic interoperability Patterns (see subsection 7.2)
- Analysis of the role of the Backend Semantic Validator - the core component of Task 4.4 (see subsection 8.1)
- Final implementation of the semantic validation mechanisms (see subsection 8.3)
- Details on the role of the semantic validation mechanisms in the SEMIoTICS UCs (see section 9)

More specifically, the rest of this deliverable is structured as follows:

- **Chapters 2 and 3** provide the motivation, background and related work for the interoperable solutions in the IoT domain.
- **Chapter 4** describes the datatype mapping approach that is adopted for SEMIoTICS, the semantic ontologies and the Thing Descriptions of the different types of smart objects in SEMIoTICS.
- **Chapter 5** defines the implementation of the concrete semantic interoperability mechanism and the structure of Adaptor Nodes that are developed for the SEMIoTICS requirements.
- **Chapter 6** provides the integration of the SEMIoTICS framework with other IoT external platforms (FIWARE, MindSphere, openHAB).
- **Chapter 7** details the verification mechanisms that ensure the end-to-end interoperability and the final definition of semantic interoperability Patterns.
- **Chapter 8** outlines the final development of semantic interoperability/validation mechanisms.
- **Chapter 9** presents specific scenarios of the use of semantic interoperability mechanisms in each of the SEMIoTICS Use Cases.
- **Chapter 10** refers to the initial validation approach (verification and guarantee) of the semantic interoperability features and the fulfilment of the SEMIoTICS's objectives and architectural requirements.
- Finally, **Chapter 11** provides the concluding remarks.

1.1 PERT chart of SEMIoTICS


WP1: Project Management
01.01.2018
31.12.2020
36 months
Leader: SAG
T1.1: Project coordination
01.01.2018
31.12.2020
36 months
Leader: SAG
T1.2: Project technical and innovation management
01.01.2018
31.12.2020
36 months
Leader: FORTH
T1.3: Coordination with EU programme level activities
01.01.2018
31.12.2020
36 months
Leader: SAG
WP2: Requirements and Architecture for Smart Sensing and Smart Actuation
01.01.2018
30.11.2019
23 months
Leader: ST-I
T2.1: Analysis of emerging business and technical IoT value drivers
01.01.2018
31.03.2018
3 months
Leader: STS
T2.2: Specification of use case scenarios & applications and their requirements
01.01.2018
30.04.2018
4 months
Leader: SAG
T2.3: Specification of infrastructure requirements
01.03.2018
30.06.2018
4 months
Leader: ST-I
T2.4: SEMIoTICS architecture design
01.07.2018
31.12.2019
18 months
Leader: BS
T3.1: Software defined Aggregation, Orchestration and cloud networks
01.05.2018
29.02.2020
22 months
Leader: SAG
T3.2: IIoT Network Function Virtualization
01.05.2018
29.02.2020
22 months
Leader: CTTC
T3.3: Semantics-based bootstrapping & interfacing
01.05.2018
29.02.2020
22 months
Leader: SAG
T3.4: Network-level semantic Interoperability
01.05.2018
29.02.2020
22 months
Leader: STS
WP3: Smart objects and networks
01.05.2018
30.04.2020
24 months
Leader: SAG
T3.5: Implementation of Field-level middleware & networking toolbox
01.07.2018
30.04.2020
22 months
Leader: IQU
T4.6: Implementation of SEMIoTICS backend API
01.09.2018
30.06.2020
22 months
Leader: BS
T4.1:Architectural SPDI patterns
01.06.2018
30.04.2020
23 months
Leader: STS
T4.2: Monitoring, prediction and diagnosis
01.07.2018
30.04.2020
22 months
Leader: ENG
T4.3: Embedded Intelligence and local analytics
01.07.2018
30.04.2020
22 months
Leader: ST-I
T4.4: End-to-End Semantic Interoperability
01.07.2018
30.04.2020
22 months
Leader: FORTH
T4.5: End-to-End Security and Privacy
01.07.2018
30.04.2020
22 months
Leader: UP
WP4: Pattern-driven smart behavior of IIoT with End-to-End Security and Privacy
01.06.2018
30.06.2020
25 months
Leader: FORTH
T5.1: KPIs and Evaluation Methodology
01.06.2019
31.10.2019
5 months
Leader: UP
T5.2: Software system integration
01.06.2019
31.08.2020
15 months
Leader: BS
WP5: System Integration and Evaluation
01.01.2019
31.12.2020
24 months
Leader: ENG
T5.3: IIoT Infrastructure set-up and testing
01.01.2019
31.08.2020
20 months
Leader: IQU
T5.4: Demonstration and validation of IWPC- Energy scenario
01.12.2019
31.12.2020
13 months
Leader: SAG
T5.5: Demonstration and validation of SARA-Health scenario
01.12.2019
31.12.2020
13 months
Leader: ENG
T5.6: Demonstration and validation of IHES-Generic IoT scenario
01.12.2019
31.12.2020
13 months
Leader: ST-I
WP6: Impact, Dissemination and Standardization
01.01.2018
31.12.2020
36 months
Leader: CTTC
T6.1: Impact Creation and Dissemination
01.01.2018
31.12.2020
36 months
Leader: CTTC
T6.2: Exploitation of results
01.06.2018
31.12.2020
31 months
Leader: ENG
T6.3: Standardization
01.01.2019
31.12.2020
24 months
Leader: SAG

Please note that the PERT chart is kept on task level for better readability.

2 MOTIVATION

Interoperability is the ability of components in a system or between systems to work together using common procedures. In such a way it is relatively easy to achieve integration of different systems within the same domain or between different implementations within the stack of a specific software vendor [5]. In the current Internet of Things (IoT) ecosystems, the various devices and applications are installed and operate in their own platforms and cloud services, but without adequate compatibility with products from different brands [6], [7], [8], [9]. For example, a smart watch developed in Android cannot interact with a smart bulb without the relevant proprietary gated application provided by the same vendor. Thus, islands of IoT functionality are established leading towards a vertically oriented ‘Intranet of Things’ rather than the ‘Internet of Things’.

To take advantage of the full potential of the IoT vision, we need standards to enable the horizontal and vertical communication, operation, and programming across devices and platforms, regardless of their model or manufacturer. As it concerns the meaning of data, which is the focus of this deliverable, semantics can settle commonly agreed information models and ontologies for the used terms that are processed by the interfaces or are included in the exchanged data. However, as there are several ontologies for describing each distinct Thing, we need semantic interoperability mechanisms in order to perform common data mapping across the various utilized formats (e.g. XML or JSON) and ontology alignment. Our goal is to enable end-to-end compatibility and cooperation at all layers. Thus, semantic interoperability mechanisms across all layers must be deployed in order to resolve semantics between the field, network, and backend components.

The next subsections analyse the challenges for accomplishing semantic interoperability in the IoT sector. A motivating example is also described, presenting the main features of our proposed solution in a smart sensing setting. The operation of the GW Semantic Mediator (GWSM) component and the Backend Semantic Validator (BSV) component in the backend system is presented (based on the SEMIoTICS architecture D2.4). For clarity, the application of semantic solutions at the field layer of SEMIoTICS is detailed in Deliverable 3.3 and its follow-up, D3.9.

2.1 Challenges for Semantic Interoperability

2.1.1 OVERVIEW

According to the European Research Cluster on the Internet of Things [10], the main purpose of the IoT is not only the connection between devices by using the Internet, but it is also the exchange of web data, due to enabling systems with more capacities to become “smart” (Figure 1). In other words, IoT pursues the integration between the physical and the virtual world by using the Internet as the medium to communicate and exchange information. On the other hand, the heterogeneity of devices, communication technologies and interoperability in different layers in an IoT ecosystem, is a challenge that should be overcome to realize generic IoT solutions at a global scale. Particularly, some high-level interoperability issues should be resolved, for a seamless communication and interaction in IoT environments, such as [11]:

- **Integration of multiple data-sources**: This describes the fundamental requirement for the integration of multiple data/events coming from heterogeneous data sources [12].
- **Unique ontological point of reference**: The semantic interoperability can be achieved by having a unique point of reference at the ontology level. This can be accomplished by
  - third party responsible for translating between different schemes or via ontology merging/mapping
  - protocols for agreeing upon a specific ontology.
- **Mobility and Crowdsensing**: This is related to the necessity of supporting the mobility of the device and the transmission of data beyond boundaries
- **Peer to Peer (P2P) communication**: It is the requirement for applications to communicate at a higher level.
- **Data Modelling and Data Exchange**: Modelling data (data should be based on standards) is one of the major challenges in IoT service deployment; storage and retrieval of this information are also important.

Other main challenges in Semantic Interoperability:

- Ontology merging / Ontology matching & alignment
- Data/Event Semantic Annotation (and dedicated ontologies)
- Knowledge Representation and related ontologies
- Knowledge Sharing
- Knowledge Revision & Consistency
- Semantic Discovery of Data Sources, Data and Services
- Semantic Publish/subscribe & Semantic Routing
- Analysis & Reasoning

Except the above, which are the main high-level challenges in semantic interoperability to be resolved, there are some crucial new security threats to be dealt with in order to allow the continued growth of such ecosystems [13]. Specifically, sensitive data are stored, sent, or received by IoT platforms; as a result, security mechanisms are needed to protect these data from unauthorized access and maybe they should be more complex than in conventional networks. Also, as new security vulnerabilities could be discovered over time, there is the necessity to update IoT platforms on a regular basis, which is not effortless on most of them.

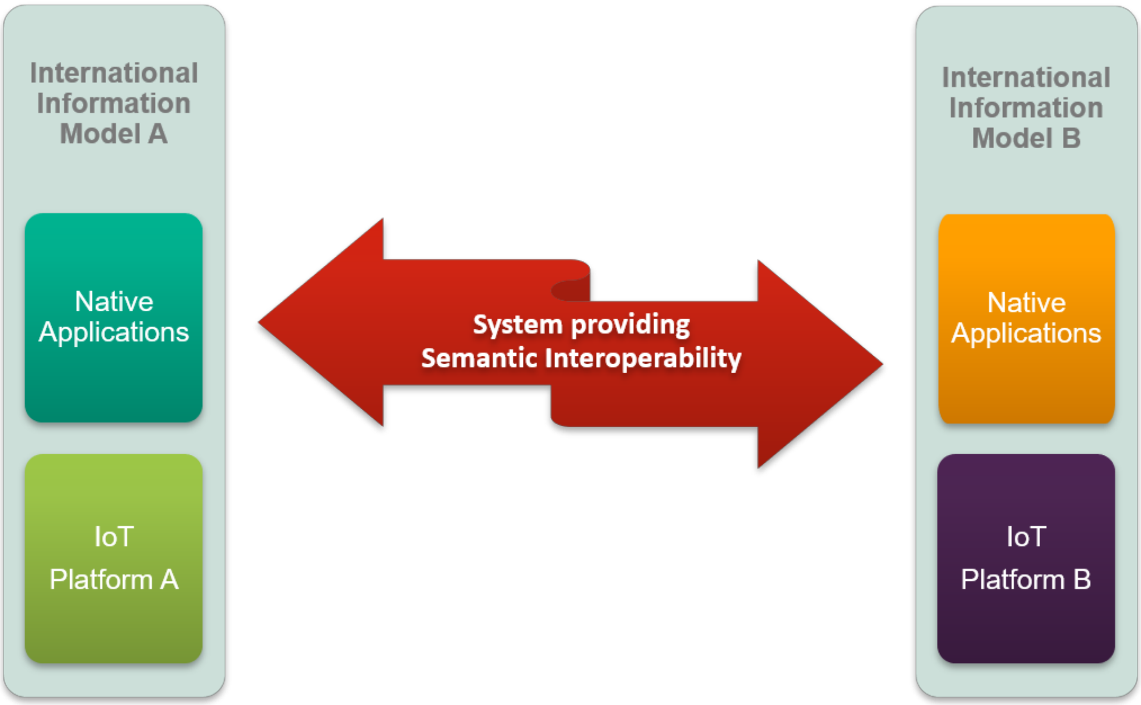

International Information Model A
Native Applications
IoT Platform A
System providing Semantic Interoperability
International Information Model B
Native Applications
IoT Platform B

FIGURE 1 SEMANTIC INTEROPERABILITY SYSTEM IN DIFFERENT IOT PLATFORMS

2.1.2 SEMIOTICS CHALLENGES FOR SEMANTIC INTEROPERABILITY

Taken together, the challenges of IoT, which are related to semantic technologies are organized in the following categories: scalability and flexibility; standardization and reusability; high-level processing; data quality; data confidentiality and privacy; and interpretation and synthesis. Based on the SEMIoTICS requirements, it should tackle some of these challenges.

Specifically, in case that a brownfield device needs to be initialized and registered in SEMIoTICS framework, without a semantic description, a user should add this information (many users should add this information for different UCs). Due to that fact, SEMIoTICS should overcome the challenges of standardization, reusability and data quality. This problem becomes more critical in a healthcare scenario (UC2), as different sensors should be integrated into SARA (see D2.4).

Moreover, systems, like in UC2, will not only own and record information about users but will also produce sensitive data that are rich in context. In this case, data confidentiality and privacy issues should be addressed by SEMIoTICS framework.

2.2 Motivating Scenario – Smart Sensing

We use this motivating example, in order to describe the initial interoperability mechanisms; the final implementation and the specific scenarios of the use of semantic interoperability mechanisms in each of the SEMIoTICS Use Cases are presented in sections 8 and 9 respectively.

Hence, we consider the following smart sensing scenario. A smart building deploys several sensing equipment in order to support pervasive and ubiquitous functionality. Horizontal operation in the field layer is mandatory as well as vertical cooperation with the backend.

The main functionality of the system is the optimization of energy consumption and can be deployed either in the home gateway and/or in the backend layer. The interoperability of the underlying IoT devices and the system services must be guaranteed regardless of their brand or manufacturer. The user should be able to buy and install any smart device while retaining the full functionality of the integrated system.

As an indicative scenario, we consider the case where the user installs temperature environmental sensors. Three types of sensory devices are modelled (see Figure 2):

- the first one is bought from a European vendor – it measures the temperature in the Celsius scale (^{o}C) and transmits data in an XML format
- the second one is bought from the USA – it measures the temperature in the Fahrenheit scale (^{o}F) and transmits JSON messages
- the third sensor is compatible with the semantics of the FIWARE[6] project – it measures the temperature in ^{o}C and transmits JSON messages.

[6] https://www.fiware.org/

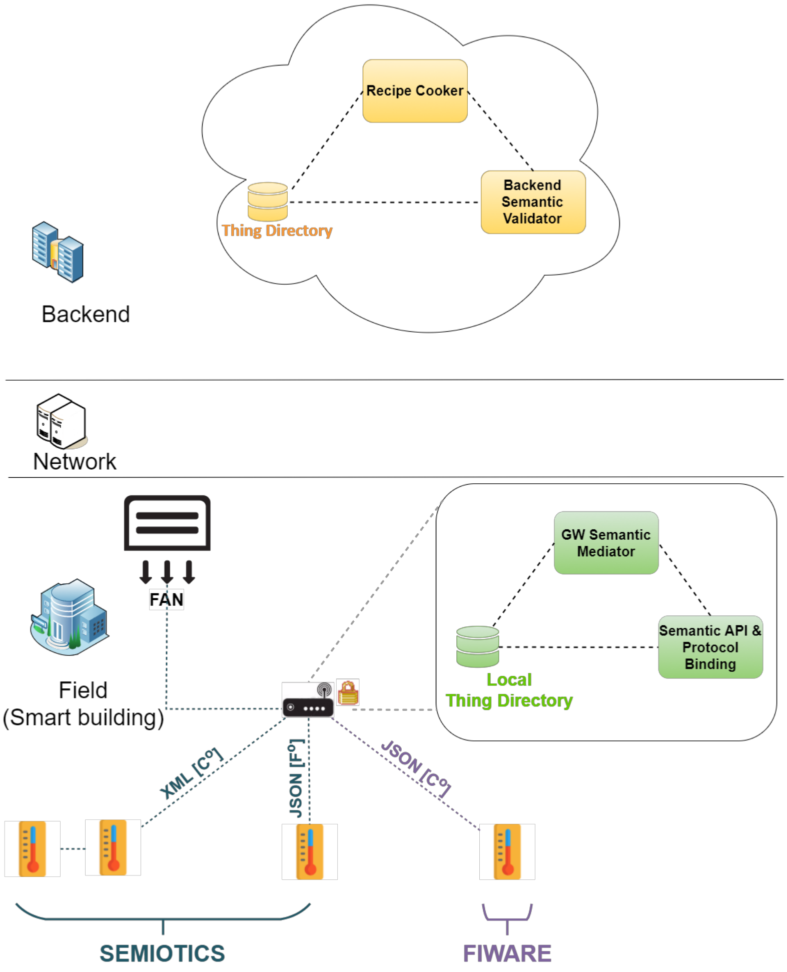


FIGURE 2 THE SMART SENSING INTEROPERABILITY SCENARIO

Then, we model the process of semantic interoperability where the system uses the collected data in order to take real-time decisions. The system functionality must retain a specified temperature value in the building.

The components of the SEMIoTICS architecture (see Deliverable 2.5) that are involved in this process are:

- **Backend Layer:**
  - Backend Semantic Validator: Component responsible for semantic validation mechanisms at the backend layer.
  - Thing Directory: The repository of knowledge containing the necessary Thing models.
  - Recipe Cooker: Component responsible for cooking (creating) recipes reflecting user requirements on different layers (cloud, edge, network) as well as transforming recipes into understandable rules for each layer. It uses the Thing Directory with all the models required to create these rules.
- **Field Layer:**
  - GW Semantic Mediator: Component responsible for the semantic mapping between different data models.
  - Semantic API & Protocol Binding: Component responsible for binding different protocol and exposing a common semantic API located at the Generic IoT Gateway layer.

- Local Thing Directory: The purpose of Local Thing Directory is to store locally the semantic description of Things in the Generic IoT Gateway.

2.2.1 HORIZONTAL SCENARIO – FIELD LAYER

In the first scenario, the semantic interoperability mechanisms run in the local gateway with the aim to retain a specified temperature value in the building. If the temperature in a room goes beyond the specified threshold, the relevant fan equipment is adjusted accordingly. The data flow depicts the sensor device which sends data (temperature) in °C and uses the XML format to transmit data (Figure 3). However, the actuator (fan) requires data in °F and a JSON format file for transmission. Hence, the semantic interoperability mechanisms are responsible for resolving this semantic difference. Particularly, the procedure starts with searching for the necessary Thing models in the Thing Directory Component, in order to detect the above potential semantic conflicts between the interacting Things (sensor, actuator). Afterwards, the Semantic Edge Platform in the Semantic API & Protocol Binding (SAPB) component is responsible to solve the semantic conflicts of temperature units, using the Adaptor Nodes that configure an Interaction Pattern in accordance to the application's requirements. Finally, the GWSM component is triggered to send the request in an appropriate format to the target Thing (actuator).

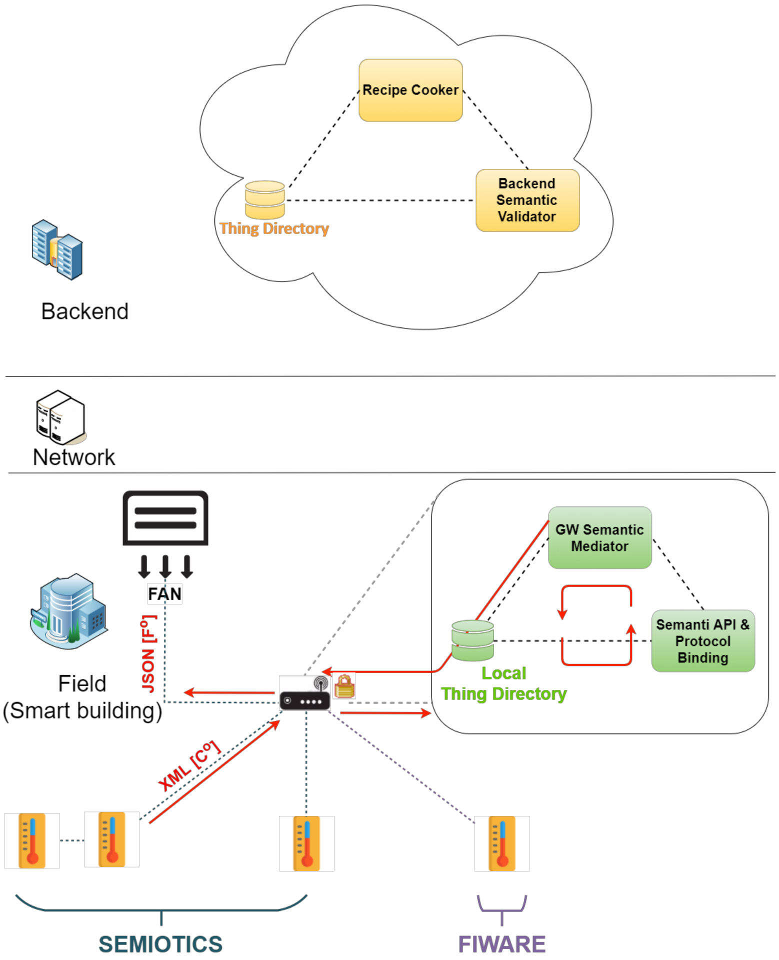


FIGURE 3 HORIZONTAL SCENARIO – IN FIELD LAYER

2.2.2 VERTICAL SCENARIO – BACKEND LAYER

The purpose of the backend layer scenario is the same as the previous, but it requires vertical operation and communication across devices from different layers (Figure 4). Specifically, The BSV can receive a request from an IoT application for the interaction between two Things (i.e. sensor, actuator), which are described with two different TDs (based on W3C Thing Descriptions that are serialized to the JSON-LD standard format), respectively. The functionality of this component consists of:

- Searching for the necessary Thing models in the Thing Directory component, in order to detect any potential semantic conflicts between the interacting domains. In this case, the request refers to a sensor device, which sends the data (temperature) in ºC and uses JSON format to transmit data. However, the actuator (fan) requires data in ºF and XML format file.
- Connecting with the Recipe Cooker component to resolve the semantic conflicts of temperature units, using the Adaptor Nodes that configure an Interaction Pattern in accordance with the application's requirements.
- Transferring the translated request to the SAPB component which is responsible to trigger the GW Semantic Mediator in the field layer, in order to send the request in an appropriate format to the target Thing (actuator).

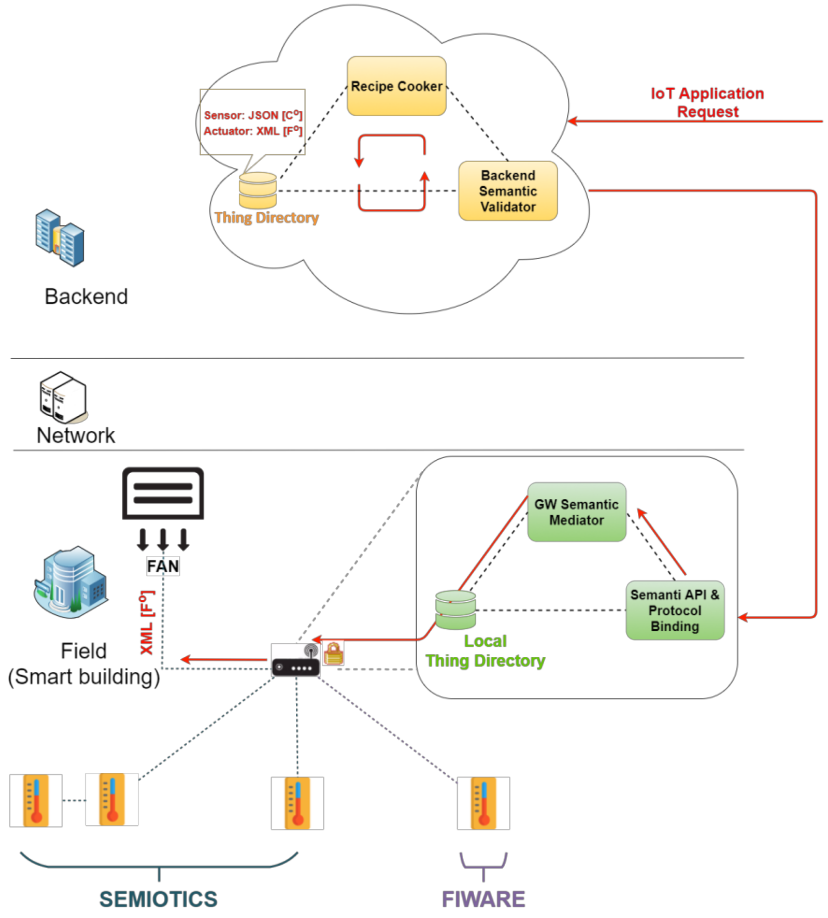


FIGURE 4 VERTICAL SCENARIO – IN BACKEND LAYER

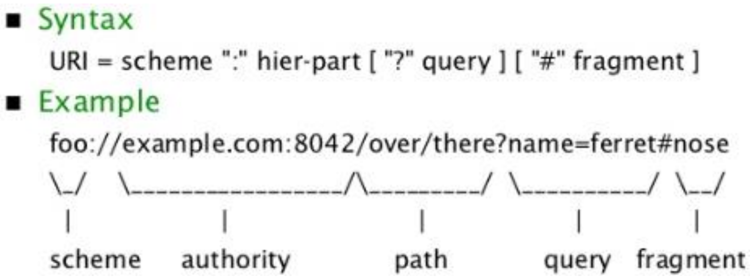
In cases where the abovementioned communication between the field and the backend must be **encrypted**, the semantic functionality is performed in the cloud by the endpoint that decrypts and processes the data.

3 BACKGROUND AND RELATED WORK

This chapter details the background and related work regarding the various semantic technologies. This includes basic notation, the description of Things, ontologies and semantic models for smart objects.

3.1 The Basis for Semantics

To enable the interaction with representations of resources over networks, a form of identification is needed. A **Uniform Resource Identifier (URI)** provides a simple and extensible means for identifying a resource (RFC 3986[7]). The most common form of URI is the Uniform Resource Locator (URL) used to identify webpages on the World Wide Web (WWW). The next figure disassembles the URI syntax.



- Syntax

  URI = scheme ":" hier-part ["?" query] ["#" fragment]

- Example

```
foo://example.com:8042/over/there?name=ferret#nose
\_/   \______________/\_________/ \_________/ \__/
 |           |            |            |        |
scheme   authority       path        query   fragment
```

FIGURE 5 URI SYNTAX

Several schemes are used nowadays for URIs, like *http://*, *ftp://*, *tel:*, *urn:*, and *mailto:* (the URI scheme is not the same as the underlying protocol). The HTTP URIs are the most common data access and things identification mechanism. They provide globally unique names, distributed ownership, and allow people to look up those names.

The **Resource Description Framework (RDF)** is a data format for representing things and their interrelations. The RDF data model is formed as triples of {*subject* → *predicate* → *object*}. For example, we can express the working relations of a person named George, who is employed by FORTH that is based in Heraklion (Code 1).

```
George → worksFor → FORTH
FORTH → basedin → Heraklion
<http://dbpedia.org/resource/FORTH>
    <http://xmlns.com/foaf/0.1/based_near>
        <http://sws.geonames.org/351940/>
```

CODE 1 RDF EXAMPLE

In contrast to the Semantic Web technologies that focus on the ontological level or knowledge inference, Linked-Data (LD) is mainly designed for publishing structured data in RDF using URIs. The provided simplification lowers the entry barrier for data provider and enables the wide-spread adoption.

The LD approach proposes 4 principles:

1. Use URIs to name things on the Web
2. Use HTTP URIs allowing to look-up those names on the Web
3. When someone looks up a URI, provide useful information
4. Include links to other URIs to allow discovery of more things

The abovementioned links are usually RDF properties that are interpreted as hyperlinks. The LD setting accomplishes ease of discovery and information consumption, reduced redundancy, and added value.

[7] https://www.ietf.org/rfc/rfc3986.txt

JSON-LD is a popular implementation of the LD concept. It is developed by leveraging the *Schema.org* vocabulary. It is a joint effort by Google, Bing, Yahoo, and Yandex to establish a unified structured data vocabulary for the Web. JSON-LD annotates elements on a web page and structures the data. These features are utilized by search engines in order to disambiguate elements and derive facts surrounding entities. Once associated, they can create a more organized and better Web overall.

The following code sample (Code 2) describes a technician that can repair the sensors in the motivating example (European sensors that measure temperature in the Celsius scale and transmit XML messages).

```
{
    "@context": "http://schema.org/",
    "@type": "Person",
    "name": "George Brown",
    "jobTitle": "Technician",
    "telephone": "(425) 123-4567",
    "url": "http://www.georgebrown.com",
    "expertise": {
      "@type": "Sensor",
      "@id": "semiotics:sensor_type1@example.org",
      "name": "European Temperature Sensor"
    }
  }
```

CODE 2 JSON-LD TECHNICIAN EXAMPLE

The next code sample (Code 3) represents the expanded version of the abovementioned JSON-LD data which is also signed with RSA.

```
{
    "@context": [
      {
        "@version": 1.1
      },
      "http://schema.org/",
      "https://w3id.org/security/v1"
    ],
    "@type": "Person",
    "name": "George Brown",
    "jobTitle": "Technician",
    "telephone": "(425) 123-4567",
    "url": "http://www.georgebrown.com",
    "expertise": {
      "@type": "Sensor",
      "@id": "semiotics:sensor_type1@example.org",
      "name": "European Temperature Sensor"
    },
    "signature": {
      "type": "LinkedDataSignature2015",
      "created": "2018-10-02T09:37:24Z",
      "creator": "https://example.com/jdoe/keys/1",
      "domain": "json-ld.org",
      "nonce": "b99461eb",
      "signatureValue": "ltGu17tYQWE0MsI3dmqXv2POJF1MykjTOXtn2A5EtMXdmrSmRKUCtvv8jOTmJxJT
BxAIgZR0nDfwEUj+DSa2SUA41NRK6+plPGsj5fvCCFK5rwM8KFVPTDP8CfBG5CSsQW3faf3oudu5yjNCbxuHUFNvL
6UMkDEeyP1MCcGOR0s="
    }
  }
```

CODE 3 JSON-LD TECHNICIAN EXAMPLE – EXPANDED AND SIGNED

3.2 Semantic Models for Smart Objects

In computer and information science an ontology is defined as “a formal, explicit specification of a shared conceptualization” [14] and is used to represent knowledge within a domain as a set of concepts related to each other. In the area of IoT domain, an ontology provides all the crucial semantics for the IoT devices as well as the specifications of the IoT solution (input, output, control logic) that is deployed in such devices. The abovementioned semantics shall include the terminology related to sensors and observations and extend them to capture also the semantics of devices beyond sensors (e.g. actuators, tags, embedded devices, features of interest). Ontologies should be:

- clear (definitions should be objective and complete),
- coherent (should sanction inferences that are consistent with the definitions),
- extendable (should be able to define new terms based on the existing vocabulary without the need of revising the existing definitions),
- the conceptualization should be specified at the knowledge level without depending on a symbol-level encoding.

There are four main components that an ontology is composed of: classes (concepts), individuals (instances), relations and attributes. Classes being the main concepts to be described, they can have one or several children, known as subclasses, used to define more specific concepts. Classes and subclasses have attributes that represent their properties and characteristics. Individuals are instances of classes or their properties. Finally, relations are the edges that connect all the presented components. There are numerous IoT ontologies, such as:

- SWAMO[8] - created to enable dynamic, composable interoperability of sensors, web products and services. It focuses on the sensor domain and particularly on processes to control them. It is interoperable with the Sensor Web Enablement (SWE) descriptions.
- CSIRO[9] - designed to describe and reason about sensors, observations and scientific models. It provides a semantic description of sensors for use in workflows. It was used to develop the Semantic Sensor Network (SSN) ontology.
- The OMA LWM2M architecture from the Open Mobile Alliance[10] for M2M[11] or IoT[12] device management is based on a client component, which resides in the LWM2M Device, and a server component, which resides within the M2M Service Provider or the Network Service Provider. A client can have any number of resources and these resources are organized into objects. The OMA Lightweight M2M enabler focuses on device management and service enablement for LWM2M Devices. Each resource supports

[8] https://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/#SWAMO

[9] https://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/#CSIRO_Sensor_Ontology

[10] https://en.wikipedia.org/wiki/Open_Mobile_Alliance

[11] https://en.wikipedia.org/wiki/Machine_to_machine

[12] https://en.wikipedia.org/wiki/Internet_of_things

one or more operations. The Omalwm2m ontology describes the resources, objects and operations supported by the OMA LWM2M architecture.

The SSN ontology is based on the concepts of systems, processes and observations. It supports the description of the physical and processing structure of sensors. An SSN declares descriptions of sensors, networks and domain concepts in order to provide support in querying, searching, managing data and the network. Semantics follow a horizontal and vertical modularization architecture by including a lightweight but self-contained core ontology called Sensor, Observation, Sample, and Actuator (SOSA) for its elementary classes and properties with their different scope and different degrees of axiomatization. The SOSA provides a formal but lightweight general-purpose specification for modelling the interaction between the entities involved in the acts of observation, actuation, and sampling.

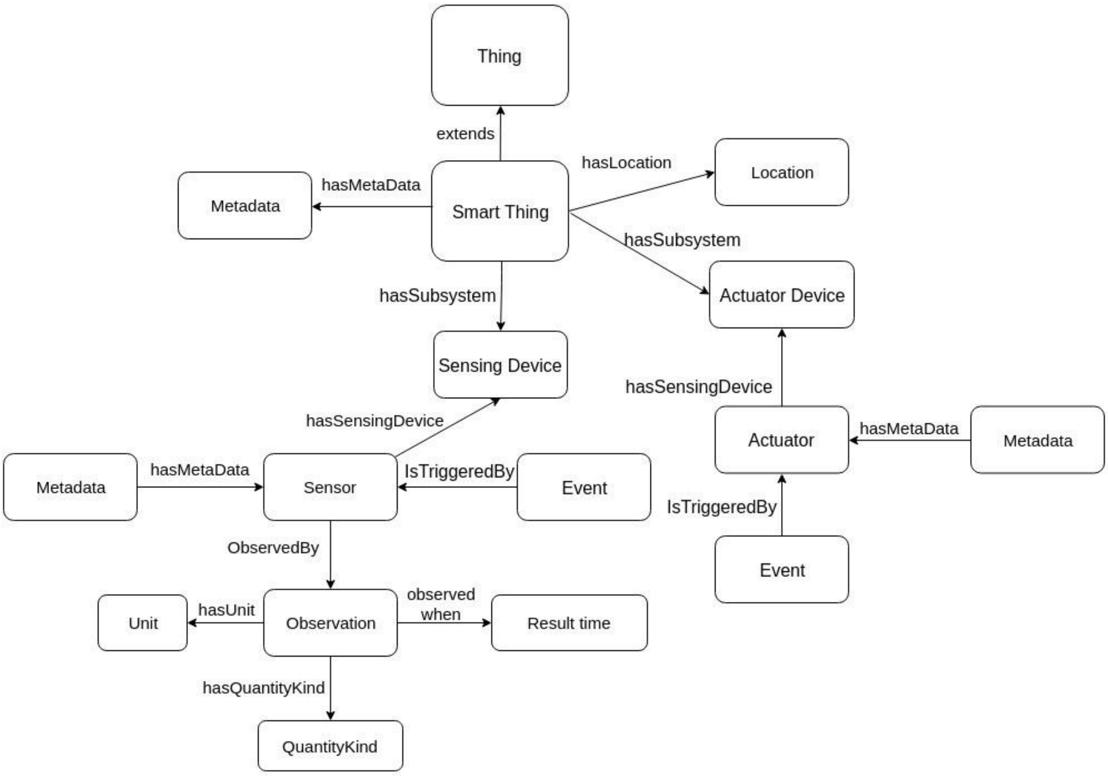


FIGURE 6 EXTENDED MODEL BASED ON SSN ONTOLOGY

Particularly, the above model could be extended with the additional required concepts to model the targeted application scenarios in a specific system (Figure 6). A possible semantic model is structured around entities like smart thing, actuator device, actuator, event, sensing device, sensor, observation, result time, unit, quantity kind, metadata, location. Smart thing stands for an IoT object. Sensing device refers to a device that implements sensing and it can contain many Sensors. Sensor is a device that has the capability to measure a physical property of the real world (e.g. temperature or smoke sensor). Observation is an activity conducted by a sensor in order to measure a physical property (e.g. the readings of a thermometer). QuantityKind represents the essence of a quantity without any numerical value or unit. Unit is real scalar quantity, defined and adopted by convention, with which any other quantity of the same kind can be compared to express the ratio of the two quantities as a number (e.g. degrees Celsius, meter, pound). Result time is the time when the Observation act was completed. Metadata refers to data about the properties (e.g. the metadata for a given sensor could be its precision, sensitivity, accuracy and so on). Actuator device refers to a device that implements actuating, an Actuator Device could contain many Actuators. The actuator is a device that has the capability to perform an operation on or control a system/physical entity in the real world (e.g. relays, solenoids, linear actuators). Event is a certain action that triggers an Actuator to perform an operation or it triggers Sensor to start sensing. We assume that Sensors can work in two modes, the first one conducts observations at specific, time intervals (e.g. the observation of temperature every 10 minutes), the second is that the observation begins only when a specific Event occurs (e.g. observation of the GPS location of a car might be initialized when the car is in motion, in specific terms when speedometer's value is not equal to 0). Location identifies a point or place where the Smart Thing is deployed [15], [16].

3.3 Ontologies

3.3.1 OVERVIEW

The most notable effort in the IoT field is the SSN ontology and Sensor Observation Sampling Actuator (SOSA) ontology by the W3C community [17]. The SOSA/SSN ontologies model sensors, actuator, samplers as well as their observation, actuation, and sampling activities. The ontologies capture the sensor and actuator capabilities, usage environment, performance, and enable contextual data discovery. This also constitutes the standardized ontologies for semantic sensor networks. The cooperation of SSN and SOSA offers different scopes and degrees of axiomatization that enable a wide range of application scenarios towards the Web of Things [18].

More specifically, the SSN ontology is a suite of general-purpose ontologies. It embodies the following 10 conceptual modules: 1) Device, 2) Process, 3) Data, 4) System, 5) Deployment, 6) PlatformSite, 7) SSOPlatform, 8) OperatingRestriction, 9) ContraintBlock, and 10) MeasuringCapability. The modules consist of 41 concepts and 39 object properties.

The general approach regarding the semantic interoperability that is followed by several IoT initiatives, like the EU funded projects Open source solution for the Internet of Things (OpenIoT) [19] and INTER-IoT [20], is the usage of the SSN/SOSA ontologies as the semantic base. The ontologies are then extended with the additional required concepts to model the targeted application scenarios. Such concepts usually include relevant standards and ontologies for specific application areas, like e-health [21], and less often extensions at the sensor level (as the relevant SSN/SOSA information is quite complete). Other similar and popular IoT ontologies include the Smart Appliance REFerence (SAREF) [22] and the MyOntoSens [23].

The following table lists the potential namespaces that are utilized for the SEMIoTICS.

TABLE 1 MODEL NAMESPACES FOR SEMIOTICS

Prefix	Ontology/Language	Namespace
core	SEMIoTICS ontology	http://schema.semiotics.org/core/
rdf	RDF concepts vocabulary	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	RDF schema ontology	http://www.w3.org/2000/01/rdf-schema#
schema	Schema.org ontology	http://schema.org/
iotschema	iotschema.org	http://iotschema.org/
td	Thing Description	http://www.w3.org/ns/td#
xsd	XML schema definition	http://www.w3.org/2001/XMLSchema#

3.3.2 ONTOLOGIES IN HEALTHCARE DOMAIN

This section gives an overview of ontologies that are relevant within the healthcare domain. Traditionally, the semantic ontologies and standards play a significant role in the medical sciences since much of the available medical research needs an avenue to be shared across disparate computer systems [24]. Specifically, the ontologies can provide a basis for searching context-based medical research information, hence it can be integrated and used for future research; the semantic web standards can offer the communication across different Electronic Health Records (EHRs) systems. Thus, a challenging issue in the field of healthcare domain is providing interoperability among healthcare systems [25] that enable universal forms of knowledge representation that integrate heterogeneous information, answer complex queries, and pursue data integration and knowledge sharing in healthcare [26].

In the literature, there has been a growing interest in the medical sciences/healthcare ontologies and standards. Particularly, there are three main organizations that are included in international standards for EHRs. These incorporate the International Organization for Standardization (ISO), the Committee European Normalization (CEN), and the Health Level 7 (HL7)—U.S. based (HL7, 2004). Table 2 presents a few of the standards currently used for interoperability in the semantic web [24].

TABLE 2 EXTRA STANDARDS FOR INTEROPERABILITY - HEALTHCARE DOMAIN

Name	Purpose	Associated Organization
Clinical Document Architecture CDA	Leading standard for clinical and administrative data exchange among organizations	HL7
Guidelines Interchange Format (GLIF)	Specification for structured representation of guidelines	InterMed Collaboratory
CORBAmed	Provides interoperability among health care devices	Object Management Group
HL7	Messaging between disparate systems	HL7

A list of medical sciences/healthcare ontologies is summarized in Table 3 [24].

TABLE 3 ONTOLOGIES- HEALTHCARE DOMAIN

Name	Purpose	Associated Organization
DAML	Extension of RDF which allows ontologies to be expressed; formed by DARPA Markup	DAML Researcher Group
Arden Syntax	Standard for medical knowledge representation	HL7
Riboweb Ontology	Facilitate models of ribosomal components and compare research results	Helix Group at Stanford Medical Informatics
Gene Ontology	To reveal information regarding the role of an organism's gene products	GO Consortium
LinkBase	Represents medical terminology by algorithms in a formal domain ontology	L&C
GALEN	Uses GRAIL language to represent clinical terminology	OpenGALEN
ADL	Formal language for expressing business rules	openEHR
SNOWMED	Reference terminology	SNOMED Int'l
LOINC (Logical)	Database for universal names and codes for lab and clinical observations	Regenstrief Institute, Inc.

UMLS—Unified Medical Language	Facilitates retrieval and integration of information from multiple sources; can be used as a basic ontology for any medical	US National Library of Medicine
ICD-10	Classification of diagnosis codes; is the newer version after ICD-9	National Center for Health Statistics
CPT Codes	Classification of procedure codes	American Medical Association

4 SEMIoTICS SEMANTICS

This chapter details the semantics that are supported by the SEMIoTICS project. Linked Data is the main method that is utilized for publishing structured data using standard Web and Semantic technologies, like HTTP, RDF, and URIs. Ontologies that are designed for the IoT domain by the World Wide Web Consortium (W3C) community are used to address the SEMIoTICS requirements.

4.1 Data Mappings in SEMIoTICS

Semantic mappings is a layer that we introduce in the SEMIoTICS project with the aim to map and integrate brownfield semantics (existing meta-data related to field devices) with Industrial Internet of Things (IIoT) semantics (new semantic models developed for IoT applications). In this layer, we have to provide a mapping knowledge, which can be used to map semantics from a brownfield semantic standard into another IIoT standard. The SEMIoTICS IoT Gateway is able to utilize the mapping knowledge and thus is enabled to integrate data and metadata from appropriate field devices into a harmonized IoT access layer. This is accomplished based on the W3C Web of Things (WoT) standard – Thing Description.

Thing Description is serialized in **JSON for Linking Data** (JSON-LD). It is a serialization format for JSON (a widely adopted serialization and messaging format on the Web). JSON-LD enables JSON data to be interlinked and structured based on semantic models. Thus, it brings the Linked Data paradigm to JSON. There exist implementations and tools for processing and querying JSON-LD data.

4.2 Ontologies in SEMIoTICS

One of the purposes of SEMIoTICS is to integrate an extremely large amount of IoT heterogeneous entities, which need to be consistently and formally represented and managed. Such entities are sensor and actuation devices as well as applications that utilize them. For that reason, an API will be designed to provide access semantically described data along with descriptions of capabilities of connected devices. This API is based on **WoT** upcoming standard, and things are specified in the WoT TD format. TD is semantically annotated with **iot.schema.org**. The iot.schema.org is a community organization for extending **schema.org** to connected Things. Jointly, W3C WoT and iot.schema.org, instate a layer for semantic interoperability which renders the software capable in interacting with the physical world. This interplay is abstracted in such a manner where the development of applications across various IoT settings and domains is easy and simplified. Hence, in SEMIoTICS, we will focus on using existing standards to describe things, as only the standard semantics provides the necessary base for the interoperability. Thus, we will extend iot.schema.org with standard semantics that is required for SEMIoTICS use cases (see Deliverable 3.3 and its follow-up, D3.9).

4.3 TDs for all the Types of Smart Objects in SEMIoTICS

The purpose of this subsection is to give a brief overview of the TDs of SEMIoTICS sensors; it describes, from semantically perspective, the smart objects in SEMIoTICS. The following presentation is divided according to the sensors used in each UC.

4.3.1 TDS OF UC1 SENSORS

This subsection contains the semantic models with the Thing Description for the sensors that are used in Use Case 1. The following Table 4 summarizes the properties, the data types and the actions for each Thing.

TABLE 4 DESCRIPTION OF A SUBSET OF SENSORS ONBOARD USED IN UC1

Thing (type)	Properties	Data Types	Actions
IpCamera	count	integer	StartRecording
	lastChange	string	StopRecording
Microphone	count	integer	startMicrophone
	lastChange	string	stopMicrophone
TemperatureSensing	count	integer	
	lastChange	string	
	temperature	Celsius	

An example of the corresponding JSON-LD file based on the above Table 4 is following (Code 4):

```
{
    "title": "counter",
    "description": "counter example Thing",
    "@context": ["https://www.w3.org/2019/wot/td/v1", {
        "iot": "http://iotschema.org/",
        "schema": "http://schema.org/"
    }, {
        "@language": "en"
    }],
    "@type": ["Thing", "iot:IpCamera", "iot:Microphone", "iot:TemperatureSensing"],
    "security": ["nosec_sc"],
    "properties": {
        "count": {
            "type": "integer",
            "description": "current counter value",
            "iot:Custom": "example annotation",
            "observable": true,
            "readOnly": true,
            "writeOnly": false,
            "forms": [{
                "href": "http://localhost:8080/counter/properties/count",
                "contentType": "application/json",
                "op": ["readproperty"],
                "htv:methodName": "GET"
            }, {
                "href": "http://localhost:8080/counter/properties/count/observable",
                "contentType": "application/json",
                "op": ["observeproperty"],
                "subprotocol": "longpoll"
            }, {
                "href": "coap://localhost:5683/counter/properties/count",
                "contentType": "application/json",
                "op": ["readproperty"]
            }]
        },
        "lastChange": {
            "type": "string",
            "description": "last change of counter value",
            "observable": true,
            "readOnly": true,
            "writeOnly": false,
            "forms": [{
                    "href": "http://localhost:8080/counter/properties/lastChange",
                    "contentType": "application/json",
                    "op": ["readproperty"],
                    "htv:methodName": "GET"
                },
                {
                    "href": "http://localhost:8080/counter/properties/lastChange/observab
le",
                    "contentType": "application/json",
                    "op": ["observeproperty"],
                    "subprotocol": "longpoll"
                },
                {
                    "href": "coap://localhost:5683/counter/properties/lastChange",
                    "contentType": "application/json",
                    "op": ["readproperty"]
                }
            ]
        },
        "temperature": {
            "@type": "iot:Temperature",
            "iot:capability": "iot:TemperatureSensing",
            "description": "current temperature value",
            "observable": true,
            "readOnly": true,
            "writeOnly": false,
            "type": "integer",
            "schema:minValue": -10,
            "schema:maxValue": 100,
            "iot:temperatureUnitCode": "iot:Celsius",
            "forms": [{
                    "href": "http://18.222.210.109/counter/properties/temperature",
                    "contentType": "application/json",
                    "op": ["readproperty"],
                    "htv:methodName": "GET"
                },
                {
                    "href": "http://18.222.210.109/counter/properties/temperature/observa
ble",
                    "contentType": "application/json",
                    "op": ["observeproperty"],
                    "subprotocol": "longpoll"
                },
                {
                    "href": "coap://localhost:5683/counter/properties/temperature",
                    "contentType": "application/json",
                    "op": ["readproperty"]
                }
            ]
        }
    },
    "actions": {
        "startMicrophone": {
            "@type": "iot:StartRecording",
            "iot:capability": "iot:Microphone",
            "description": "Starts recording the audio.",
            "forms": [{
                "href": ".../actions/startMicrophone",
                "contentType": "audio/mpeg",
                "op": ["invokeaction"],
                "htv:methodName": "POST"
            }],
            "idempotent": false,
            "safe": false
        },
        "stopMicrophone": {
            "@type": "iot:StopRecording",
            "iot:capability": "iot:Microphone",
            "description": "Stops recording the audio.",
            "forms": [{
                "href": ".../actions/stopMicrophone",
                "contentType": "audio/mpeg",
                "op": ["invokeaction"],
                "htv:methodName": "POST"
            }],
            "idempotent": false,
            "safe": false
        },
        "startCamera": {
            "@type": "iot:StartRecording",
            "iot:capability": "iot:Camera",
            "description": "Starts recording the video.",
            "forms": [{
                "href": "http://.../counter/actions/startCamera",
                "contentType": "video/mp4",
                "op": ["invokeaction"],
                "htv:methodName": "POST"
            }],
            "idempotent": false,
            "safe": false
        },
        "stopCamera": {
            "@type": "iot:StopRecording",
            "iot:capability": "iot:Camera",
            "description": "Stops recording the video.",
            "forms": [{
                "href": "http://.../counter/actions/stopCamera",
                "contentType": "video/mp4",
                "op": ["invokeaction"],
                "htv:methodName": "POST"
            }],
            "idempotent": false,
            "safe": false
        }
    }
}
```

CODE 4 TDS OF UC1 SEMIOTICS SENSORS

For clarity, the description of a subset of sensors on board used in use case 1 of SEMIoTICS is detailed in D3.9.

4.3.2 TDS OF UC2 SENSORS

This subsection contains the semantic models with the Thing Description for the sensors that are used in Use Case 2. The following Table 5 summarizes the properties, the data types and the actions for each Thing.

TABLE 5 DESCRIPTION OF A SUBSET OF SENSORS ON BOARD USED IN UC2

Sensor	Properties	Actions	Data Types
Ultrasonic range	obstacleL (ObstacleSensors)		Meters
	obstacleC (ObstacleSensors)		Meters
	obstacleR (ObstacleSensors)		Meters
Accelerometer	Acceleration		Number (float)
	Resolution	SetResolution	

An example of the corresponding JSON-LD file based on the above Table 5 is following (Code 5):

```
{
    "@context ":"https://www.w3.org/2019/td/v1",
    "id":"urn:wotrrsara",
    "title":"WoT_RR_SARA",
    "description":"RR_SARA",
    "security":[
      "psk_sec"
    ],
    "securityDefinitions":{
      "psk_sec":{
        "scheme":"psk"
      }
    },
    "properties":{
      "obstacleL":{
        "description":"obstacleL",
        "type":"application/json",
        "forms":[
          {
            "href":"coap://localhost:5683/obstacleL"
          }
        ]
      },
      "obstacleC":{
        "description":"obstacleC",
        "type":"application/json",
        "forms":[
          {
            "href":"coap://localhost:5683/obstacleC"
          }
        ]
      },
      "obstacleR":{
        "description":"obstacleR",
        "type":"application/json",
        "forms":[
          {
            "href":"coap://localhost:5683/obstacleR"
          }
        ]
      },
      "accelx":{
        "description":"accelx",
        "type":"application/json",
        "forms":[
          {
            "href":"coap://localhost:5683/accelx"
          }
        ]
      },
      "accely":{
        "description":"accely",
        "type":"application/json",
        "forms":[
          {
            "href":"coap://localhost:5683/accely"
          }
        ]
      }
    }
  }
```

CODE 5 TDS OF UC2 SEMIOTICS SENSORS

For clarity, the description of a subset of sensors on board used in use case 2 of SEMIoTICS is detailed in D3.9.

4.3.3 TDS OF UC3 SENSORS

This subsection contains the semantic models with the Thing Description for the sensors that are used in Use Case 3. The following Table 6 summarizes the properties, the data types and the actions for each Thing.

TABLE 6 DESCRIPTION OF A SUBSET OF SENSORS ONBOARD USED IN UC3

Sensor	Properties	Actions	Events	Data Types
IHES Sensing Node	nodeIdentifier	reset	temperatureSensor	array(float)
	operationStatus	state	humiditySensor	array(float)
	connected		pressureSensor	array(float)
			accelerometerSensor	array(float)
			changeDetected	string

An example of the corresponding JSON-LD file based on the above Table 6 is following (Code 6):

```
{
    "@context":"https://www.w3.org/2019/wot/td/v1",
    "id":"urn:00-80-e1-00-00-99",
    "title":"IHES Sensing Node",
    "securityDefinitions":{
      "no_sc":{
        "scheme":"nosec"
      }
    },
    "security":[
      "no_sc"
    ],
    "properties":{
      "nodeIdentifier":{
        "description":"Shows the node ID.",
        "type":"string",
        "forms":[
          {
            "href":"mqtt://192.168.200.2:1883/ihes/node/out/00-80-e1-00-00-99/events"
          }
        ]
      },
      "operationStatus":{
        "description":"Shows the current status of the node.",
        "type":"string",
        "forms":[
          {
            "href":"mqtt://192.168.200.2:1883/ihes/node/out/00-80-e1-00-00-99/events"
          }
        ]
      },
      "connected":{
        "description":"Shows if the node is connected or not.",
        "type":"boolean",
        "forms":[
          {
            "href":"mqtt://192.168.200.2:1883/ihes/node/out/00-80-e1-00-00-99/events"
          }
        ]
      }
    },
    "actions":{
      "reset":{
        "description":"Reset the node.",
        "inputSchema":{
          "type":"string",
          "const":"RESET"
        },
        "forms":[
          {
            "href":"mqtt://192.168.200.2:1883/ihes/node/in/00-80-e1-00-00-99"
          }
        ]
      }
    },
    "events":{
      "temperatureSensor":{
        "description":"Provides periodic temperature value updates.",
        "data":{
          "type":"array"
        },
        "forms":[
          {
            "href":"mqtt://192.168.200.2:1883/ihes/node/out/00-80-e1-00-00-99/data"
          }
        ]
      },
      "humiditySensor":{
        "description":"Provides periodic humidity value updates.",
        "data":{
          "type":"array"
        },
        "forms":[
          {
            "href":"mqtt://192.168.200.2:1883/ihes/node/out/00-80-e1-00-00-99/data"
          }
        ]
      },
      "pressureSensor":{
        "description":"Provides periodic pressure value updates.",
        "data":{
          "type":"array"
        },
        "forms":[
          {
            "href":"mqtt://192.168.200.2:1883/ihes/node/out/00-80-e1-00-00-99/data"
          }
        ]
    },
    "accelerometerSensor":{
      "description":"Provides periodic accelerometer value updates.",
      "data":{
        "type":"array"
      },
      "forms":[
        {
          "href":"mqtt://192.168.200.2:1883/ihes/node/out/00-80-e1-00-00-99/data"
        }
      ]
    },
    "changeDetected":{
      "description":"Provides change in data, when detected.",
      "data":{
        "type":"string"
      },
      "forms":[
        {
          "href":"mqtt://192.168.200.2:1883/ihes/node/out/00-80-e1-00-00-99/data"
        }
      ]
    }
  }
}
```

CODE 6 TDS OF UC3 SEMIOTICS SENSORS

5 SEMANTIC MEDIATOR MECHANISMS

This chapter describes in more detail the operation of the semantic interoperability mechanisms. Particularly, data models and techniques are presented in order to map the data in a common format (i.e. JSON). For ontology alignment, transformation rules are retrieved and performed by the semantic interoperability procedure.

5.1 Ontology Alignments & Data Transformation Techniques

5.1.1 DATA MAPPING

5.1.1.1 YANG MODEL

YANG is a data modelling language which was originally developed to model Remote Procedure Calls (RPCs), notifications, configurations, state data of network elements, as well as constraints to be enforced on the data [27]. Additionally, it can be used to describe other network constructs, such as services, protocols, policies and customers [28]. It follows a hierarchical organization; namely, data is structured into a tree and it can contain complex types, such as lists and unions. Also, YANG supports the NETCONF [29] and RESTCONF [30] interfaces for the deployment of network and RESTful services, respectively. The service operations are modelled in YANG. Then, the YANG processor parses the model and exports the abstract development project in a denoted programming language (e.g. JAVA, C/C++, etc). YANG is also utilized to transform data from one format to another.

These features could be adapted in order to deploy the smart functionality that collects, processes, and transmits the sensed information. Particularly, the RESTCONF and the implemented RESTful web services could be used to run in the field and backend systems. Moreover, a YANG model was used to establish a common data mapping between the involved operations. The interfaces can process messages (such as get the current temperature value from a sensor) with semantic information. Afterwards, at runtime, XML messages can be transformed into JSON and vice versa, according to the specific format which is supported by each interface. The IEFT Internet Draft *draft-ietf-netmod-yang-json*[13] establishes a one-to-one mapping between JSON and the subset of XML that can be modelled by YANG. The overall functionality is also tailored in order to cooperate with legacy formats, as in the IoT domain there could be several constrained devices, like motes/sensors, that do not process structured data.

5.1.1.2 DEVELOPMENT FUNCTION:

An alternative solution for mapping between XML and JSON is the implementation of a function using algorithms and libraries from specific programming languages. There are many mappings between JSON and XML, and programs implementing those mappings. Specifically, these techniques are based on the fact that XML and JSON data objects are multi-branch tree structures. Any XML/JSON file can be parsed and translated by recursive traversal of its tree object. Particularly, a traversal algorithm can be divided into the steps below:

1. Get the root of the tree object
2. Get a value from the node of a, if the value is a number of child nodes, then traverse all child nodes, do operation a on all child trees whose roots are these child nodes, otherwise return the value of the node

However, there are already many libraries for translating XML to JSON, JSON to XML or both. One of the most widespread solutions is the JSON package in Java [package org.json][14], which includes the capability to convert between JSON and XML in Java language. It is open-source and could address the requirements of the data mapping semantic interoperability mechanism in SEMIoTICS. For that reason, this approach is most suitable for SEMIoTICS framework and is used for the data mapping in a common format (i.e. JSON).

5.1.2 ONTOLOGY ALIGNMENT

[13] https://tools.ietf.org/html/draft-ietf-netmod-yang-json-10

[14] https://github.com/stleary/JSON-java.git

After defining a common format, the next step is to resolve any potential semantic conflicts and perform ontology alignment between the interacting domains. In that case, **transformation rules** can be used to describe how we can transform the data that are processed by one application into a compatible form that is understandable by another machine.

5.1.2.1 JSON REGULAR EXPRESSION:

A possible and simplified solution for this issue is to model the rules as specific JSON tags that are included in the related TD/JSON-LD files. Each rule tag can contain the identification of the two domains (from-to) and a **Regular Expression (RE)**. The RE could be a valid PERL[15] program that models the search pattern (matching the data to be altered) and the transformation formula itself (how the data will be changed). For example, the next TD sample (Code 7) transforms the temperature value from the Celsius to the Fahrenheit scale. Once parsed by the inference engine, the rule could take as input the JSON-LD file from a FIWARE's set_temperature service operation, search for the temperature value and change it to the other scale.

```
{ … JSON-LD TD file …
  "transformation_rules": [
    {
        "from": "temperature_celsius",
        "to": "temperature_fahrenheit",
        "RE": "my $data=$ARGV[0];
         if ($data =~ m/fan_power=(\\d+)/){
             my $fahrenheit=$1*9/5+32;
            $data =~ s/fan_power=(\\d+)/fan_power=$fahrenheit/g;
        }
         if ($data =~ m/set_temperature=(\\d+)/) {
            my $fahrenheit=$1*9/5+32;
            $data=~ s/set_temperature=(\\d+)/set_temperature=$fahrenheit/g;
        }
    }
  {other rules},] …… end of data TD … }
```

CODE 7 TRANSFORMATION RULE IN JSON-LD FILE IN PERL

The expressiveness of this type of RE is even more advanced than just performing a single mathematic formula. REs could perform complicated transformations and successfully resolve the conflicts that occur from the incorrect OWL correlations.

Figure 7 illustrates the above description of the semantic interoperability procedure.

[15] https://www.perl.org/

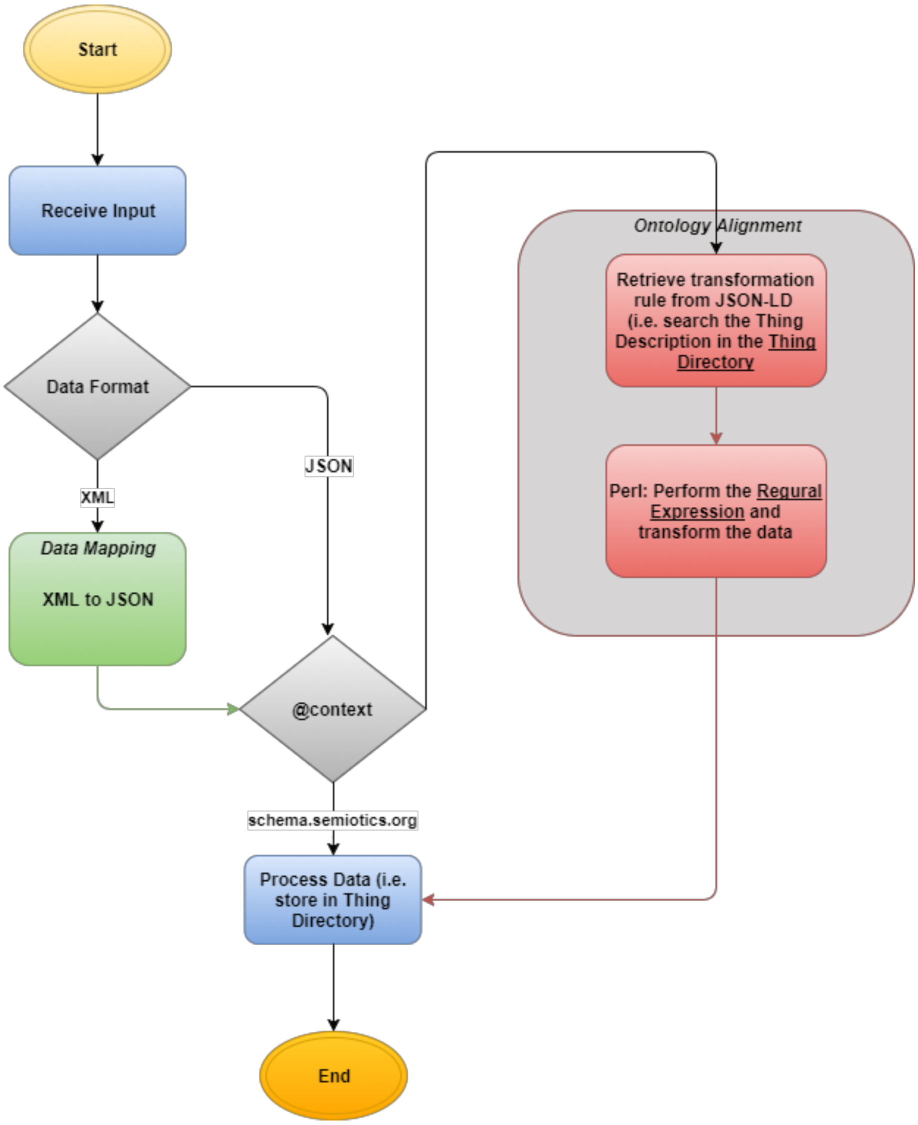

Start
Receive Input
Data Format
JSON
XML
Data Mapping
XML to JSON
@context
schema.semiotics.org
Process Data (i.e. store in Thing Directory)
End
Ontology Alignment
Retrieve transformation rule from JSON-LD (i.e. search the Thing Description in the Thing Directory
Perl: Perform the Regural Expression and transform the data

FIGURE 7 SEMANTIC INTEROPERABILITY PROCEDURE: DATA MAPPING AND ONTOLOGY ALIGNMENT WITH JSON RE-ACTIVITY DIAGRAM

5.1.2.2 DROOLS PATTERN ENGINE:

Another suggestion for the description of transformation rules is the use of Drools[16] . It is a rule engine system, which combines rule-based techniques and object-oriented programming. It is based on the Event-Condition-Action (ECA) structure of rules which is a flexible method to achieve process adaptation, different from a series of nested if-then-else statements that do not offer workflow readability (Code 8). From the developing point of view [31]:

1. Drools uses objects as marked out by patterns and rules that invoke certain actions.
2. A pattern is a coded expression (program), which manipulates one or more objects to form a pattern to make, adapt or fashion behavior according to designed logic.

[16] https://www.drools.org/

3. Drools objects are Java objects and can be represented by instances of Java classes or XML schemas.
4. A rule can perform many types of actions, such as a), add or remove an object from the working memory, b) modify an object, c) execute a method on one of the objects

```
rule "<name of rule>"
      <attribute> <value>
      when
            <LHS>
      then
            <RHS>
end

*LHS: Pattern-matching against objects in the Working Memory, RHS: Code executed
when a match is found
```

CODE 8 RULE EXAMPLE - DROOLS

Additionally, the decisive advantage of Drools is the fact that the development is interactive; the work environment provides the capability to quickly add change rules and re-run test cases. This supports the automatic synchronization between the semantic reasoning procedure and the rule engine. Whenever an object is updated not only the corresponding semantic concept in the model is updated, but the Drools engine is also triggered to evaluate its rules. In this way, the changes in the semantic model are propagated all the way up to Drools.

Together with the above analysis, making use of Drools for the transformation rules gives the following benefits: a) the object-oriented character is very suitable to topic map elements, b) the rule engine adopts Rete algorithm[17] and implementation for the Java language and it can reason on rules effectively, c) its open-source project brings the benefit of overriding the code expediently to fit the practical applications.

5.1.2.3 EXTENSION RECIPE MODEL – ADAPTOR NODES IN NODE-RED PLATFORM:

Node-RED[18] is an open-source programming tool for wiring the IoT, hardware devices, APIs and online services, created by the Emerging Technology team of IBM. It includes a browser-based flow editor to wire together flows using a wide range of nodes. The flows can be then deployed at runtime. Hence, **nodes** and **flows** are the two fundamental components in Node-RED. Nodes can be divided into three categories, input, output and function. The connection between specific nodes designs the above flows. Additionally, from a programming point of view, JavaScript functions can be created/extended within the editor using a rich text editor and flows are stored using the JSON format, which can be easily imported and exported for sharing with other applications.

The Recipe Model (see Deliverable 4.1 and its follow-up, D4.8) and the Recipe Cooker component (see Deliverable 2.4) are based on Node-RED. Thus, a proposed methodology, which the purpose of resolving any potential semantic conflicts, can be the addition of extra nodes to the core node palette of Node-RED. Particularly, this technique is used for SEMIoTICS to resolve any potential semantic conflicts and perform ontology alignment between the interacting domains, in order to have consistency and interactivity between the components of the architecture. In the next subsection, a full set of Adaptor nodes are described in order to address the SEMIoTICS requirements.

5.1.3 FULL SET OF ADAPTOR NODES

The SEMIoTICS framework aims to provide ways of adapting and/or replacing concrete IoT application smart objects/components if it becomes necessary at runtime (e.g., when some components become unavailable) in

[17] https://en.wikipedia.org/wiki/Rete_algorithm

[18] https://nodered.org/

order to ensure that data flow is possible between smart objects. In such circumstances, it requires the dynamic adaptation of the available infrastructure (sensors) in order to cope with changes in smart objects and IoT platform configurations. For this reason, is it possible to change one sensor to another with the same functionality (belonging to the same category e.g. a temperature Thing) in this case it should be checked whether the input / output of the two devices obey the same format. For this purpose, Adaptor Nodes are offered.

Based on subsection 4.3, which presents the TDs for all types of smart objects that are used in SEMIoTICS, we can come up with the possible corresponding classes of Adaptor Nodes that are intended to meet the requirements of all UCs. Particularly, in UC1 (see 4.3.1), the TD of temperature sensor has the attribute ***"iot:temperatureUnitCode": "iot:Celsius"***; it means that the temperature units is Celsius. For this case we provide nodes to convert data:

- from Celsius to Fahrenheit and vice versa
- from Celsius to Kelvin and vice versa
- from Fahrenheit to Kelvin and vice versa.

In UC2 (see 4.3.2), the accelerometer Thing uses as data type number (float) for counting. Thus, we provide nodes to convert data:

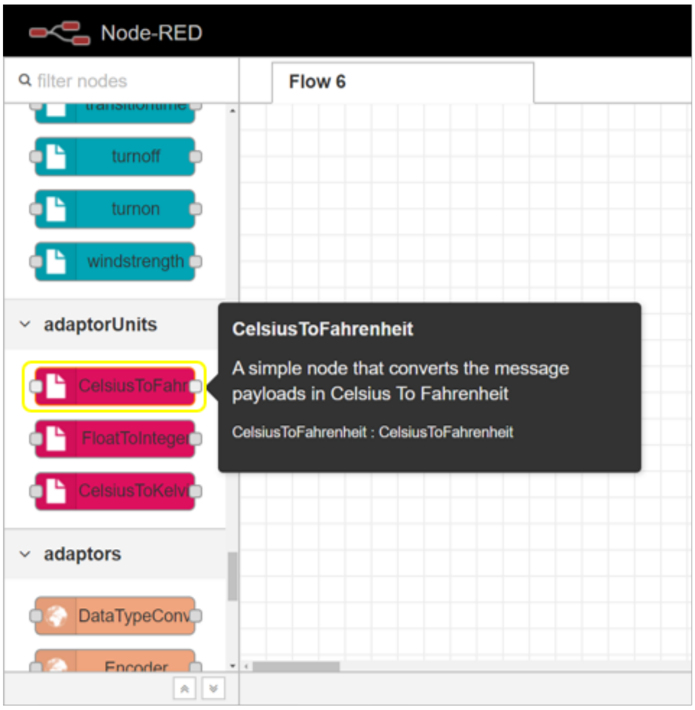
- from float to string and vice versa
- from float to integer and vice versa.

Finally, in UC3 (4.3.3) the temperatureSensor, humiditySensor, pressureSensor and accelerometerSensor Thing use as data type float for counting. For this UC, in the same token as the UC2, we provide nodes to convert data:

- from float to string and vice versa
- from float to integer and vice versa.

Said Adaptor Nodes are produced at runtime with the corresponding functionality. The implementation and the application of them are detailly described in Section 8. Figure 8,Figure 9,Figure 10 highlight some of the Adaptor Nodes that are available and developed in SEMIoTICS to deal with the UCs requirements.

The above Adaptor Nodes can be used in combination; for example, if it is required an advanced conversion such as Fahrenheit to Celsius and String to Integer. In this case, two Adaptor Nodes are used parallel in line and the output of the first is the input of the second.

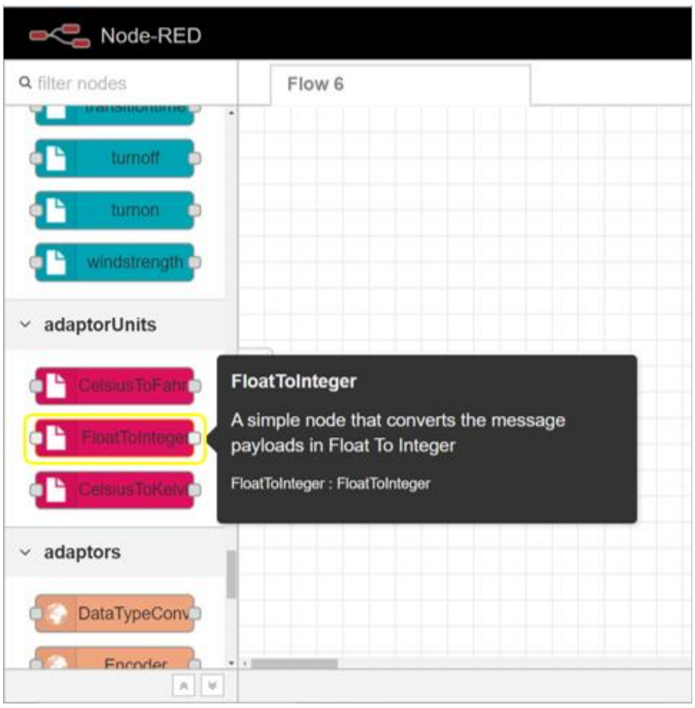


CelsiusToFahrenheit

A simple node that converts the message payloads in Celsius To Fahrenheit

CelsiusToFahrenheit : CelsiusToFahrenheit

FIGURE 8 1ST ADAPTOR NODES EXAMPLE IN SEMIOTICS UCS

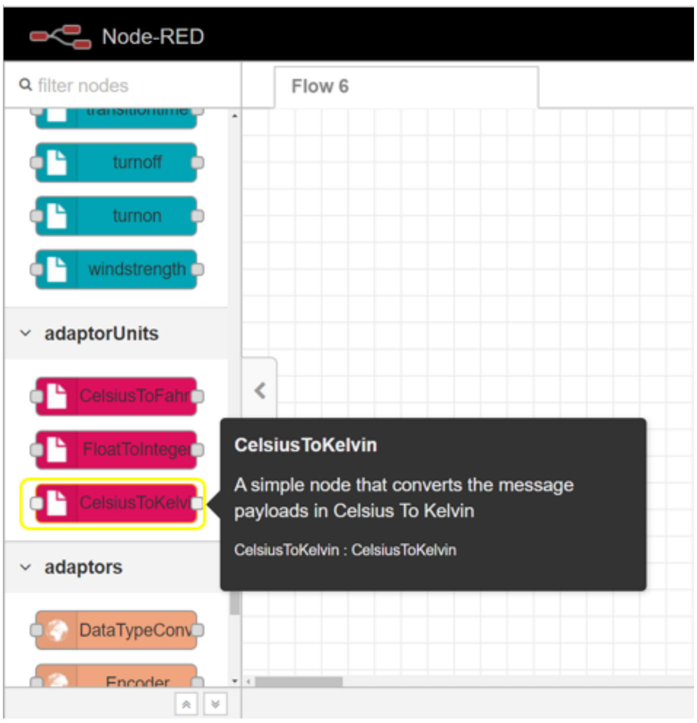


FloatToInteger

A simple node that converts the message payloads in Float To Integer

FloatToInteger : FloatToInteger

FIGURE 9 2ND ADAPTOR NODES EXAMPLE IN SEMIOTICS UCS



CelsiusToKelvin

A simple node that converts the message payloads in Celsius To Kelvin

CelsiusToKelvin : CelsiusToKelvin

FIGURE 10 3RD ADAPTOR NODES EXAMPLE IN SEMIOTICS UCS

s

5.2 Semantic Reasoning

The term semantic alignments has been used in the literature to adapt and transform the data into the declared ontologies [32]. Depending on the expressiveness of the ontologies, reasoning engines can further infer associations and links into the data. Hence, the reasoning is responsible for making conclusions and deriving new facts, which do not exist in the knowledge base. Traditionally, reasoning with rules is based on first-order predicate logic or description logic to make conclusions from a sequence of statements derived by predefined rules [33]. A reasoning engine (i.e., a reasoner) is a software tool that realizes reasoning with rules. Most of them manage a comprehensive set of RDFS and OWL vocabularies and most RDF data formats.

Various semantic reasoning engines have been proposed. Specifically, the Jena inference subsystem[19], Pellet[20], RacerPro[21], HermiT[22], RIF4J[23] and Fact++[24] are based on different rule languages and can give

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

[20] https://www.w3.org/2001/sw/wiki/Pellet

[21] http://franz.com/agraph/racer/

[22] http://www.hermit-reasoner.com/

[23] http://rif4j.sourceforge.net/

[24] http://owl.man.ac.uk/factplusplus/

support for ontologies and OWL. On the other hand, there are some reasoners which support SWRL[25] and RIF[26] rule languages and others have implemented their own human-readable rule syntaxes.

Moreover, there are sensor-based linked open rules [34] for sharing and reusing rules in IoT applications based on Jena rules syntax. Besides that, another methodology [35] is used to link and reuse rules on the Web which is expressed as SPARQL queries. Also, in [36], a cloud platform has been proposed for editing and reusing SPARQL queries online.

Taking all the above together, the query language for the Semantic Web is SPARQL, which offers an appropriate way to interrogate multiple triple-stores (RDF stores) over HTTP. It was designed by the W3C RDF Data Access Working Group (DAWG) and it is considered one of the key technologies for the semantic web. Conjunctions, disjunctions, triple patterns, and some optional patterns are supported by SPARQL; also, SQL syntax such as SELECT and WHERE clauses can be used as part of a SPARQL query, i.e. (Code 9):

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>

SELECT ?name ?email
WHERE {
?subj a foaf:Person .
?subj foaf:name ?name .
?subj foaf:mbox ?email .
        }
```

CODE 9 SPARQL QUERY FOR SELECTING ALL THE NAMES AND EMAIL ADDRESSES OF RDF DATA THAT HAS THE TYPE FOAF:PERSON

The above example builds on the friend-of-a-friend ontology definition (foaf), and it provides a simple query to return all the names and email addresses of RDF data that has the type foaf:Person.

Despite the fact that SPARQL is the standard query language for retrieving and manipulating RDF data, the majority of SPARQL implementations requires the data to be available in advance (in main memory or in a repository), thereby not exploiting the real-time and dynamic nature of Linked Data. An interesting solution is presented by the DAWG group [37], SPARQL-LD[27], which is an extension of SPARQL and allows to directly fetch and query RDF data from any HTTP Web source.

The following piece of code in SPARQL-LD (Code 10) selects the technicians that are expert in repairing a specific sensor type.

```
SELECT DISTINCT ?technician

WHERE {
      SERVICE <http://schema.semiotics.org/core/Technician> {
            Semiotics:Technician semiotics-owl ?expertise }
      VALUES ?templ {
            <http://schema.semiotics.org/core/Senor/Type1>}
}
```

CODE 10 SPARQL-LD QUERY FOR SELECTING THE EXPERT TECHNICIANS FOR A SPECIFIC

[25] http://www.w3.org/Submission/SWRL/

[26] http://www.w3.org/TR/rif-in-rdf/

[27] https://github.com/fafalios/sparql-ld

Another solution, which is based on SPARQL, is the Thingweb Directory[28]. It is an open-source implementation for TD models; these models are recommended by the W3C Web of Things working group. It provides an API to Create, Read, Update and Delete (CRUD) a TD. It can be used both to browse and discover Things based on their TDs.
Specifically, TDs can be filtered using SPARQL. Matching a SPARQL filter requires the inclusion of semantic annotations in the TD. The following SPARQL filter (Code 11) selects TDs that include the char "2" in the name property. A SPARQL filter for Thingweb Directory should be sent in a query parameter and percent-encoded[29].

Particularly, this technique (see Thing Directory component) is used for SEMIoTICS for semantic reasoning procedure.

```
?prop <http://www.w3.org/ns/td#name> ?name . FILTER contains(?name, "2");
```

CODE 11 SPARQL FILTER IN THINGWEB DIRECTORY

[28] https://github.com/thingweb/thingweb-directory

[29] https://lists.w3.org/Archives/Public/public-wot-ig/2017Nov/0005.html

6 SEMIoTICS INTEGRATION APPROACH WITH OTHER IoT PLATFORMS

This chapter presents the means of integrating SEMIoTICS with other IoT platforms. That includes the interoperability between said heterogeneous IoT platforms and delivery of dynamic interaction between them to interconnect the variety of smart objects covering both multiple technologies and intelligent services.

6.1 Introduction

The interoperability between the SEMIoTICS framework and other external IoT platforms, such as FIWARE, MindSphere and openHAB, works in two directions. The first direction is originating from other IoT platforms, moving towards the SEMIoTICS framework. In that way, SEMIoTICS can use the exposed interfaces of said IoT platforms, in order to take advantage of IoT devices whose descriptions are available in repositories outside SEMIoTICS framework. The second direction is originating from the SEMIoTICS framework, moving towards other IoT platforms. In a similar way, said platforms utilize the SEMIoTICS' exposed interfaces of selected components in order to employ IoT smart objects and services. The next subsections describe the interaction of SEMIoTICS and the above IoT external platforms.

6.2 Interaction with FIWARE

6.2.1 OVERVIEW

The SEMIoTICS platform components can expose dedicated APIs which are visible outside the platform thanks to a router functionality embedded in the backend platform itself. Each component can interact with any other component through the provided API but also with outside components like other platforms APIs or FIWARE's Generic Enablers (GEs) through provided Next Generation Service Interface (NGSI). On the other hand, the component creator can restrict API visibility.

To create more permanent and quick ways to connect with other IoT platforms, the code responsible for connection is created in the form of reusable components. Such components take care of the common task like request mapping, API calling, response mapping, etc.

As it was mentioned in previous SEMIoTICS deliverables, FIWARE is not commercial, but an open-source cloud-based infrastructure for IoT platforms. This solution can even be regarded as an IoT platform or as a platform for platforms, so it is reasonable to consider this solution apart from the other IoT platforms. Taking that unique aspect into consideration, it is reasonable to use or even to incorporate some of FIWARE's GEs to build the SEMIoTICS framework.

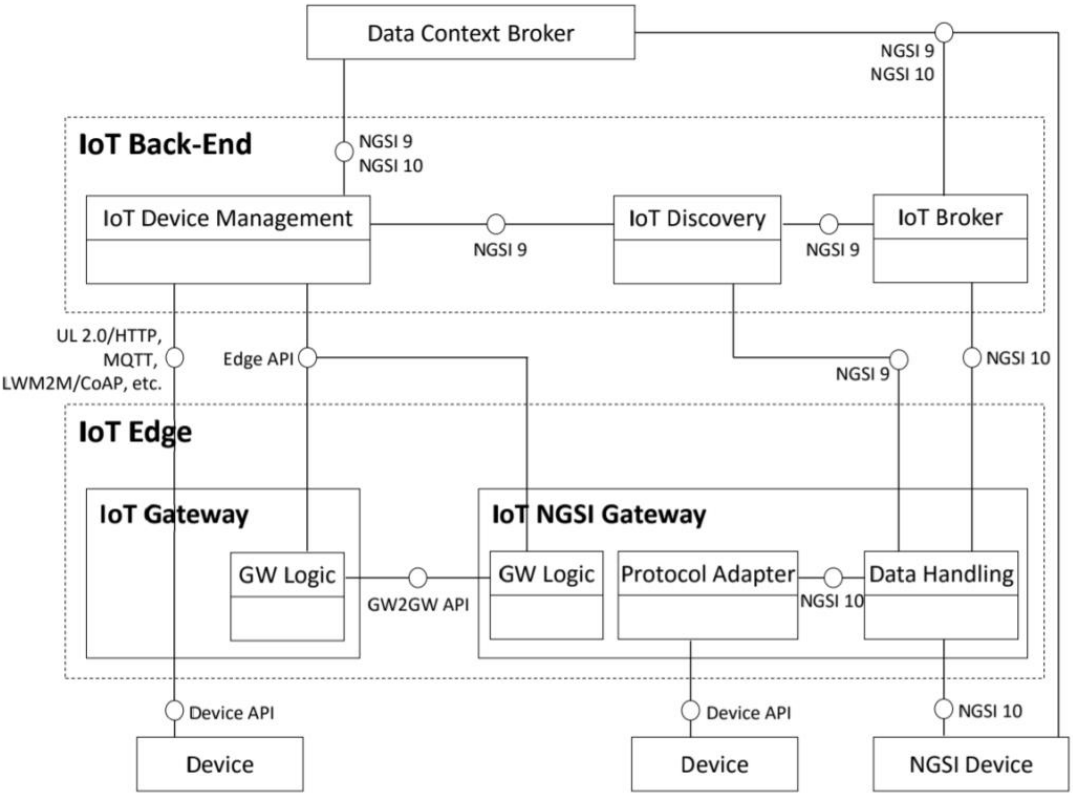


FIGURE 11 FIWARE IOT SERVICES ENABLEMENT ARCHITECTURE

From a semantic point of view, it is worth to mention that FIWARE follows an approach consistent with SEMIoTICS, to represent a Device with integrated specific entities as a whole and do not distinguish between Sensors and Actuators (Figure 11). Moreover, FIWARE proposes semantics built upon iot.schema.org, which is the target semantic core for SEMIoTICS (data in this core semantics are defined in JSON Schema). This allows the use of the open-source FIWARE semantics (schema.fiware.org) instead of working with the raw iot.schema.org. Adopting and improving schemas.org vocabularies can reduce duplication of efforts and focus attention on innovation rather than replication[30]. The exposed by FIWARE Data Models repository contains JSON Schemas corresponding to the models. What can be useful from the perspective of SEMIoTICS semantics is that it is possible to contribute to this project and create additional data models[31] [32] [33].

This data model datasets are exposed through the FIWARE NSGI version 2 API[32]. According to specification, the FIWARE NGSI API defines a data model and a context data interface as well as a context availability

[30] https://iot.schema.org/docs/iot-gettingstarted.html

[31] https://www.fiware.org/developers/data-models/

[32] https://github.com/Fiware/dataModels

[33] https://www.fiware.org/2016/09/02/towards-schema-fiware-org/

interface[34] [35]. The NGSI context data has been presented in the image below (Figure 12). Each entity can have many attributes and can be identified by type and id.

As it was mentioned, integration with FIWARE Data Models is possible via NSGI API, if SEMIoTICS leverages the same API for context queries, context subscription and context updates to interact with the respective context elements (i.e., sensors and actuators) in a FIWARE domain. The Orion context broker is an implementation of the NSGIv2 REST API binding developed as a part of the FIWARE platform. A few exemplifying queries are listed below (more to be found in documentation[34] [35]):

- query for retrieving API resources. Retrieve API Resources returns links to affordances in the form of links in the JSON body,
- retrieving query List Entities which returns all matches by different criteria,
- query Create Entity where object follows JSON Entity Representation.

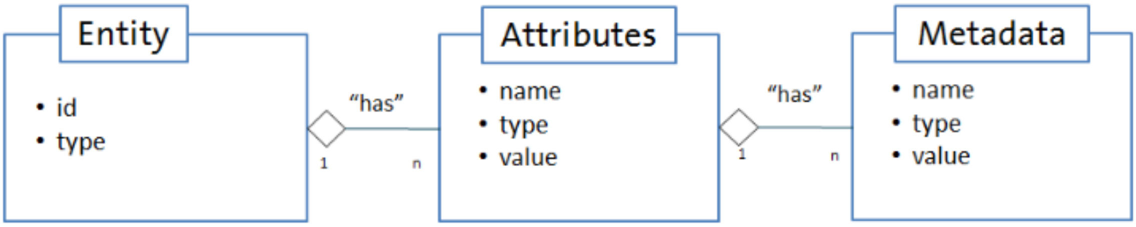


FIGURE 12 NSGI CONTEXT DATA [34] [35]

FIWARE provides a few GEs related to semantic interoperability that can be applied to the SEMIoTICS project. These GEs are listed and shortly described below:

IoT Discovery – is a GE written in Java and the meeting point between IoT Context Producers and IoT Context Consumers; it provides APIs for contextual information exchange, or the Sense2Web API that supports Linked Open Data (LOD) (more information in[36]).

The NSGI-9 Server provides a repository for the storage of NGSI entities and allows NSGI-9 clients [38] to register context information about Smart Things as well as discover context information using id, attribute, attribute domain and entity type

Sense2Web API is a platform that provides a semantic repository for IoT providers to register and manage semantic descriptions (in RDF/OWL)

IoT Broker – is a GE exposed by FIWARE as a docker image or to be built from source exchanging information between other components via the NGSI interface.A part of the IoT Broker is an IoT Knowledge Server which contains a large amount of IoT semantic knowledge useful from the perspective of a project. The IoT Knowledge Server is a standalone component created to provide semantic information, which serves IoT Broker semantic ontologies (for example, to explore the information structure contained in real world data). It

[34] http://telefonicaid.github.io/fiware-orion/api/v2/stable/

[35] http://fiware.github.io/specifications/ngsiv2/stable/

[36] https://fiware-iot-discovery-ngsi9.readthedocs.io/en/latest/index.html

provides REST APIs and Subscribe / Notify functionality in JSON format. It is composed by two components (web servers) and two databases along with the servers (Figure 13).

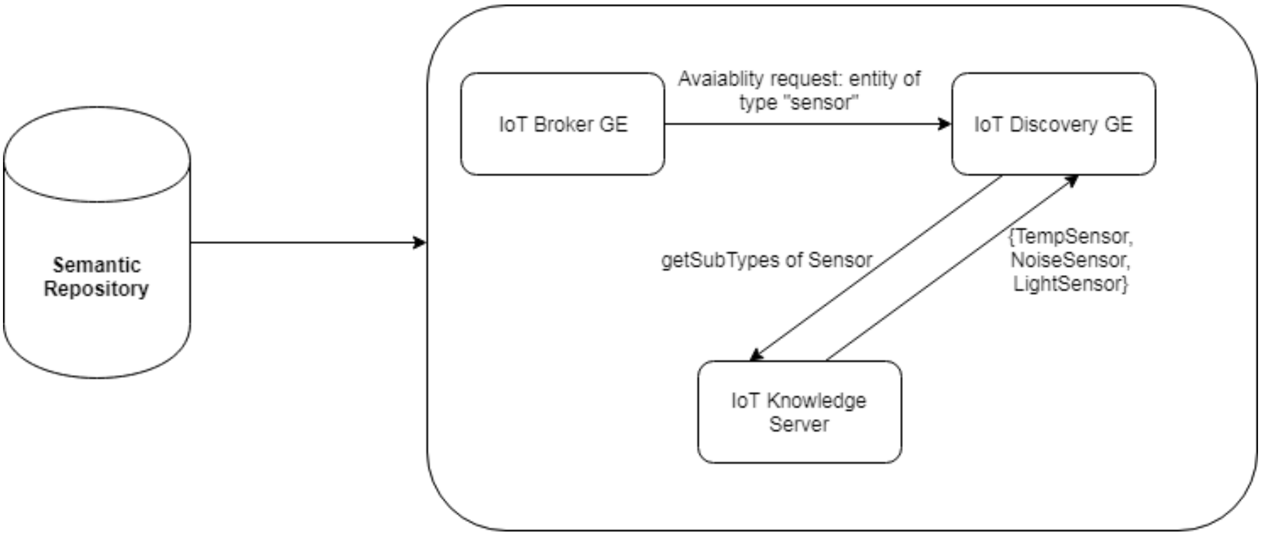


FIGURE 13 IOT KNOWLEDGE SERVER, IOT BROKER AND IOT DISCOVERY INTEROPERATION EXAMPLE (BASED ON [81])

Summary of the FIWARE integration perspective:

It is possible to use data models or to define proprietary data models using a JSON Schema which covers the key-value representation of NGSI v2 context data (not normalized but shorter)[37].

Sense2Web API (from the IoT Discovery GE) which supports LOD, could be used for semantic querying via SPARQL and to register and manage semantic descriptions (in RDF/OWL) so it could be used as a semantic descriptions repository or as a part of it.

IoT Knowledge Server (from the IoT Broker GE) which, as mentioned, contains a large amount of IoT semantic knowledge. The IoT Knowledge Server is a standalone component created for serving semantic information to the IoT Broker semantic ontologies (for example, to explore the information structure contained in the real-world data). It provides REST API and Subscribe /Notify in JSON format. This knowledge could also be incorporated during the process of plugging a new object, for example, creating a new entity (finding matching types with subtypes).

[37] https://fiware-datamodels.readthedocs.io/en/latest/howto/index.html

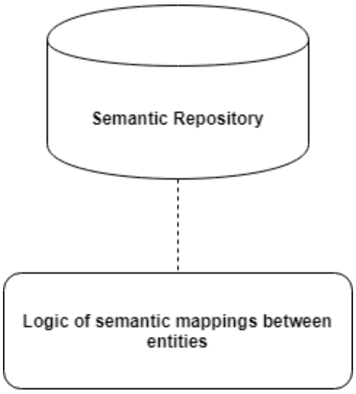


FIGURE 14 LOGIC OF SEMANTIC MAPPINGS BETWEEN COMPONENTS AS A SEPARATE COMPONENT DEFINING TRANSFORMATION MAPPINGS BETWEEN ENTITIES

FIWARE is recently switching to the NSGI-LD specification to enhance relationships between entities, but currently, it is up to the logic of the application (in this case the SEMIoTICS platform) to navigate between entity relationships[37]. A possible need for developing such a component responsible for the logic of semantic mappings has been presented in Figure 14.

The following JSON-LD code (Code 12) describes a smart building for FIWARE.

```
{
    "id": "57b912ab-eb47-4cd5-bc9d-73abece1f1b3",
    "type": "BuildingOperation",
    "dateCreated": "2016-08-08T10:18:16Z",
    "dateModified": "2016-08-08T10:18:16Z",
    "source":  "http://www.example.com",
    "dataProvider": "OperatorA",
    "refBuilding": "building-a85e3da145c1",
    "operationType": "airConditioning",
    "description": "Air conditioning levels reduced due to out of hours",
    "result": "ok",
    "startDate": "2016-08-08T10:18:16Z",
    "endDate": "2016-08-20T10:18:16Z",
    "dateStarted": "2016-08-08T10:18:16Z",
    "dateFinished": "2016-08-20T10:18:16Z",
    "status": "finished",
    "operationSequence": [
     "fan_power=0",
     "set_temperature=24"
    ],
    "refRelatedBuildingOperation": [
      "b4fb8bff-1a8f-455f-8cc0-ca43c069f865",
      "55c24793-3437-4157-9bda-667c9e1531fc"
    ],
    "refRelatedDeviceOperation": [
      "36744245-6716-4a28-84c7-0e3d7520f143",
      "33b2b713-9223-40a5-87a0-3f80a1264a6c"
    ]
  }
```

CODE 12 FIWARE TEMPERATURE JSON-LD EXAMPLE

6.2.2 SCENARIO OF INTEGRATION BETWEEN SEMIOTICS AND FIWARE

This subsection presents the scenario and the corresponding implementation of the first direction (SEMIoTICS uses services from other target external IoT platforms).

For the sake of simplicity, the current example includes the interoperability between SEMIoTICS and FIWARE for the description and analysis of the development of the proposed approach.

Firstly, the key components of the SEMIoTICS architecture (see Figure 15) related to interoperability with external IoT platforms that are involved in this process are:

- Recipe Cooker (RC) which is responsible for cooking (creating) recipes reflecting user requirements,
- Pattern Orchestrator (PO) which oversees the automated configuration, coordination, and management of different patterns (in this case Interoperability patterns) and their deployment,
- Pattern Engine - Backend (PEB) which allows the insertion, modification, execution, and retraction of patterns through the Pattern Orchestrator,
- Backend Semantic Validator (BSV) which resolves semantic interoperability issues and
- Thing Directory - Backend (TDB) which is the repository of knowledge containing the necessary Thing models

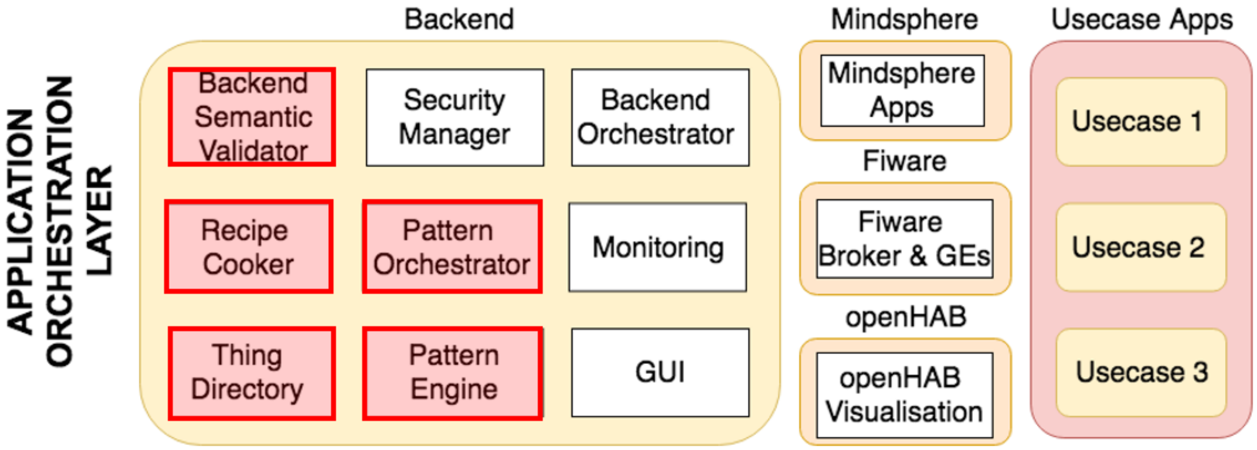


FIGURE 15 KEY COMPONENTS OF THE SEMIOTICS ARCHITECTURE RELATED TO INTERACTION WITH EXTERNAL IOT PLATFORMS

The implementation of the approach includes the interoperability between the components and respective APIs of the framework and the platforms. A number of different steps are required, as described below and as highlighted in the sequence diagram in Figure 16:

- **Step1**: The first phase includes the design of the flow interaction between two Things i.e. FIWARE Sensor, SEMIoTICS TemperatureSensing expressed as a recipe/flow in RC. The main goal is to validate the semantic interoperability between the identified nodes to ensure the required interconnection. For that reason, RC sends the cooked recipe to the PO in order to transform it into architectural patterns (in this case interoperability patterns – see subsection 7.2). The RC is developed based on the Node-RED framework and is responsible for cooking (creating) recipes reflecting user requirements, send the recipe in PO using a POST method request. Particularly, this POST parameters include the recipe/flow (JSON format), as body and the header is application/json.

- **Step2**: The second phase includes the transmission of the interoperability requirement as a POST from the PO to the PEB to enforce the respective pattern rules. These rules are expressed as Drools (see 7.2.3) business production rules, running and the associated rule engine. The latter is an efficient pattern-matching algorithm known to scale well for large numbers of rules and data sets of facts, thus allowing for an efficient implementation of the pattern-based reasoning process. The IoT service workflow in question is described in a dedicated language that the PO understands and is given as input, using the language constructs and methodology defined in D4.8. It exchanges information with the PEB in order to check if a specific property holds throughout the whole workflow and corresponds accordingly.
- **Step3**: The third phase of PEB includes the validation of the semantic interoperability for any links in the recipe/flow as triggered the BSV. In the described example, there is only one link/wire identifying the connection between FIWARE Sensor and SEMIoTICS Thermostat Sensor. The BSV component receives a request from the PEB to check the semantic interoperability between two Things (link) in JSON-LD/JSON format. The JSON-LD/JSON Parser is implemented as part of the BSV, to analyze the received input and extract the meaningful information from these set of data. The communication between said components is achieved using a POST method.
- **Step4**: The fourth phase includes the BSV that begins the procedure to tackle the semantic interoperability issues between these two Things from said recipe/flow. In order to give this answer, the semantic description for any Thing is required (for FIWARE Sensor and SEMIoTICS Thermostat Sensor). For that reason, it sends two requests: (i) SPARQL query to TDB (is developed based on the Thingweb Directory) in order to receive the Thing Description of SEMIoTICS Thermostat and (ii) GET request to the Orion Context Broker FIWARE platform to receive the context data Description of FIWARE Sensor. The response consists of JSON format with the FIWARE Sensor attributes (type, metadata elements).
- **Step 5**: The last phase involves the received by the BSV information in order to decide regarding the interoperability between the Things and harmonize the semantic model capabilities with the registration of extra Adaptor Nodes in the recipe if required. Particularly, there are three possible results. Firstly, the link source and destination are interoperable, so the BSV updates the PEB with the TRUE response. Secondly, the link source and destination are not interoperable and the BSV can add Adaptor Nodes (see 8.3.1 for the development of new nodes in Node-RED framework) in order to guarantee the interoperability. In this case, BSV not only sends the TRUE response in PEB but also updates the recipe in RC using the corresponding Adaptor Nodes. Lastly, the link source and destination are not interoperable and BSV does not have the required information to develop the Adaptor Nodes; hence, the PEB receives the FALSE response by the BSV.

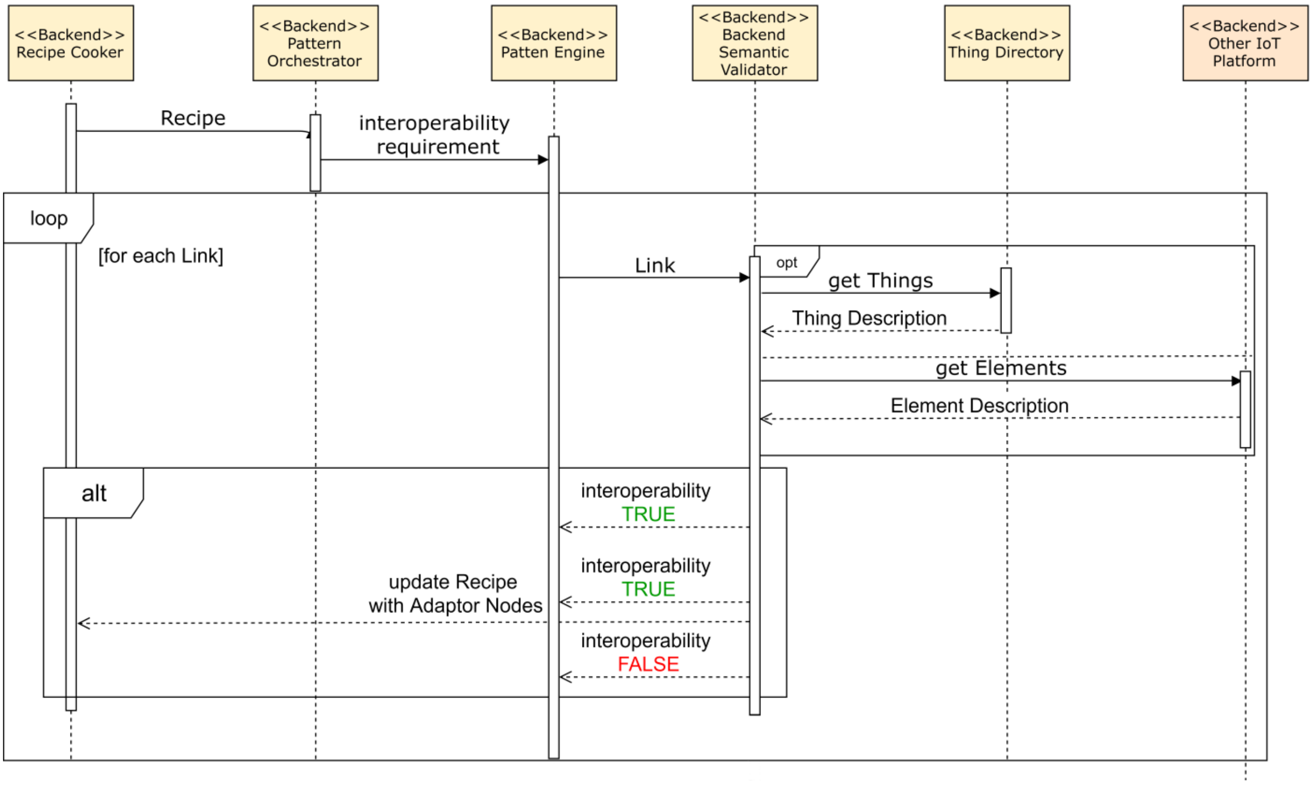

<<Backend>> Recipe Cooker
<<Backend>> Pattern Orchestrator
<<Backend>> Patten Engine
<<Backend>> Backend Semantic Validator
<<Backend>> Thing Directory
<<Backend>> Other IoT Platform
Recipe
interoperability requirement
loop
[for each Link]
Link
opt
get Things
Thing Description
get Elements
Element Description
alt
interoperability TRUE
interoperability TRUE
update Recipe with Adaptor Nodes
interoperability FALSE

FIGURE 16 SEQUENCE DIAGRAM OF SEMIOTICS INTEROPERABILITY WITH EXTERNAL IOT PLATFORMS

Figure 17 demonstrates the initial status of the recipe and the final structure taking advantage of the BSV procedure which tackles the semantic interoperability issues between these two Things. Additionally, it should be mentioned that the above development can be applicable in any other external IoT platforms, except FIWARE, since it provides compatible exposed interfaces and services.

Following the implementation analysis in which the SEMIoTICS is able to use the exposed interfaces of said IoT external platforms, it is worthwhile noting that the opposite direction of interoperability - external IoT platforms utilize the SEMIoTICS' exposed interfaces of selected components in order to employ IoT smart objects and services provides - is available by some components. Namely, one of the selected components that has exposed interfaces is the TDB. Things and their description reside in the TDB. The interface of it can be used to retrieve the already stored semantic description of Things. The TD of the corresponding Thing that is returned, complies with the iotschema and can be used for consumption from other IoT platforms (for more details see Deliverable D3.9).

Furthermore, another component that has exposed interfaces is the PO which, along with the Pattern Engines, can offer not only interoperability but also verification of Security, Privacy, Dependability and along with Interoperability (SPDI) and Quality of Service (QoS) properties as a service to be used from the other IoT platforms. In that way, an external IoT platform may utilize said service in order to verify the SPDI/QoS properties to an existing workflow that is comprised of IoT smart objects (IoT service workflow). The IoT service workflow in question is described in a dedicated language that the PO understands and is given as input, using the language constructs and methodology defined in D4.8. PO exchanges information with the Pattern Engines in order to check if a specific property holds throughout the whole workflow and corresponds accordingly. For more details on this process, the SEMIoTICS pattern language and the mechanisms enabling the pattern-driven monitoring and adaptation please refer to deliverable D4.8.

Finally, the SDN controller of SEMIoTICS (SSC) is also exposed in the scope of offering a service which provides means of managing available OpenFlow devices in the other IoT platforms. More details on the pattern-driven interface of the SSC can be found in deliverable D3.10.

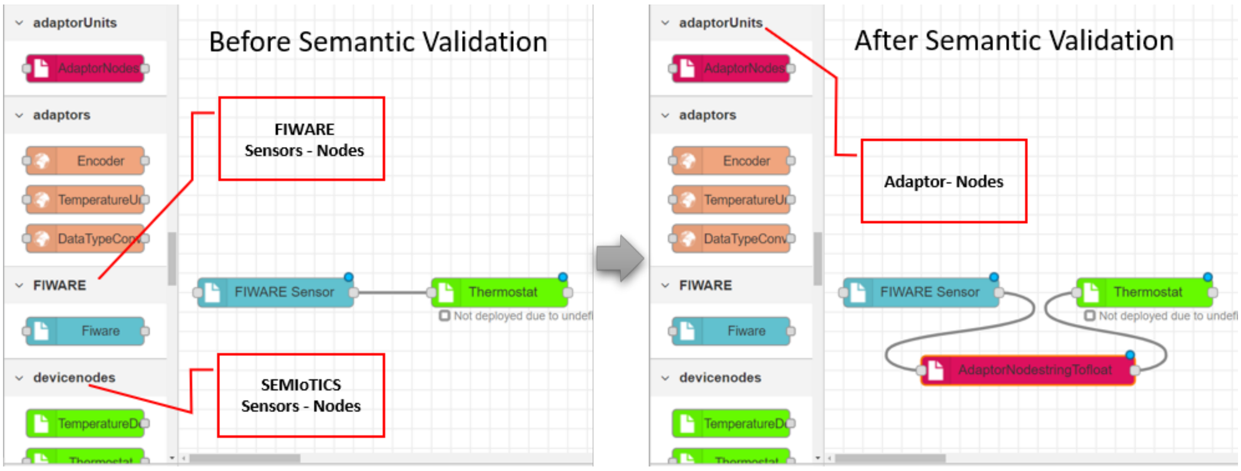


FIGURE 17 INTEROPERABILITY BETWEEN SEMIOTICS AND FIWARE SENSORS - EXAMPLE USING FLOW FROM RECIPE COOKER

6.2.3 FIWARE – SEMIOTICS INTEROPERABILITY LAYER BRIDGE

In Section 6.2.1, the FIWARE Data Models have already presented. Namely, Figure 12 describes the original OMA NGSI meta-model, the fundamental metamodel around the FIWARE Data Models. There are three main elements in this meta-model, as shown in the figure above: Entities, Attributes and Metadata. Entities are the center of gravity in the NGSI information model. An entity represents a thing, i.e., any physical or logical object (e.g., a sensor, a person, a room, an issue in a ticketing system, etc.). Attributes are properties of context entities. Finally, Metadata is used as an optional part of the attribute value; like attributes, each piece of metadata has a name, describing the role of the metadata in the place where it occurs.

Thus, for the interconnection and interoperability between FIWARE and SEMIoTICS, except the proposed approach that has already described in the previous subsection (exposed services which provide the delivery of dynamic interaction between two said platforms), the deployment of a bridge capable to read and write inserted data is required in order to achieve the communication between the two IoT platforms. In fact, based on a study of the data model of FIWARE, an adaptation of the FIWARE Data Model is proposed to match iotschema.org, as used by SEMIoTICS. The main mappings between FIWARE and SEMIoTICS semantic concept are expressed in Table 7.

TABLE 7 AN OVERVIEW OF MAPPING FIWARE DATA MODEL TO IOTSCHEMA.ORG

FIWARE Data Model	iotschema.org
Entity	Thing Description
Id of Entity	Id of Thing Description
Type of Entity	Type of Thing Description
Attributes	Properties

6.3 Interaction with MindSphere

An IIoT Gateway (see deliverable D3.9) is expected to be deployed for the integration of devices from an existing brownfield devices (e.g., wind turbine), as well as new IoT devices that may be added to the system (e.g., a camera, microphone, accelerometer etc.). The role of IIoT Gateway is to expose the functionality of field devices over a uniform interface with clear semantics, i.e., machine interpretable descriptions of field devices. For that purpose, we use the W3C Web of Things with its Thing Description and iotschema.org. Apart from this (southbound) functionality, IIoT Gateway is also used to transfer data to MindSphere[38]. The semantics created in the Field level and exposed over a Thing Description is also used for creating MindSphere asset model. In this way, we have a transparent IoT semantics, not only at the Field level but also across complete SEMIoTICS platform.

The data model of MindSphere is based on *assets*[39]. An asset is a digital representation of a machine or an automation system with one or multiple automation units connected to MindSphere. An asset is based on a type. The type consists of *aspects* and *variables*. Aspects provide a mechanism to model data of assets. They gather related data in logical groups. For example, an asset “pump” has an aspect "energy consumption". Further on, an aspect contains a set of concrete *variables* (datapoints). For instance, the aspect "energy consumption" contains the variables: "power", "current", "voltage" etc.

Studying the data model of MindSphere a proposed mapping of semantic constructs from iotschema.org (that aligned to W3C WoT Thing Description) to constructs of MindSphere data model is derived.

TABLE 8 AN OVERVIEW OF MAPPING MINDSPHERE MODEL TO IOTSCHEMA.ORG

MindSphere asset model	**iotschema.org**
AssetType	Capability that is the composition of other Capabilities or a Thing Description template
AspectType	Capability
Asset	Thing Description
Aspect	Thing Description’s part marked up with a Capability
Variable	Thing Description’s Event, Property or Action

Table 8 shows the mapping of main concepts from MindSphere asset model and iotschema.org. Apart from this, metadata of each concept should be mapped too. For example, name and description of AspectType should be mapped to the name and description of a corresponding Capability of iotschema.org. Units and data types should be mapped from iotschema.org to the new model, which is created in MindSphere too.

Code 13 shows an example of Thing Description (TD) for a microphone device used in a wind turbine (Use Case 1). TD has been semantically enriched with the mark up from iotschema.org. This TD is exposed by IIoT Gateway and available in Local and Global Thing Description Directory. The same TD has been used to create an asset model in MindSphere. Code 14 shows the equivalent model of the microphone, represented in MindSphere asset model. As we see the model contains semantic constructs from TD, and as such ensures the semantic interoperability between Cloud and Field/Edge level.

```
{
    "@context":[
```

[38] https://siemens.mindsphere.io/en

[39] https://documentation.mindsphere.io/resources/pdf/asset-manager-en.pdf

```
    "http://www.w3.org/ns/td",
    {
      "iot":"http://iotschema.org/"
    }
  ],
  "id":"urn:dev:wot:com:example:servient:thing5",
  "name":"SEMIoTICS-Microphone",
  "description":"Microphone used in a wind turbine",
  "securityDefinitions":{
    "psk_sc":{
      "scheme":"psk"
    }
  },
  "security":[
    "psk_sc"
  ],
  "@type":[
    "iot:Microphone"
  ],
  "actions":{
    "on":{
      "@type":"iot:iot:startRecording",
      "description":"Starts recording the audio.",
      "type":"boolean",
      "forms":[
        {
          "href":"http://thing-5.iiot-gtw.org:5683/mic/on"
        }
      ]
    },
    "off":{
      "@type":"iot:iot:stopRecording",
      "description":"Stops recording the audio.",
      "type":"boolean",
      "forms":[
        {
          "href":"http://thing-5.iiot-gtw.org:5683/mic/off"
        }
      ]
    }
  }
}
```

CODE 13 EXAMPLE OF TD FOR A MICROPHONE

```
{
    "assettype":{
      "name":"SEMIoTICS-Wind-Turbine-Monitoring-Type",
      "description":"SEMIoTICS monitoring system in a wind turbine.",
      "scope":"private",
      "aspects":[
        {
          "name":"Microphone",
          "aspectTypeId":"iot:Microphone"
        }
      ]
    },
    "aspecttype":{
      "name":"iot:Microphone",
      "category":"dynamic",
      "scope":"private",
      "description":"A capability for an Internet Protocol (IP) microphone.",
      "variables":[
        {
          "name":"iot:startRecording",
          "dataType":"boolean",
          "unit":"N/A"
        },
        {
          "name":"iot:stopRecording",
          "dataType":"boolean",
          "unit":"N/A"
        }
      ]
    }  "asset":{
      "name":"Wind-Turbine-Monitoring-5",
      "externalId":null,
      "description":"SEMIoTICS monitoring system in a wind turbine no.5.",
      "aspects":[
        "MicrophoneAspects#5",
        {
          "name":"iot:Microphone",
          "variables":[
            {
              "name":"iot:startRecording",
              "value":"%(.value)"
            },
            {
              "name":"iot:stopRecording",
              "value":"%(.value)"
            }
          ]
        }
      ]
    }
  }
```

CODE 14 EXAMPLE OF MINDSPHERE ASSET MODEL FOR A MICROPHONE

6.4 Interaction with openHAB

OpenHAB is an Open Source IoT platform, which mainly targets smart home and smart buildings environments, and is leveraged in SEMIoTICS in the context of the Generic IoT use-case. Said platform supports many off-the-shelf Smart Sensors already present in building environments, and can be extended through “add-ons” that handle the interaction with external sensors, data storage backends and chart libraries for sensor value visualization. Furthermore, openHAB supports a scripting language to implement automation “if-this-then-that” scenarios. openHAB relies on a JSON-LD message bus[40] and offers a REST API powered by the Jetty HTTP server. The RESTful service offered by openHAB, gives access to Things, Channels and Items, represented via the Eclipse Smarthome data model. In more detail:

- **Things** are entities that can be physically added to a system. They may provide more than one function (for example, a Z-Wave multi-sensor may provide a motion detector and measure room temperature). Things do not have to be physical devices; they can also represent a web service or any other manageable source of information and functionality. From a user perspective, they are relevant for the setup and configuration process, but not for the operation. Things can have configuration properties, which can be optional or mandatory. Such properties can be basic information like an IP address, an access token for a web service or a device specific configuration that alters its behavior. Things expose their capabilities through Channels.
- **Channels** represent the different functions the Thing provides. Where the Thing is the physical entity or source of information, the Channel is a concrete function from this Thing. A physical light bulb might have a color temperature Channel and a color Channel, both providing functionality of the one light bulb Thing to the system. For sources of information, the Thing might be the local weather with information from a web service with different Channels like temperature, pressure and humidity. Channels are linked to Items, where such links are the glue between the virtual and the physical layer. Once such a link is established, a Thing reacts to events sent for an item that is linked to one of its Channels. Likewise, it actively sends out events for Items linked to its Channels. Whether an installation takes advantage of a capability reflected by a Channel depends on whether it has been configured to do so. When you configure your system, you do not necessarily have to use every capability offered by a Thing. You can find out what Channels are available for a Thing by looking at the documentation of the Thing's Binding.
- **Bindings** can be thought of as software adapters, making Things available to the system. They are add-ons that provide a way to link Items to physical devices. They also abstract away the specific communications requirements of that device so that it may be treated more generically by the framework.

[40] https://www.eclipse.org/smarthome/rest/index.html

- **Items** represent capabilities that can be used by applications, either in user interfaces or in automation logic. Items have a State which may store sensor values and they may receive commands (e.g., for actuation purposes).

The openHAB support of REST and JSON-LD protocols offers a simple integration path with the SEMIoTICS protocol suite without the need of any additional parsers. This is simply achieved with openHAB Transformation Services[41] that map the SEMIoTICS WoT TDs to the openHAB's Eclipse Smarthome Data Model. An example of a SEMIoTICS thing shown at the openHAB Dashboard is shown in Figure 18, Code 15.

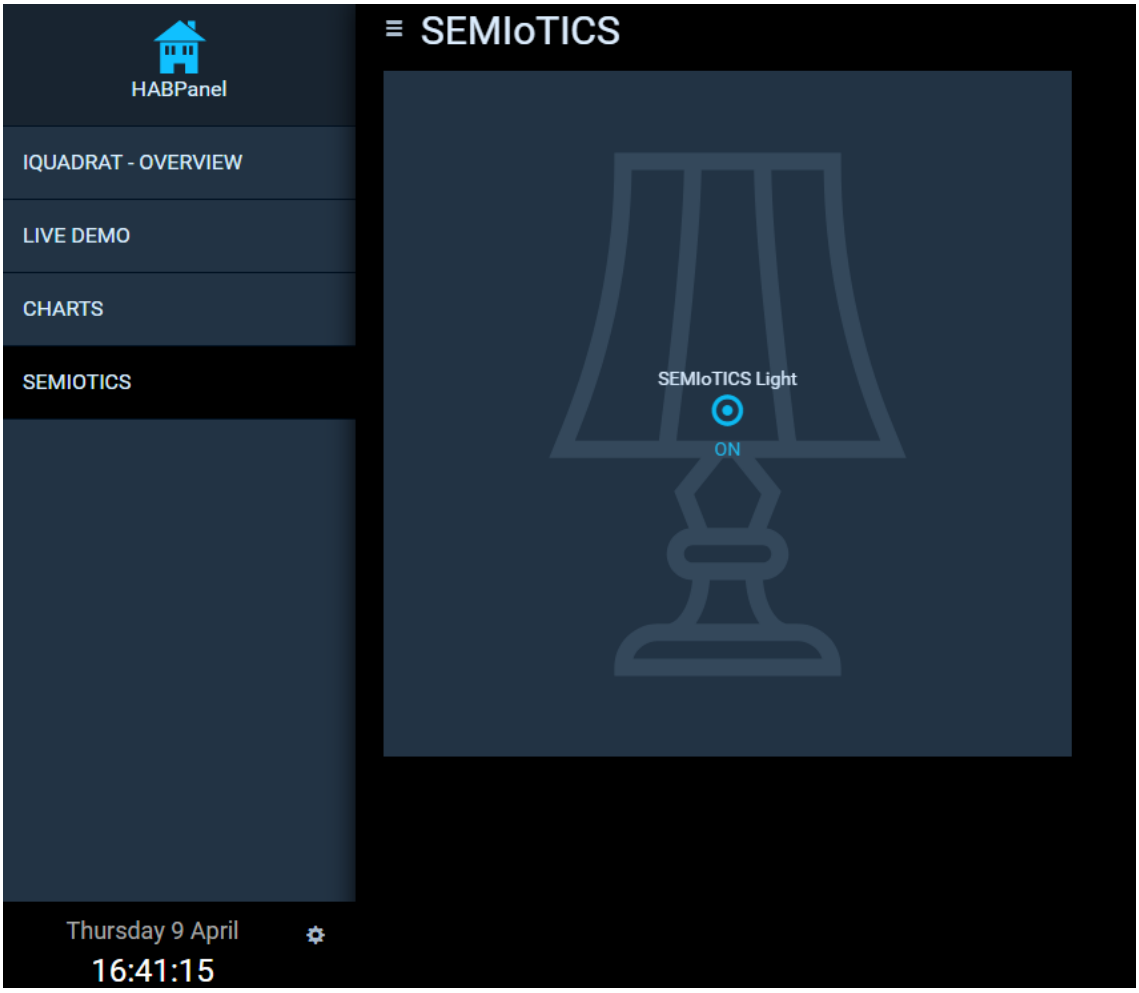


FIGURE 18 SEMIOTICS THING (I.E., CEILING LAMP) CONNECTED TO OPENHAB DASHBOARD

```
{
  "@context": [ "http://www.w3.org/ns/td",
               {"iot": "http://iotschema.org/"} ],
  "@type" : [
```

[41] https://www.openHAB.org/docs/configuration/transformations.html

```
    "Thing", "iot:LightControl", "iot:BinarySwitchControl"
],
"id": "urn:dev:wot:lamp",
"name": "WirelessLamp",
"description" : "WirelessLamp uses JSON-LD 1.1 serialization",
"securityDefinitions": {
    "basic_sc": {"scheme": "basic", "in":"header"}
},
"security": ["basic_sc"],
"properties": {
  "status" : {
    "@type" : "iot:SwitchStatus",
    "type": "string",
    "forms": [{
       "href": "mqtt://192.168.1.11:1883/house/lamp/status",
       "mediaType": "application/json"}]
  }
},
"actions": {
   "toggle" : {
   "@type" : "iot:ToggleAction",
   "forms": [{
      "href": "mqtt://192.168.1.11:1883/house/lamp/toggle",
      "mediaType": "application/json"}]
   }
},
"events":{
  "overheating":{
    "@type" : "iot:TemperatureAlarm",
    "data": {"type": "string"},
    "forms": [{
        "href": "mqtt://192.168.1.11:1883/house/lamp/oh",
        "subprotocol": "longpoll"
    }]
  }
}
```

CODE 15 SEMIOTICS THING (I.E., CEILING LAMP) CONNECTED TO OPENHAB

7 END-TO-END INTEROPERABILITY VERIFICATION MECHANISMS

This chapter analyses the verification mechanisms that ensure the end-to-end interoperability of a currently composed setting. This process is materialized via the translation of Recipes into SPDI Patterns. The tool facilitates the design of workflows and guarantees the composition properties of the system. The development of the SPDI pattern language and the associated reasoning mechanisms, a set of initial interoperability patterns, along with their interplay and integration with IoT orchestrations' definitions via Recipes, are detailed in D4.1 and its follow up D4.8.

7.1 Translation of Recipes into SPDI Patterns

In order to ensure interoperability from the application definition all the way through to the execution at runtime, we have implemented four levels of abstraction and accordingly three steps of transformation between them. These steps are indicated in Figure 19 below.

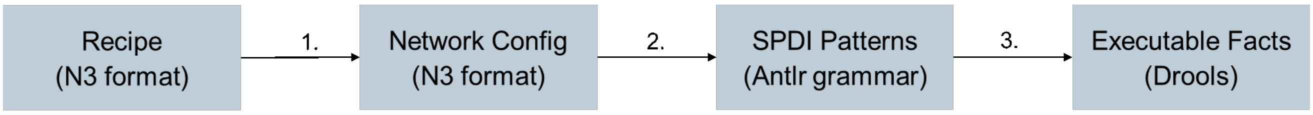


FIGURE 19 TRANSLATIONS FROM RECIPES TO EXECUTABLE RULES

Recipes are designed in the Recipe Cooker tool and serialized in the N3 language. Then, we have implemented various N3-based rules that can transform the Recipe into a network configuration. An overview of these 2 models (Recipe and Network model) and their transformation is shown in Figure 20. They are defined as triples in the RDF format and serialized in the N3 format.

On the left side of Figure 20 , the model to define abstract IoT compositions as *recipes* is shown. This model is based on our previous work [39], [40], [41]. A recipe is a template for a workflow of interactions between multiple components, or *ingredients*. When a recipe is instantiated, ingredients are replaced with concrete components, which we call IoT *offerings*. An offering is a concrete service of an IoT device or platform that has inputs, outputs and a semantic category. Here, the recipe model is extended to allow the definition of application-level Quality of Service (QoS) constraints, which are then translated to Software Defined Networking (SDN) QoS constraints. Therefore, the concept *QoSConstraint* has been associated with an interaction of the recipe.

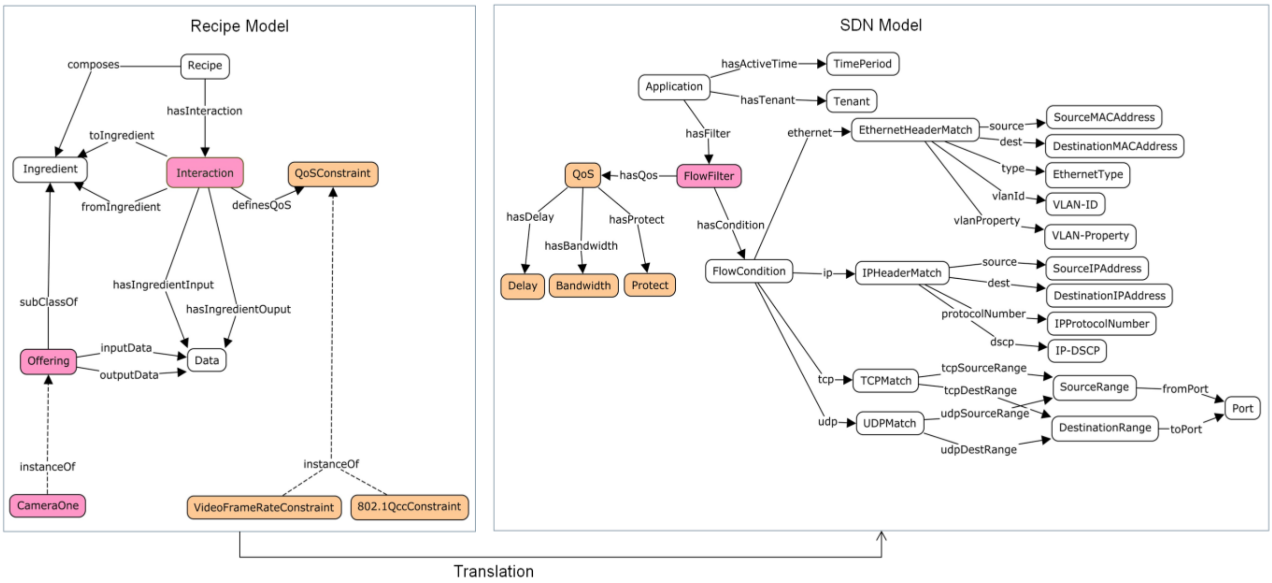


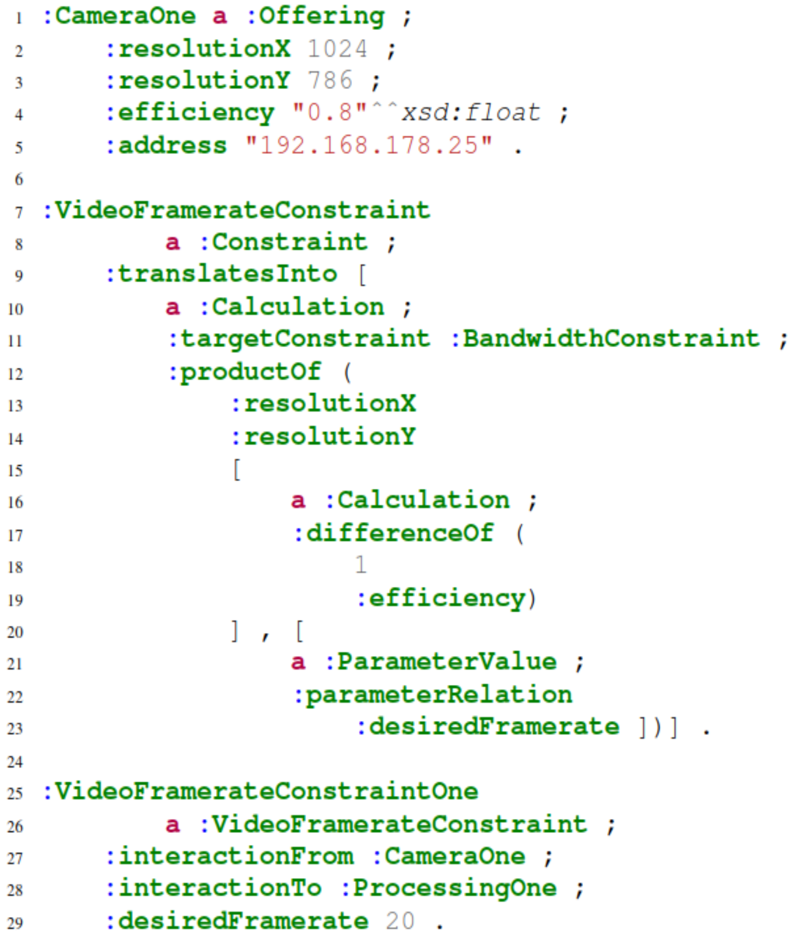
FIGURE 20 TRANSFORMATION FROM RECIPE TO NETWORK CONFIGURATION [42]

SDN enables the enforcement and validation of QoS constraints on a service composition's network communication. To take advantage of these tools, we need to model its parameters in a manner compatible with a service model. We have chosen to model SDN concepts in a semantic fashion, for simplified integration with semantic service composition systems. Our SDN model is depicted on the right side of Figure 20. The design of this model is inspired by the data structures used by the northbound interfaces of SDN controllers, such as the ones defined by [43]. The central component of this model is the application. When the model is instantiated, this is the entry point to the definition of a specific SDN configuration. Associated with the application is a time period during which it is valid and a tenant who represents the user of the network. Every application is associated with an interface that comprises the network node on which it runs as well as the physical port it is attached to.

A key concept associated with the application is the flow filter. Here, a destination (pointing to a specific interface), filter conditions, and QoS requirements are defined. As QoS requirements, we have added delay, bandwidth and protect constraints. This modelling is non-exhaustive, and it depends on the functionality available at the store and more constraints can be added. The delay constraint describes a maximum allowed latency between two endpoints, while the bandwidth constraint specifies a minimum guaranteed bandwidth between two endpoints. The protect constraint provides a mean to specify redundant packet transmission, which facilitates sending the same packet over different network links to improve the connection reliability.

These constraints are applied to flows that match the conditions attached to a single filter. Currently, we included flow conditions to check for matches on the ethernet, IP, TCP and UDP protocols. Further protocols can be added, e.g., based on ARP addresses or ICMP packets. As an example, to specify the maximum delay for a connection between a sensor and an actuator, we can instantiate a flow filter with a *delay* QoS and a flow condition consisting of an IP header match with a source IP address of the sensor, and the destination IP address that of the actuator. Then, the maximum delay constraint would be applied to all packets being sent from the sensor to the actuator.

Figure 21 shows an example of N3 triple encodings that represent a camera as an offering to be used as part of a recipe. In addition to the camera definition, a video frame-rate constraint is given. In this example of an application-specific constraint, the *frame rate (f)* for a camera stream specifies the minimum frames/second the network needs to be able to transmit. Since we define this constraint on the application level, information on the camera's data format and the resolution of the video stream is available to us. If the video format's efficiency is $e \in [-\infty, 1]$ and the video's resolution is $x \times y$, we can infer a minimum bandwidth with the calculation *bw = (1−e)∗x∗y∗f*. The bandwidth constraint derived from this equation can then be configured on the network. If the application then changes (for example, switching to a video camera with a less effective video format), the application-level constraint can be re-evaluated and changes can be applied to the network.



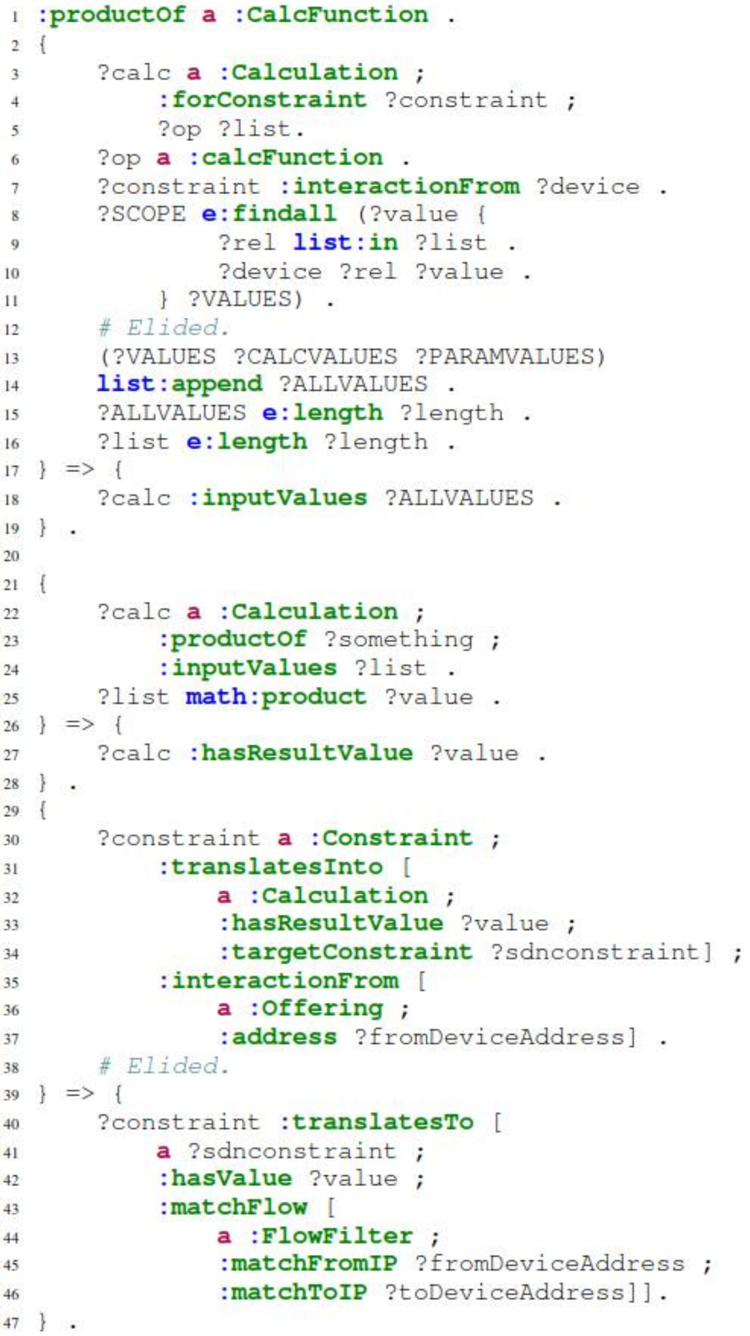
```
:CameraOne a :Offering ;
    :resolutionX 1024 ;
    :resolutionY 786 ;
    :efficiency "0.8"^^xsd:float ;
    :address "192.168.178.25" .

:VideoFramerateConstraint
        a :Constraint ;
    :translatesInto [
        a :Calculation ;
        :targetConstraint :BandwidthConstraint ;
        :productOf (
            :resolutionX
            :resolutionY
            [
                a :Calculation ;
                :differenceOf (
                    1
                    :efficiency)
            ] , [
                a :ParameterValue ;
                :parameterRelation
                    :desiredFramerate ])] .

:VideoFramerateConstraintOne
        a :VideoFramerateConstraint ;
    :interactionFrom :CameraOne ;
    :interactionTo :ProcessingOne ;
    :desiredFramerate 20 .
```

FIGURE 21 DEVICE AND CONSTRAINT DEFINITION IN N3 FORMAT

Figure 22 contains an excerpt of the translation implementation. The implementation takes the form of rules that are expressed as implications. When the premise of the rule holds, the conclusion of the rule is inserted into the triple store, with all existential variables replaced with the bindings from the rule's premise. Line 1 defines the *productOf* property as a calculation function that is resolved by the rule system. The rule in lines 2-19 results in the recursive calculation of calculation values. We do this by iterating over all the values in the argument list of the calculation relation (for example, *productOf*) and attaching the calculated values to the calculation. The argument list can contain three types of values: Literals, which are used as-is, device properties, which are resolved from the device the constraint is applied to, and parameters, which are resolved from the constraint itself. When all input values for a calculation are available (line 16), they are appended into a single list and attached to the calculation node. Then, the calculation rule on lines 21 to 28 fires and computes the result using the reasoner's built-in *math:product* predicate. This rule is replicated for other calculation instructions, such as *differenceOf* or *sumOf* (not included here). When a result value for the root of the calculation has been computed, the rule in lines 30-47 generates the target constraint with the correct value. Additionally, flow filter information from the device is used to generate a flow filter.



```
:productOf a :CalcFunction .
{
    ?calc a :Calculation ;
        :forConstraint ?constraint ;
        ?op ?list.
    ?op a :calcFunction .
    ?constraint :interactionFrom ?device .
    ?SCOPE e:findall (?value {
            ?rel list:in ?list .
            ?device ?rel ?value .
        } ?VALUES) .
    # Elided.
    (?VALUES ?CALCVALUES ?PARAMVALUES)
    list:append ?ALLVALUES .
    ?ALLVALUES e:length ?length .
    ?list e:length ?length .
} => {
    ?calc :inputValues ?ALLVALUES .
} .

{
    ?calc a :Calculation ;
        :productOf ?something ;
        :inputValues ?list .
    ?list math:product ?value .
} => {
    ?calc :hasResultValue ?value .
} .
{
    ?constraint a :Constraint ;
        :translatesInto [
            a :Calculation ;
            :hasResultValue ?value ;
            :targetConstraint ?sdnconstraint] ;
        :interactionFrom [
            a :Offering ;
            :address ?fromDeviceAddress] .
    # Elided.
} => {
    ?constraint :translatesTo [
        a ?sdnconstraint ;
        :hasValue ?value ;
        :matchFlow [
            a :FlowFilter ;
            :matchFromIP ?fromDeviceAddress ;
            :matchToIP ?toDeviceAddress]].
} .
```

FIGURE 22 TRANSLATION RULES FOR A CAMERA FRAME-RATE CONSTRAINT

Having the above-described translation rules available, we have covered the translation step (1.) in Figure 22. To translate the network configuration and details into SPDI patterns, we have developed a Python script. It converts the network configuration defined in N3 into the Extended Backus-Naur Form (EBNF) grammar defined in the Antlr[42] format. An example output that represents such a SPDI pattern in context of the video camera example above is shown in Code 16.

```
Placeholder(https://iotscheme.siemens.com/#ProcessingOne),
Placeholder(https://iotscheme.siemens.com/#CameraOne),
Link(link-0, https://iotscheme.siemens.com/#CameraOne,
https://iotscheme.siemens.com/#ProcessingOne),
Property(property-0, confirmed, qos, in_transit, Verification(monitoring, interface),
link-0),
Property(property-1, confirmed, qos, in_transit, Verification(monitoring, interface),
link-0)
```

CODE 16 SPDI PATTERN

7.2 Interoperability Patterns

7.2.1 INTRODUCTION

As mentioned above, the interoperability is the ability of different information technology systems and heterogeneous services to communicate, exchange data, and use the information that has been exchanged. Generally, the aspects of interoperability comprise technical, syntactical, semantic, and organizational issues, usually referenced as interoperability layers [44]. Similarly, four levels of interoperability are included in the SEMIoTICS framework: technological, syntactic, semantic and organizational interoperability (see Deliverable D4.8, as well as deliverable D3.10 presenting interoperability aspects from a network perspective). Each layer is dependent on another layer, i.e. syntactic interoperability is only possible if technological interoperability exists; the semantic interoperability will be the next step when syntactic interoperability is already implemented and so on (Figure 23 below). Although, it is worthwhile to note that the organizational interoperability is the most complex interoperability type. These complexities are associated not only with formal agreements on collaboration but also with practical approaches to organizational interoperability [45].

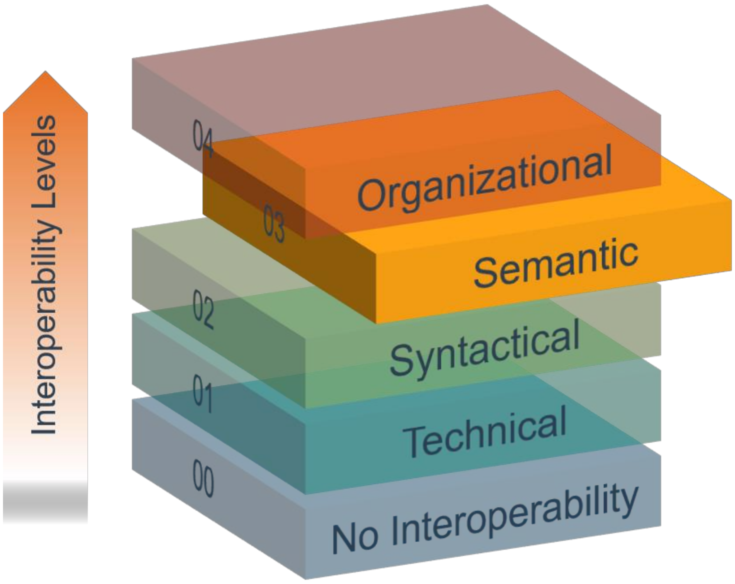


FIGURE 23 PATTERN DEFINITION – INTEROPERABILITY LAYERS IN SEMIOTICS FRAMEWORK

[42] https://www.antlr.org/

7.2.2 SEMANTIC INTEROPERABILITY PATTERN DEFINITION

To fully exploit the above pattern-driven features and provide an E2E provision for interoperability throughout the SEMIoTICS framework, a set of Interoperability-focused patterns must be defined. The full set of Interoperability patterns are presented in Deliverable D3.10 “Network-level Semantic Interoperability (final)” and D4.8 “SEMIoTICS SPDI Patterns (final)”. This deliverable focuses on the semantic category of Interoperability patterns; hence, the pattern definition and the pattern specification rule are analysed in this and in the next subsection accordingly, only from the semantic perspective.

Particularly, if it is considered that:

- C := the set of all instantiated components
- MDL := A set of semantic models
- C1,C2 ⊆ C , where C1 ≠ C2
- Ci_MDL ⊆ MDL := semantic models used by Ci
- SeMD := Semantic Mediator

then three lemmas could be defined as following

Lemma 1: If C1, C2 are syntactically interoperable and C1_MDL ∩ C2_MDL ≠ ∅ then C1 and C2 are directly semantically interoperable

Lemma 2: If C1, C2 are syntactically interoperable and are both directly semantically interoperable with SeMD, then C1, C2 are indirectly semantically interoperable

Lemma 3: If C1, C2 are directly or indirectly semantically interoperable, then C1, C2 are semantically interoperable.

7.2.3 SEMANTIC INTEROPERABILITY PATTERN SPECIFICATION RULE

Taking to account the above analysis, in the previous subsection, the workflow-based definition of semantic interoperability in the fundamental scenario of two IoT activities A1 and A2 interacting with each other is as follows:

1. **WF “semantic-interoperability”**
2. Placeholder (A1, (PlaceholderActivity, PlaceholderDescription))
3. Placeholder (A2, (PlaceholderActivity, PlaceholderDescription))
4. Placeholder (SeMD, (PlaceholderActivity,”Semantic Broker”))
5. Link (L1, A1, A2)
6. Link (L2, A1, SeMD)
7. Link (L3, A2, SeMD)
8. Property (conn01, L1, required, (pattern-based, pattern),”_syntactic-interoperability” , in_transit_ ∨ in_processing)
9. Property (conn02, L2, required, (pattern-based, pattern),”_syntactic-interoperability” , in_transit_ ∨ in_processing)
10. Property (conn03, L3, required, (pattern-based, pattern),”_syntactic-interoperability” , in_transit_ ∨ in_processing)
11. Property (conn1, L1, required, (pattern-based, pattern),”_semantic-interoperability” , in_processing)
12. Property (conn2, L2, required, (pattern-based, pattern),”_semantic -interoperability” , in_processing)
13. Property (conn3, L3, required, (pattern-based, pattern),”_semantic -interoperability” , in_processing)
14. Property (conn4, “_semantic-interoperability”, required, (pattern-based, PR1),” “_semantic-interoperability”, end_to_end)
15. **Pattern rule: (PR1: (conn01,conn1) || (conn02,conn2,conn03,conn3) ➔ conn4)**

The corresponding machine-processable Drool rule format of the said semantic interoperability rule can be defined in the below code (Code 17):

```
rule "Sequence Semantic Interoperability Verification"
when
        Placeholder($pA:=placeholderid)
        Property ($pA:=subject, category=="semantic", $prvaluein1:=input_value,
            $prvalueout1:=output_value, satisfied==true)
        Placeholder($pB:=placeholderid)
        Property ($pB:=subject, category=="semantic", $prvaluein2:=input_value,
            $prvalueout2:=out_value, satisfied==true)
        Sequence($sId:=placeholderid, $pA:=placeholdera, $pB:=placeholderb)
        $PR: Property ($sId:=subject, category=="semantic",
$prvalueout1==$prvaluein2,    satisfied==false)
then
        modify($PR){satisfied=true, input_value=$prvaluein1,
output_value=$prvalueout2};
end
```

CODE 17 SEMANTIC INTEROPERABILITY VERIFICATION DROOL RULE

8 SEMANTIC INTEROPERABILITY MECHANISMS - IMPLEMENTATION

This chapter describes the semantic interoperability techniques for resolving semantic differences and their implementation to support the usage scenarios of SEMIoTICS. To achieve this, the SEMIoTICS framework has integrated several different software components with the BSV component as the core of this procedure. In this context, the following section illustrates the role of BSV component in SEMIoTICS and continues with the initial and the final implementation of the above procedure.

8.1 Role of Backend Semantic Validator Component

As mentioned, this deliverable focuses on mechanisms to validate semantic interoperability using interoperability conditions defined in the pertinent SEMIoTICS patterns (as defined in Task 4.1 and documented in D4.8), ensuring that data flow is possible between smart objects in the SEMIoTICS architecture components. BSV is the main component responsible for this, as its role is to provide:

1. Validation mechanisms to ensure semantic interoperability,
2. Connection with external IoT platforms to enable interoperability between these targeted external IoT enabling platforms and SEMIoTICS and
3. Adaptability taking to account the interoperability of devices that are used in SEMIoTICS.

Based on the above, the following subsections explain, in detail, the proposed techniques and the procedure of said BSV roles, with subsection 8.2 presenting the initial proof-of-concept implementation and subsection 8.3 focusing on the final prototype implementation.

8.2 Proof-of-concept Implementation

The first deliverable of SEMIoTICS Task 4.4 (D4.4 Semantic interoperability mechanisms for IoT (first draft)) presented the initial development of data transformation techniques and validation mechanisms to ensure end-to-end semantic interoperability. Specifically, D4.4 outlines the SEMIoTICS architecture elements that are responsible for resolving semantic differences and provide a short analysis of the implementation of them based on the motivating scenario, as already analyzed in subsection 2.2. The components of the SEMIoTICS architecture which are involved in said semantic interoperability mechanisms are shown in Figure 24. These include the Backend Semantic Validator, the Recipe Cooker, and the Thing Directory from the backend layer, as well as the Semantic API & Protocol Binding, the GW Semantic Mediator, and the Local Thing Directory from the field layer.

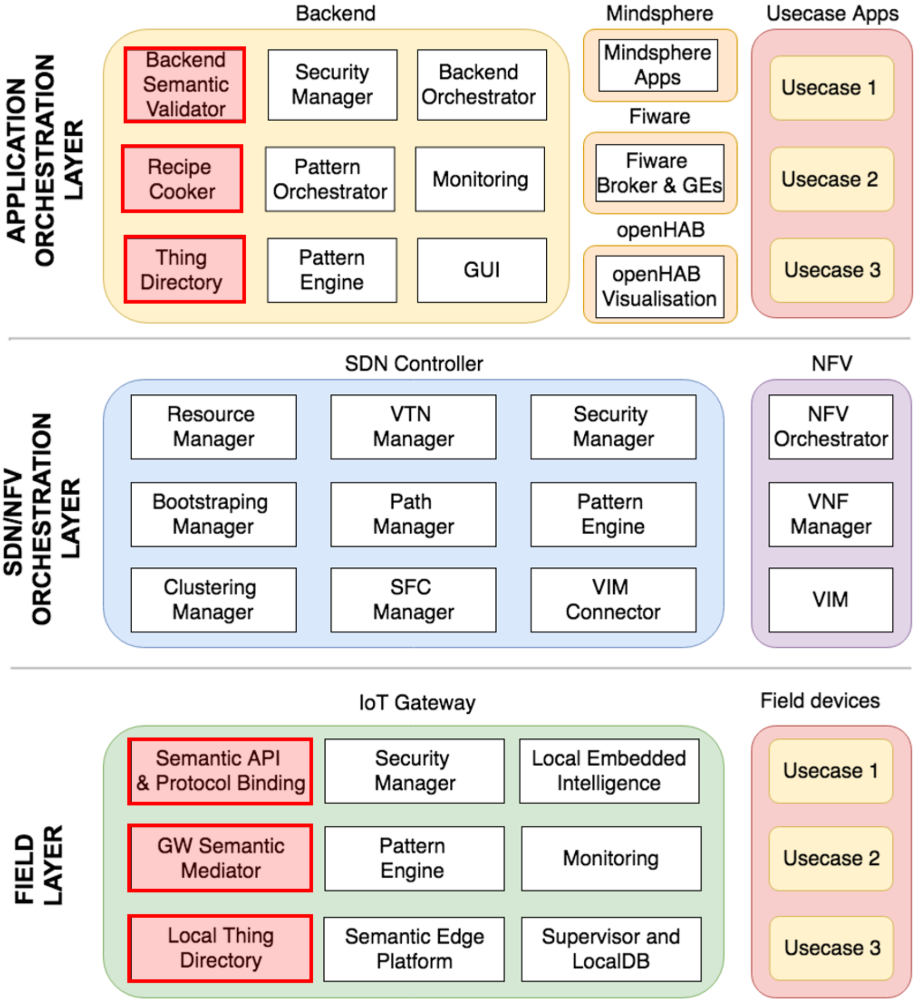


FIGURE 24 SEMIOTICS ARCHITECTURE COMPONENTS -SEMANTIC INTEROPERABILITY MECHANISMS BASED ON MOTIVATING SCENARIO (SUBSECTION 2.2)

In Figure 25 a sequence diagram depicts the procedure of the semantic interoperability mechanisms between the application orchestrator layer and the field layer. As stated in subsection 2.2 , the aim is the connection between two Things (Sensor and Actuator), by an IoT application which sends a request in the backend. This motivating scenario can be applicable in SEMIoTICS UCs, using any other sensor (e.g. for pressure, humidity, light) that aims to interact with an actuator.

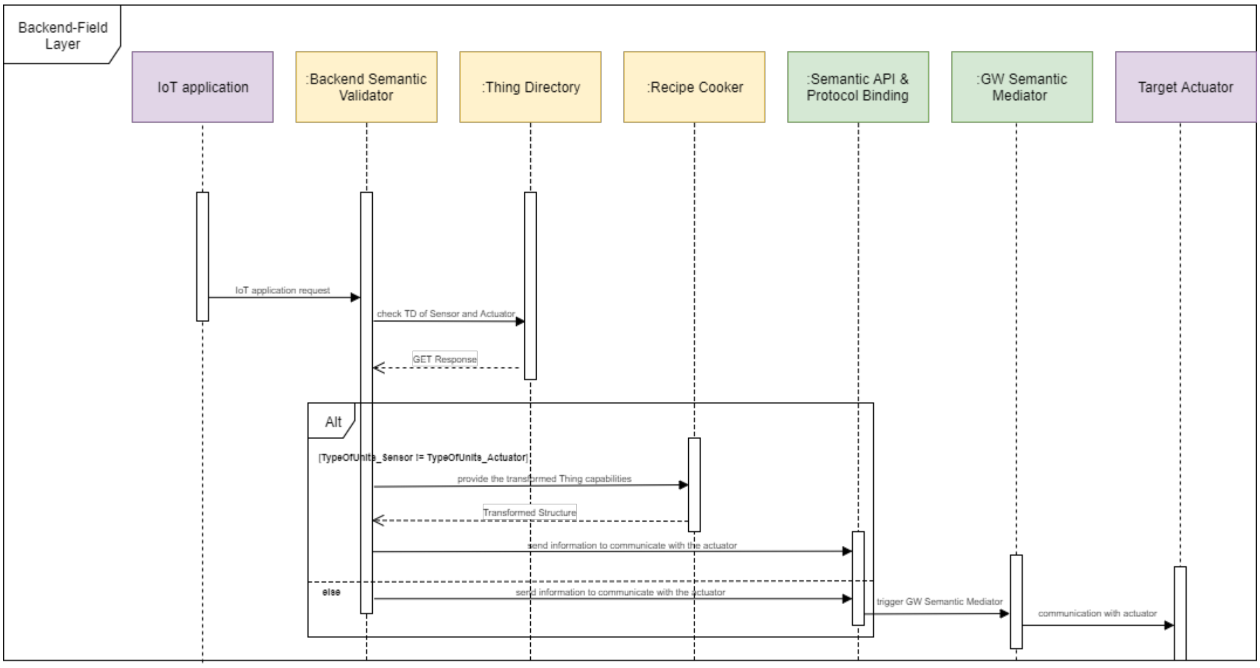

Backend-Field Layer
IoT application
:Backend Semantic Validator
:Thing Directory
:Recipe Cooker
:Semantic API & Protocol Binding
:GW Semantic Mediator
Target Actuator
IoT application request
check TD of Sensor and Actuator
GET Response
Alt
[TypeOfUnits_Sensor != TypeOfUnits_Actuator]
provide the transformed Thing capabilities
Transformed Structure
send information to communicate with the actuator
else
send information to communicate with the actuator
trigger GW Semantic Mediator
communication with actuator

FIGURE 25 SEQUENCE DIAGRAM FOR SEMANTIC INTEROPERABILITY MECHANISMS BASED ON MOTIVATING SCENARIO (SUBSECTION 2.2)

In fact, the BSV receives the request from the IoT application, using the grpc[43] framework and it takes information from the Thing Directory component; the Thingweb Directory is used for the implementation of Thing Directory component and the interaction is achieved by the HTTP endpoint that provides an HTML client to register and discover TDs. This client accesses a REST API to manage TDs that complies to the IETF Resource Directory specification. Registration is done by POSTing and discovery can be performed by using a SPARQL graph pattern as a query parameter (GET).

The Recipe Cooker (based on the Node-RED framework) is responsible to harmonize the semantic model capabilities with the registration of extra Adaptor Nodes in the recipe, based on the Temperature Control Example[44] (Figure 26). In case of an advanced scenario that will require to tackle conflicts for multiple parameters, e.g. interacting Things used different data transformation techniques for type (string, float) and units measurement (Celsius scale, Fahrenheit scale), then the corresponding Adaptor Nodes can be applied inline, using the output of the first as the input of the next. The main advantage of this approach is the reusability of Adaptor Nodes in order to address the requirements of any usage scenarios and applications and the requirements for the infrastructure.

For the final stage, as the sequence diagram depicts, the SAPB component and the GWSM component from the backend layer are responsible to transmit the information to the target actuator.

[43] https://grpc.io/

[44] https://github.com/iot-schema-collab/iotschema-node-red/blob/master/example-doc.md

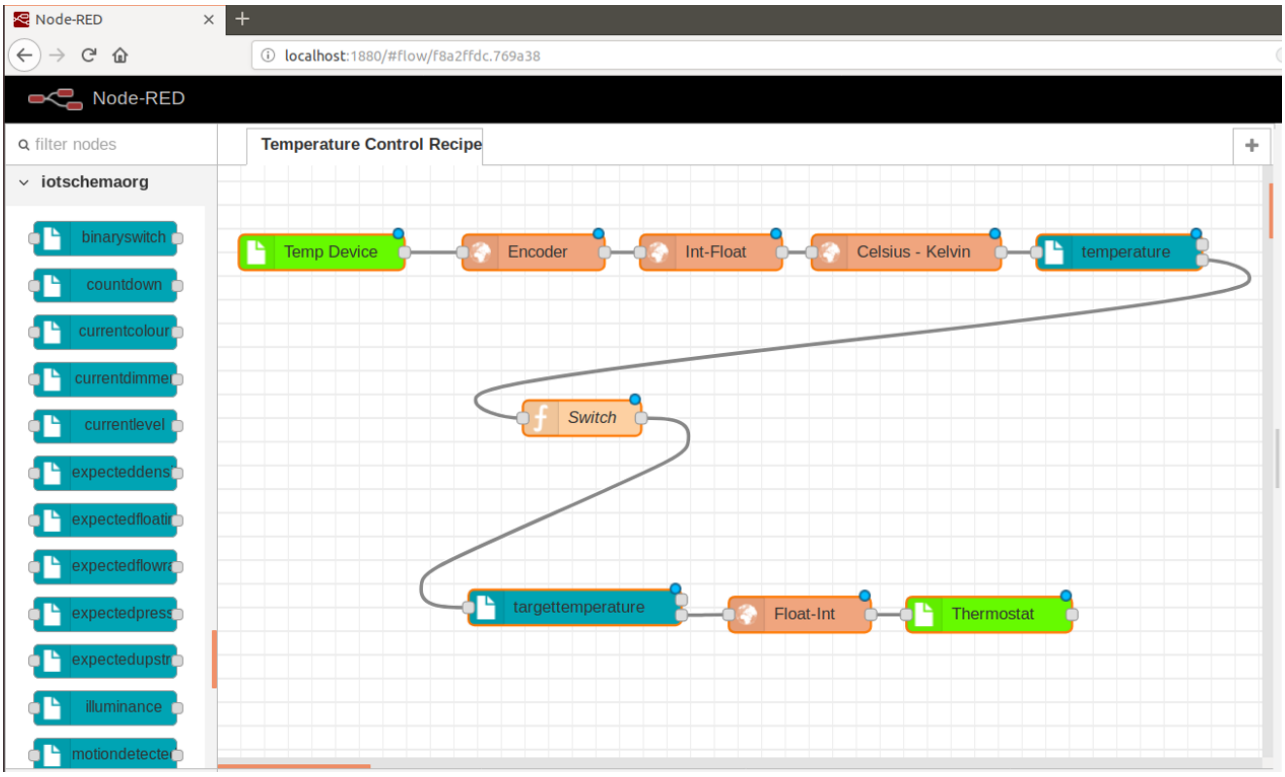


FIGURE 26 TEMPERATURE CONTROL EXAMPLE (GITHUB.COM)

The next section describes the latest development of data transformation techniques and validation mechanisms to ensure that data flow is possible between smart objects of SEMIoTICS UCs, based on the role of BSV component; these mechanisms should rely on the inclusion of interoperability conditions in the SEMIoTICS patterns.

8.3 Final Implementation

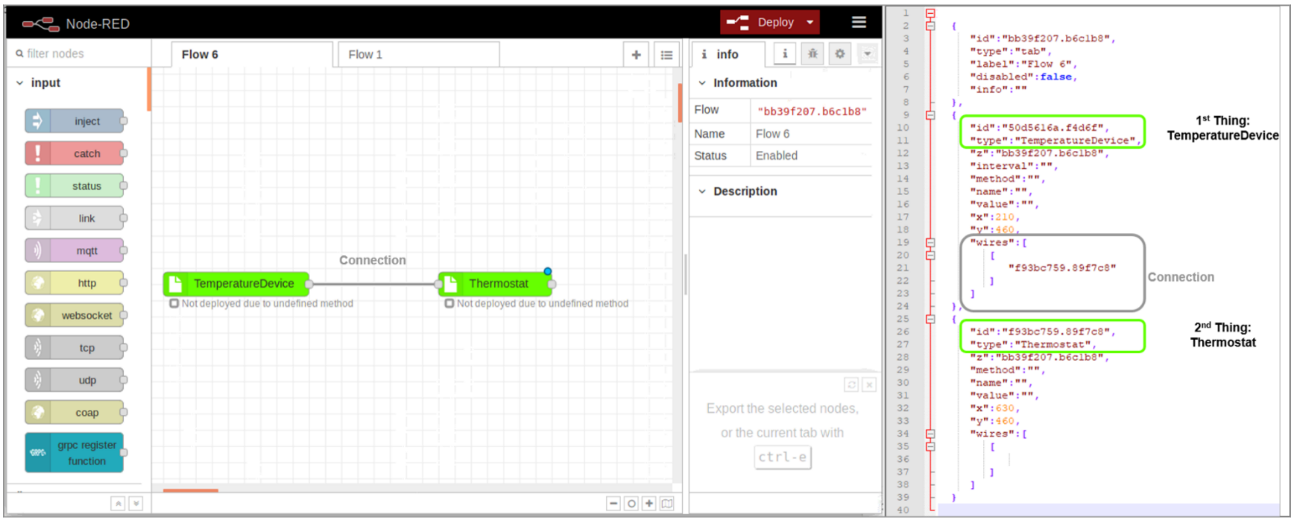
8.3.1 VALIDATION MECHANISMS – IMPLEMENTATION

One of the main aims of BSV component is to tackle the semantic interoperability issues that arise in the SEMIoTICS framework, at the application orchestration layer. The BSV can receive a request for interaction between two Things, which are described with two different TDs (based on W3C Thing Descriptions that are serialized to JSON-LD standard format), respectively. The functionality of this component consists of:

- Searching for the necessary Thing models in Thing Directory component to detect any potential semantic conflicts between the interacting domains.
- Connecting with Recipe Cooker to resolve these semantic conflicts using the Adaptor Nodes that configure an Interaction Pattern in accordance with the application's requirements.

From a technical point of view, the first step has been implemented in Cycle 1 of the project’s implementation (as documented in D4.4), while during Cycle 2 (documented herein), the service requests (POST/GET) development technology was changed in order to be compatible with the other components. Specifically, the grpc method (see Section 8.2) was replaced by the RESTful API to provide services for receiving data in a convenient format, creating new data, updating data and deleting data between the interaction of SEMIoTICS architecture components.

The second step focuses on resolving any possible semantic conflicts between the interacting different Things, using or creating the corresponding Adaptor Nodes in Recipe Cooker. A recipe is instantiated in the Recipe Cooker, which is a flow of interactions between Things (i.e. Sensor:TemperatureDevice, Actuator:Thermostat) with their own respective Thing Description. Practically, this flow is a JSON file that includes all the above information about the connectivity between Things (ingredients) (see Figure 27).



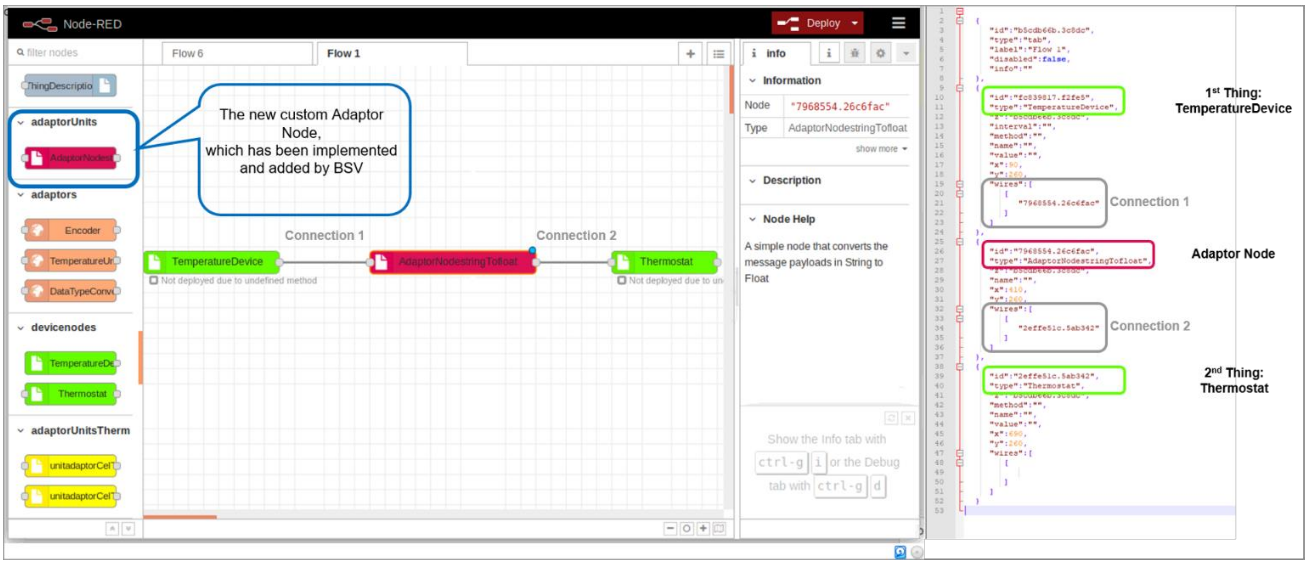
```
    {
        "id":"bb39f207.b6c1b8",
        "type":"tab",
        "label":"Flow 6",
        "disabled":false,
        "info":""
    },
    {
        "id":"50d5616a.f4d6f",
        "type":"TemperatureDevice",
        "z":"bb39f207.b6c1b8",
        "interval":"",
        "method":"",
        "name":"",
        "value":"",
        "x":210,
        "y":460,
        "wires":[
            [
                "f93bc759.89f7c8"
            ]
        ]
    },
    {
        "id":"f93bc759.89f7c8",
        "type":"Thermostat",
        "z":"bb39f207.b6c1b8",
        "method":"",
        "name":"",
        "value":"",
        "x":630,
        "y":460,
        "wires":[
            [

            ]
        ]
    }
```

FIGURE 27 RECIPE EXAMPLE BEFORE SEMANTIC VALIDATION

Thus, the above functionality of the BSV component is summarized in the following phases:

1. A Post Service Request (**validateRecipeFlow**) has been developed in order to Recipe Cooker send the JSON/flow recipe to BSV (Figure 27). This request aims to trigger BSV to check for any interoperability conflicts between the two Things of this recipe.
2. The BSV component interacts with the Thing Directory component to ensure that these specific Things have already been registered in order to receive information on their TDs. This is a required step, otherwise, the BSV cannot resolve semantic differences and ensure that data flow is possible between them.
3. The BSV parses the TDs to discover for the semantic interoperability between the connected Things. In this phase, there are two possible cases:
    a. Interacting Things used the same data transformation techniques (i.e. use the same units of measurements)
    b. Interacting Things used the different data transformation techniques (i.e. the first Thing uses **string** unit of measurements and the second **float**). In this case, the BSV searches in Recipe Cooker for the corresponding Adaptor Node (for the above example, the corresponding Adaptor Node has the name *AdaptorNodestringtofloat*). If the Adaptor Node does not exist, the BSV should develop and add it in the Recipe Cooker.
4. The BSV sends the response back to Recipe Cooker, using JSON format, with the updated flow, which has a new “wire” with the Adaptor Node between two initial Things (ingredients) of the recipe (Figure 28). The updated flow can be imported and saved by the Recipe Cooker. The advantage of this process is that after resolving the semantic interoperability conflicts between these two specific Things, in any future interaction that will be required for these, the Adapter Node will be added to the corresponding recipe to ensure semantic interoperability (Figure 29).



The new custom Adaptor Node, which has been implemented and added by BSV

A simple node that converts the message payloads in String to Float

```
    {
        "id":"b5cdb66b.3c8dc",
        "type":"tab",
        "label":"Flow 1",
        "disabled":false,
        "info":""
    },
    {
        "id":"fc839817.f2fe5",
        "type":"TemperatureDevice",
        "z":"b5cdb66b.3c8dc",
        "interval":"",
        "method":"",
        "name":"",
        "value":"",
        "x":90,
        "y":260,
        "wires":[
            [
                "7968554.26c6fac"
            ]
        ]
    },
    {
        "id":"7968554.26c6fac",
        "type":"AdaptorNodestringTofloat",
        "z":"b5cdb66b.3c8dc",
        "name":"",
        "x":410,
        "y":260,
        "wires":[
            [
                "2effe51c.5ab342"
            ]
        ]
    },
    {
        "id":"2effe51c.5ab342",
        "type":"Thermostat",
        "z":"b5cdb66b.3c8dc",
        "method":"",
        "name":"",
        "value":"",
        "x":690,
        "y":260,
        "wires":[
            [

            ]
        ]
    }
```

FIGURE 28 RECIPE EXAMPLE AFTER SEMANTIC VALIDATION

Before continuing the implementation of the other functionalities of BSV, it is worthwhile analyzing the step 3b. Specifically, in this phase, the BSV component [45] should try to detect any possible conflict between interacting Things, which are referring to the use of different data transformation formats (i.e. the first Thing uses string unit of measurements and the second float). Thus, in real time, it checks if there is any Adaptor Node available with this functionality in the Recipe Cooker; if not, a new Adapter Node is dynamically created with the required functionality at any case. This process is based on the creation of nodes in Node-Red platform, since Recipe Cooker is an extension of this platform and consists of a pair of files i) a JavaScript file that defines the role of the node, ii) an html file that defines the node's properties, edit dialog and help text and iii) a package.json file is used to package it all together as an npm module. The said files are added under the core folder of Recipe Cooker (Node-Red platform) *~/.node-red/node_modules.* For the above Adaptor Node example (*AdaptorNodestringtofloat*), the following files are needed (Code 18,Code 19,Code 20):

```
<script type="text/javascript">
    RED.nodes.registerType('AdaptorNodestringTofloat', {
        category: 'adaptorUnits',
        color: '#DD105E',
        defaults: {
            name: {
                value: ""
            }
        },
        inputs: 1,
        outputs: 1,
        icon: "file.png",
        label: function() {
            return this.name || "AdaptorNodestringTofloat";
        }
    });
</script>
```

[45] https://nodered.org/docs/creating-nodes/first-node

```
<script type="text/x-red" data-template-name="AdaptorNodestringTofloat">
    <div class="form-row">
        <label for="node-input-name"><i class="icon-tag"></i> Name</label>
        <input type="text" id="node-input-name" placeholder="Name">
    </div>
</script>
<script type="text/x-red" data-help-name="AdaptorNodestringTofloat">
    <p>A simple node that converts the message payloads in String to Float </p>
</script>
```

CODE 18 THE .HTML FILE FOR THE NEW ADAPTOR NODE IN RECIPE COOKER

```
module.exports = function(RED) {
    function AdaptorNodestringTofloatFunction(config) {
        RED.nodes.createNode(this,config);
        var node = this;
        node.on('input', function(msg) {
            msg.payload = parseFloat(msg.payload);
            node.send(msg);
        });
    }
    RED.nodes.registerType("AdaptorNodestringTofloat",AdaptorNodestringTofloatFunction);
}
```

CODE 19 THE .JS FILE FOR THE NEW ADAPTOR NODE IN RECIPE COOKER

```
....
  "_spec":"/home/…/MyNodeRedCustomNodes/AdaptorNodestringTofloat",
  "_where":"/home/…/.node-red",
  "author":{
    "name":"…."
  },
  "dependencies":{

  },
  "description":"Adapt  AdaptorNodestringTofloat",
  "devDependencies":{

  },
  "license":"ISC",
  "main":"AdaptorNodestringTofloat.js",
  "name":" AdaptorNodestringTofloat",
  "node-red":{
    "nodes":{
      "unitadaptor":"AdaptorNodestringTofloat.js"
    }
  }
....
```

CODE 20 THE PACKAGE.JSON FILE FOR THE NEW ADAPTOR NODE IN RECIPE COOKER

Three functions are implemented for the corresponding above files, to achieve the dynamic development of the new Adaptor Nodes (*createHtml(), createJavascript(), createPackageFile()*), with the relevant arguments in order to cover functionality accordingly.

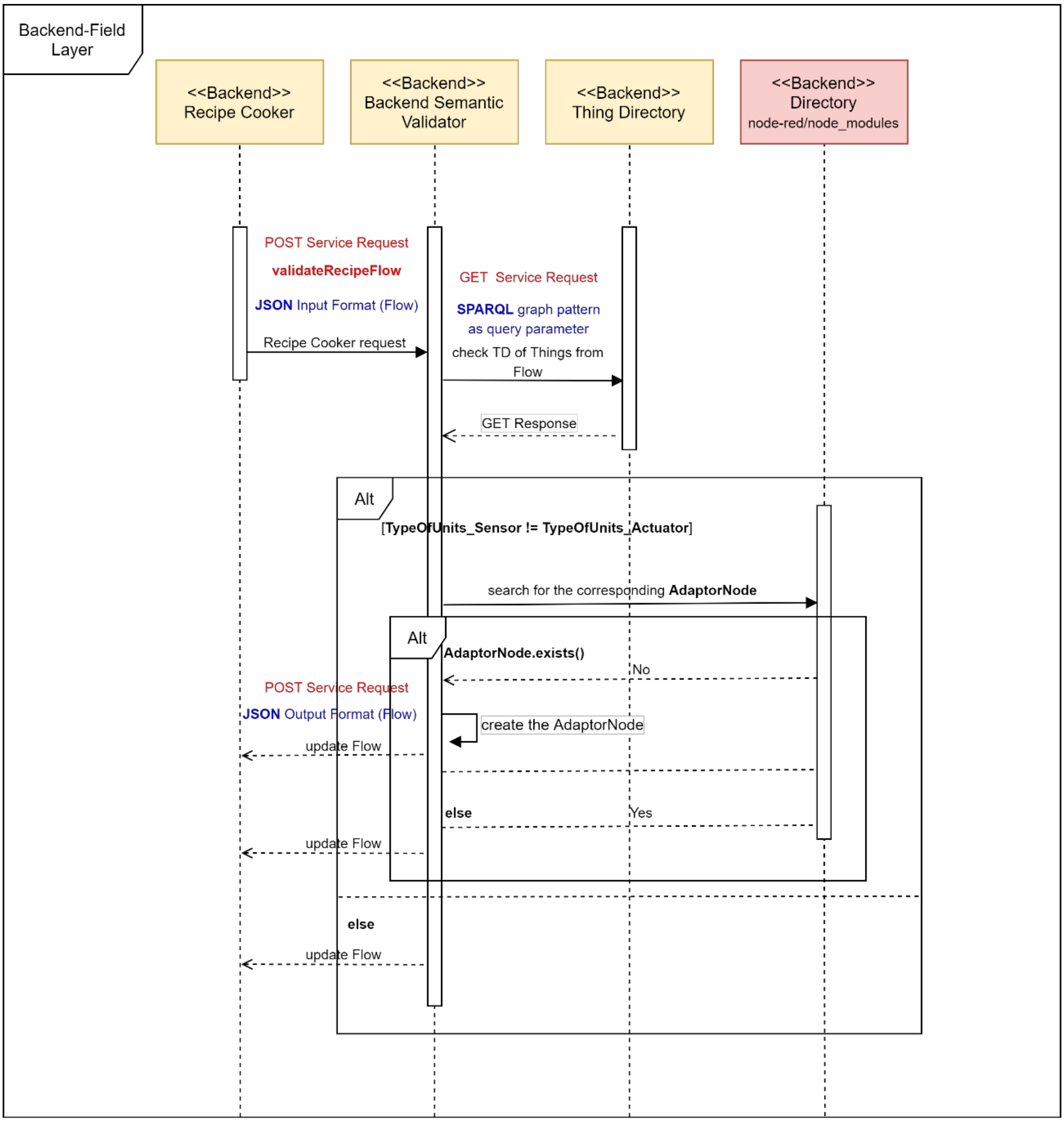

Backend-Field Layer
<<Backend>> Recipe Cooker
<<Backend>> Backend Semantic Validator
<<Backend>> Thing Directory
<<Backend>> Directory node-red/node_modules
POST Service Request
validateRecipeFlow
JSON Input Format (Flow)
Recipe Cooker request
GET Service Request
SPARQL graph pattern as query parameter
check TD of Things from Flow
GET Response
Alt
[TypeOfUnits_Sensor != TypeOfUnits_Actuator]
search for the corresponding AdaptorNode
Alt
AdaptorNode.exists()
No
POST Service Request
JSON Output Format (Flow)
create the AdaptorNode
update Flow
else
Yes
update Flow
else
update Flow

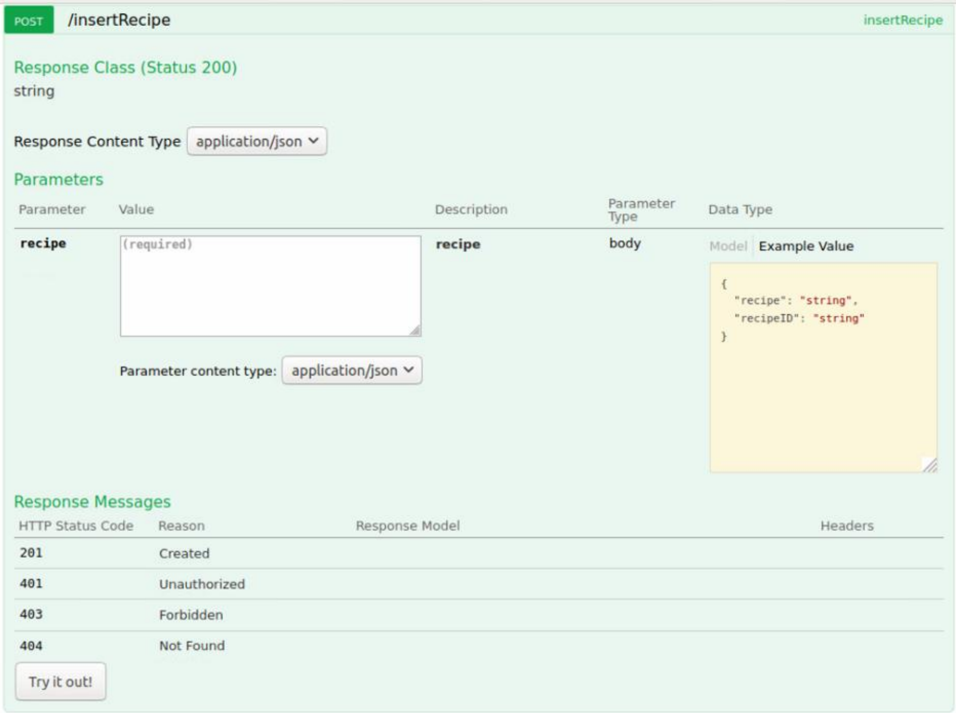
FIGURE 29 SEQUENCE DIAGRAM – RECIPE VALIDATION/CREATE ADAPTOR NODE

8.3.2 CONNECTION WITH EXTERNAL IOT PLATFORMS – IMPLEMENTATION

The integration and the connection between SEMIoTICS framework and external heterogeneous IoT platforms are presented in detail in subsection 6.2.2. Herein more technical details and development details are provided. Namely, SEMIoTICS can use the exposed interfaces of the above IoT platforms, in order to take advantage of

IoT devices whose descriptions are available in repositories outside SEMIoTICS framework. Both the key components of the SEMIoTICS architecture that are involved in this process and the specific example with target FIWARE platform that is used for the description and analysis of the development have already highlighted in Figure 15 and Figure 17 accordingly.

Based on the sequence diagram (see Figure 16), a detailed description of each component that participates and API for the interaction between them is outlined below. In fact, the Recipe Cooker component which is responsible for cooking (creating) recipes reflecting user requirements, sends the recipe in Pattern Orchestrator component using a POST method request. Particularly, this POST parameters include the recipe/flow (JSON format) as body and the header is application/json. The structure is presented in Figure 30.



POST /insertRecipe insertRecipe

Response Class (Status 200)
string

Response Content Type: application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
recipe	(required) Parameter content type: application/json	recipe	body	Model Example Value

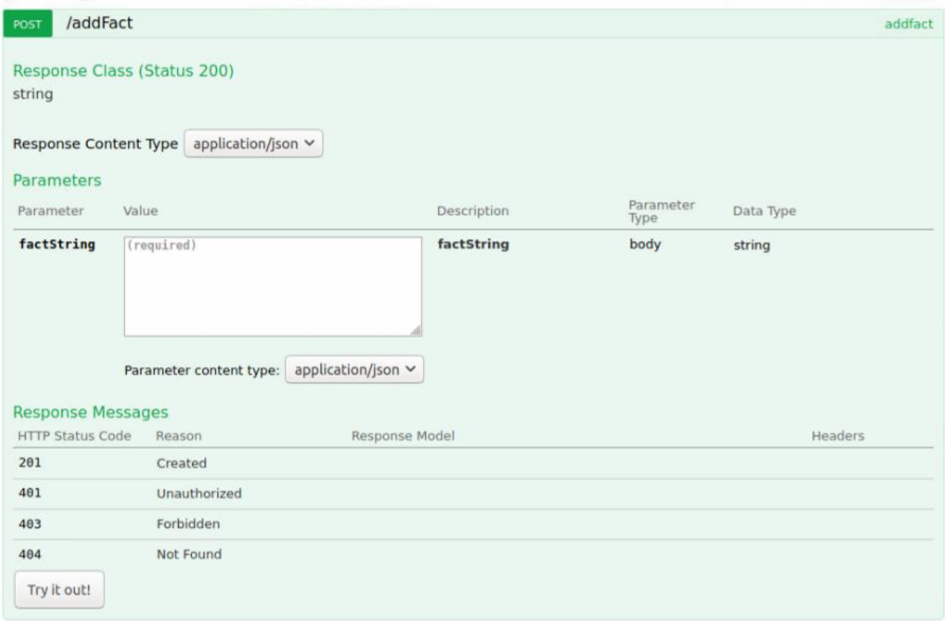
```
{
  "recipe": "string",
  "recipeID": "string"
}
```

Response Messages

HTTP Status Code	Reason	Response Model	Headers
201	Created		
401	Unauthorized		
403	Forbidden		
404	Not Found		

FIGURE 30 API BETWEEN RECIPE COOKER (BACKEND) – PATTERN ORCHESTRATOR

The next step from this approach is the request from Pattern Orchestrator to Pattern Engine (Backend) component in order to send the Interoperability requirement using a POST method with the following parameters (see Figure 31).



POST /addFact addfact

Response Class (Status 200)
string

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
factString	(required)	factString	body	string

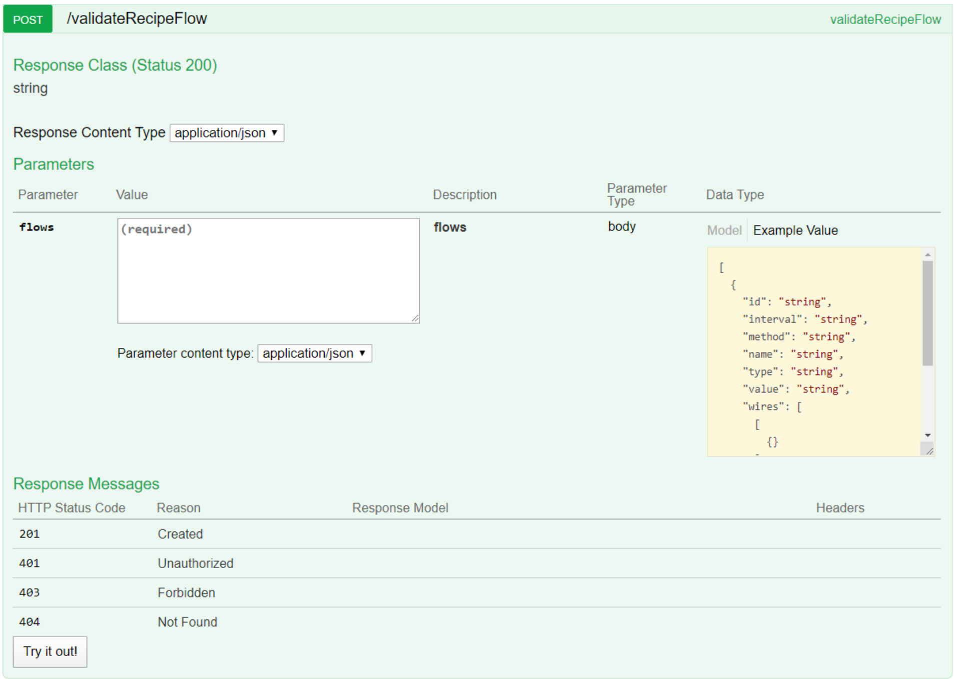
Parameter content type: application/json

Response Messages

HTTP Status Code	Reason	Response Model	Headers
201	Created		
401	Unauthorized		
403	Forbidden		
404	Not Found		

FIGURE 31 API BETWEEN PATTERN ORCHESTRATOR - PATTERN ENGINE (BACKEND)

Afterwards, the BSV component receives a request from the Pattern Engine (Backend) to check the semantic interoperability between two Things (link) in JSON-LD/JSON format. The JSON-LD/JSON Parser is implemented as part of the BSV, to analyze the received input and extract the meaningful information from these set of data. The communication between said components is achieved using the POST method (Figure 32).



POST /validateRecipeFlow validateRecipeFlow

Response Class (Status 200)
string

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
flows	(required)	flows	body	Model Example Value

Parameter content type: application/json

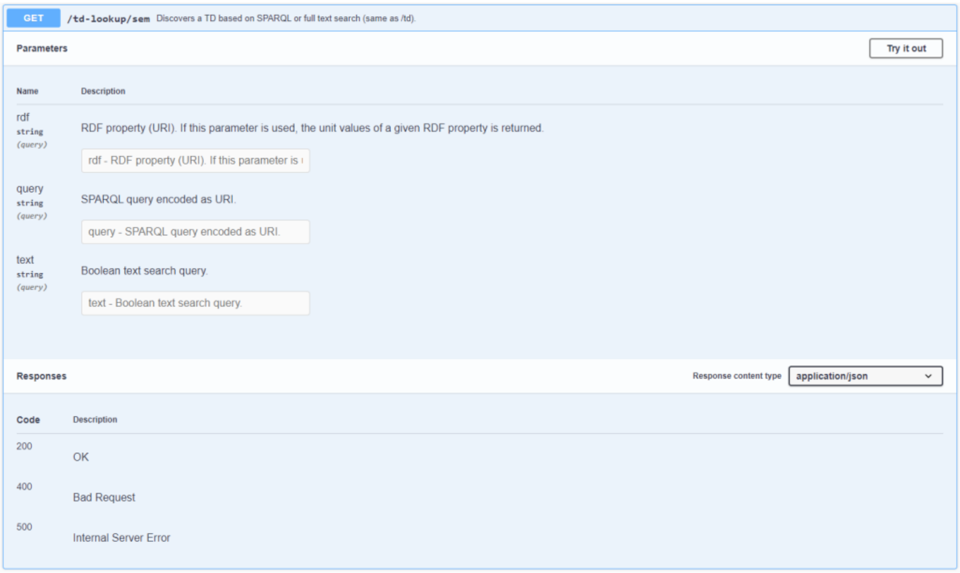
```
[
  {
    "id": "string",
    "interval": "string",
    "method": "string",
    "name": "string",
    "type": "string",
    "value": "string",
    "wires": [
      [
        {}
```

Response Messages

HTTP Status Code	Reason	Response Model	Headers
201	Created		
401	Unauthorized		
403	Forbidden		
404	Not Found		

FIGURE 32 API BETWEEN PATTERN ENGINE (BACKEND) – BACKEND SEMANTIC VALIDATOR

The BSV begins the procedure to tackle the semantic interoperability issues between these two Things from said recipe/flow. In order to give this answer, the semantic description for any Thing is required (for FIWARE Sensor and SEMIoTICS Thermostat). For that reason, it sends two requests: i) SPARQL query to Thing Directory in order to receive the Thing Description of SEMIoTICS Thermostat (see Figure 33) and ii) GET method to the Orion Context Broker FIWARE platform to receive the context data Description of FIWARE Sensor. The response consists of JSON with the FIWARE Sensor attributes (type, metadata elements).



GET `/td-lookup/sem` Discovers a TD based on SPARQL or full text search (same as /td).

Parameters

Name	Description
rdf string *(query)*	RDF property (URI). If this parameter is used, the unit values of a given RDF property is returned. rdf - RDF property (URI). If this parameter is ı
query string *(query)*	SPARQL query encoded as URI. query - SPARQL query encoded as URI.
text string *(query)*	Boolean text search query. text - Boolean text search query.

Responses

Response content type: application/json

Code	Description
200	OK
400	Bad Request
500	Internal Server Error

FIGURE 33 API FOR DISCOVERY OF THING DESCRIPTION IN THING DIRECTORY

Based on this information, the BSV could decide for the interoperability between the Things and harmonize the semantic model capabilities with the registration of extra Adaptor Nodes in the recipe. Particularly, there are three possible results. Firstly, the link source and destination are interoperable, so the BSV updates the Pattern Engine (Backend) with the TRUE response. Secondly, the link source and destination are not interoperable and the BSV can add Adaptor Nodes in order to guarantee the interoperability. In this case, BSV not only sends the TRUE response in Pattern Engine (Backend) but also updates the recipe in Recipe Cooker using the corresponding Adaptor Nodes. Lastly, the link source and destination are not interoperable and BSV does not have the required information to develop the Adaptor Nodes; hence, the Pattern Engine (Backend) receives the FALSE response by the BSV.

8.3.3 INTEROPERABILITY ADAPTATION – IMPLEMENTATION

Two different types of pattern-driven orchestration adaptations are foreseen, as described in D4.8 (“SEMIoTICS SPDI Patterns (final)”): i) at design-time, and ii) at runtime. From the semantic perspective, the second type is important and can be applicable in the development of data transformation techniques and validation mechanisms of SEMIoTICS framework. In fact, the adaptability is mentioned to the requirement of the SEMIoTICS to detect and deal with changes in the system, for example, a failure of a sensor used in a flow in Recipe Cooker (user interface); in this case adaptation mechanism should detect new available devices/components and replace it. During this procedure, the main requirement is to maintain interoperability in the system, the efficient coordination and cooperation among the sensors/components. This requirement is provided by the BSV component.

The approach and the implementation of this process has already been analyzed in detail in the previous subsection (8.3.2). The only difference, in this case, is that the BSV decides for the interoperability between the replacement component and the new, which will be introduced, in order to harmonize the semantic model capabilities with the registration of extra Adaptor Nodes.

9 SEMANTIC INTEROPERABILITY MECHANISMS IN THE SEMIoTICS USE CASES

Based on the above analysis of the development of data transformation techniques and validation mechanisms for the semantic interoperability, the following subsections describe three scenarios focusing on the use cases considered in SEMIoTICS. These scenarios are described from the perspective of said mechanisms.

9.1 Use Case 1

The aim of this Use Case scenario is the IIoT integration in Wind Park Control Network providing value-added services. Specifically, one of its parts refers to taking local action on sensing and analyzing structured data to find the inclination of a steel tower. Also, the Recipe Cooker component is used in this Use Case to build the application flow of the AI pipeline for grease leakage detection; this is represented by a flow with components and connections between them for the interaction and the exchange of data (more details are given in D2.5 “SEMIoTICS high level architecture (final)”).

However, at runtime, a possible change in resource availability may require an adaptation mechanism for the dynamic recomposition of the flow components, without any change of the system behavior. For example, let’s assume that the above flow has an Inclinometer component. If, for a reason, the Inclinometer becomes unavailable, another component from the IoT repository with the same functionality is selected to replace the one that has become unavailable. In this case, interoperability of the link between the replaced component and the connected flow component should be checked before the final decision for the replacement to ensure the initial target of the system. In particular, a potential conflict that needs to be addressed in case of the Inclinometer is the different unit of measurement of angle (degrees, radians), which is used by the two components (the replaced and the connected flow component).

The solution in this interoperability adaptation issue is provided by the approach that has already described in the previous section (see Section 8) with the participation of the BSV as a core component. It examines the interoperability between the corresponding sensors and tries to ensure the interaction between them creating Adaptor Nodes. Figure 34 highlights and explains said methodology in the specific Use Case scenario.

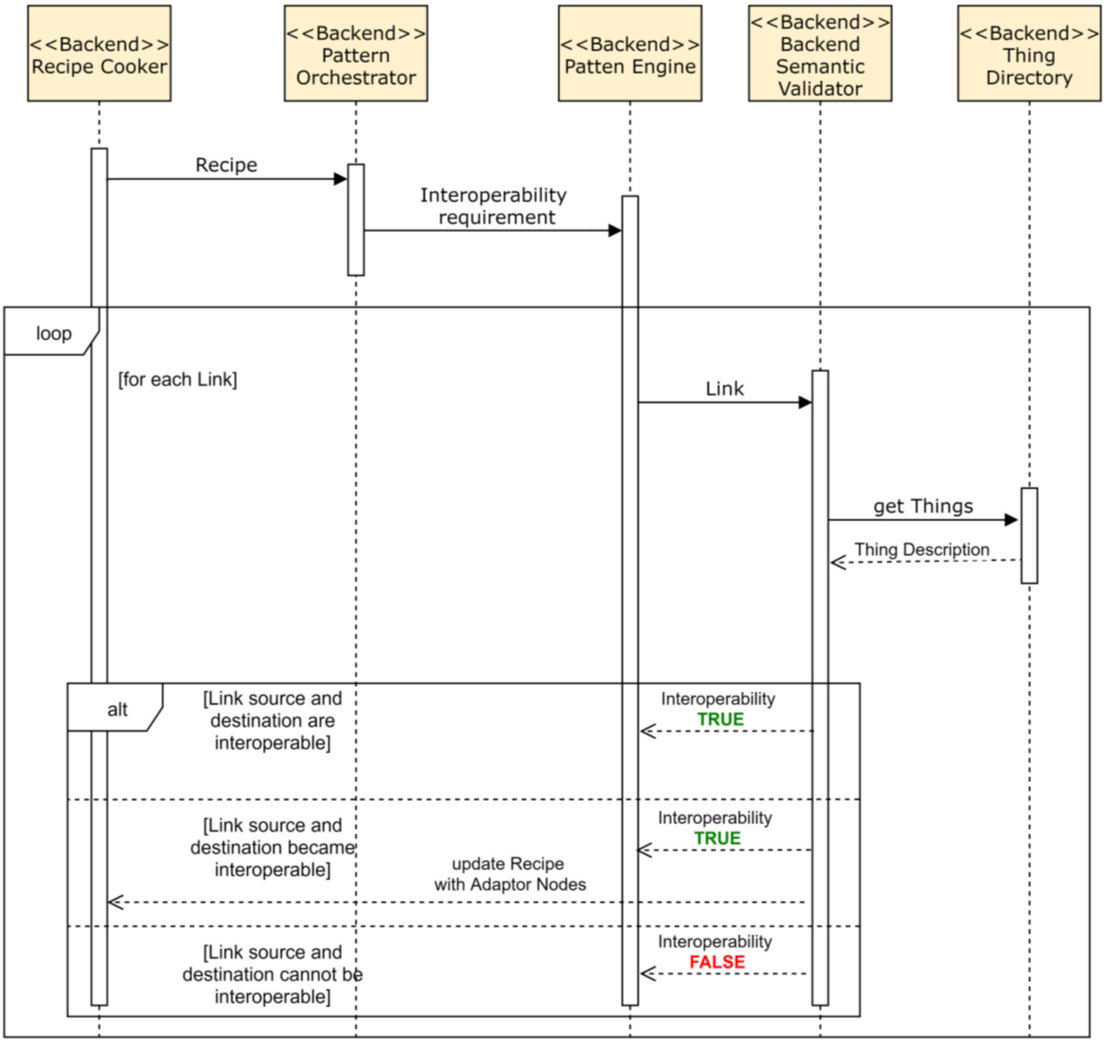


FIGURE 34 UC1 INTEROPERABILITY ADAPTATION APPROACH

9.2 Use Case 2

This Use Case employs the SEMIoTICS technologies to develop an Information and Communication Technology (ICT) solution aimed at sustained independence and preserved the quality of life for elders with Mild Cognitive Impairment or mild Alzheimer's disease supporting both 'aging in place' (individuals remain in the home of choice as long as possible) and 'community care' (long-term care for people who are mentally ill, elderly, or disabled provided within the community rather than in hospitals or institutions) - more details are given in D2.5 "SEMIoTICS high level architecture (final)".

In detail, this Use Case aims to a possible Fall Event and is consisted of three main phases i) Early Warning, ii) Search, and iii) Rescue. The objective of the Search phase is to allow the Robotic Assistant to reach the location where the (possible) fall event occurred (D2.5 – Figure 21). This phase is initiated by the Smart Environment sending to the Robot the request to move to the location of the event. Before this step, the interoperability mechanisms could be intervened in order to ensure that the data format for the location between UC2 Smart Environment and UC2 Robot is the same, in order to achieve the target of the system (for example use both Degrees Minutes Seconds (DMS) or Decimal Degrees).

The SEMIoTICS components and the interactions between them that are involved in this technique are highlighted in the sequence diagram (see Figure 35). The methodology of this mechanism has already described and analyzed in the previous section (Section 8). It is worthwhile noting that BSV component, which is responsible to resolve such conflicts, belongs in the Backend layer of SEMIoTICS architecture. Nevertheless, the Search Phase of the UC2 starts from the field layer. For that reason, the Pattern Engine (Backend) updates the Pattern Orchestrator; afterthought Pattern Orchestrator informs the Pattern Engine (field Layer) in order to trigger the UC2 Environment (field layer) component.

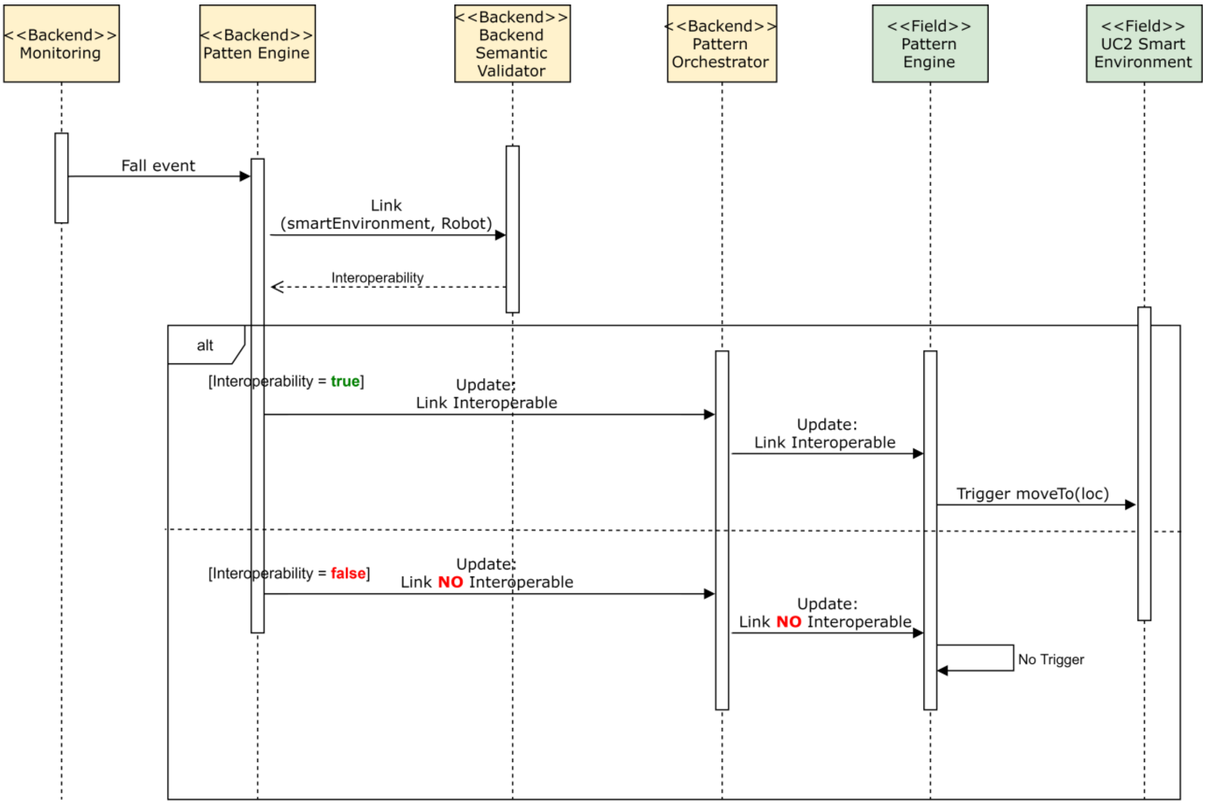


FIGURE 35 UC2 INTEROPERABILITY VALIDATION APPROACH

9.3 Use Case 3

One of the main aims of this Use Case scenario is to distribute vibration monitoring for earthquake detection. Specifically, the system is composed by the Gateway, which is connected at least of a set of IHES Sensing Units nodes (i.e. environmental sensors: temperature, humidity, pressure, luminosity) and requires monitoring and adaptation mechanisms for the redundant sensors that is used in the deployment to ensure that, even in the case of failures, another sensor is available to provide inputs (more details are given in D2.5 “SEMIoTICS high level architecture (final)).

Hence, let’s assume that three IHES Sensing Units nodes are involved in this scenario and are connected in the Gateway, but only for two of these collect data. If for a reason, one of two sensors become unavailable, the third sensor, with the same functionality (i.e. temperature sensing) should be selected to replace the one that has become unavailable. In this case, interoperability of the link between the replaced sensor and the Gateway should be checked before the final decision for the replacement to ensure the initial target of the system. In particular, a potential conflict that needs to be addressed in case of the temperature sensor is the different unit of measurement of the thermometer (Celsius, Kelvin, Fahrenheit), which is used by the two components (the link of the replaced sensor and the Gateway). The above functionality, which is part of the UC3 Pattern-based monitoring method, belongs to the recover actions such as node replacement, system reconfiguration, alerting, etc.(see Section 4.3.2 in D2.4) and is described in Figure 36. The sequence diagram in Figure 36 highlights the SEMIoTICS components that are included in this approach; it is intervened before the GUI update step in Figure 37.

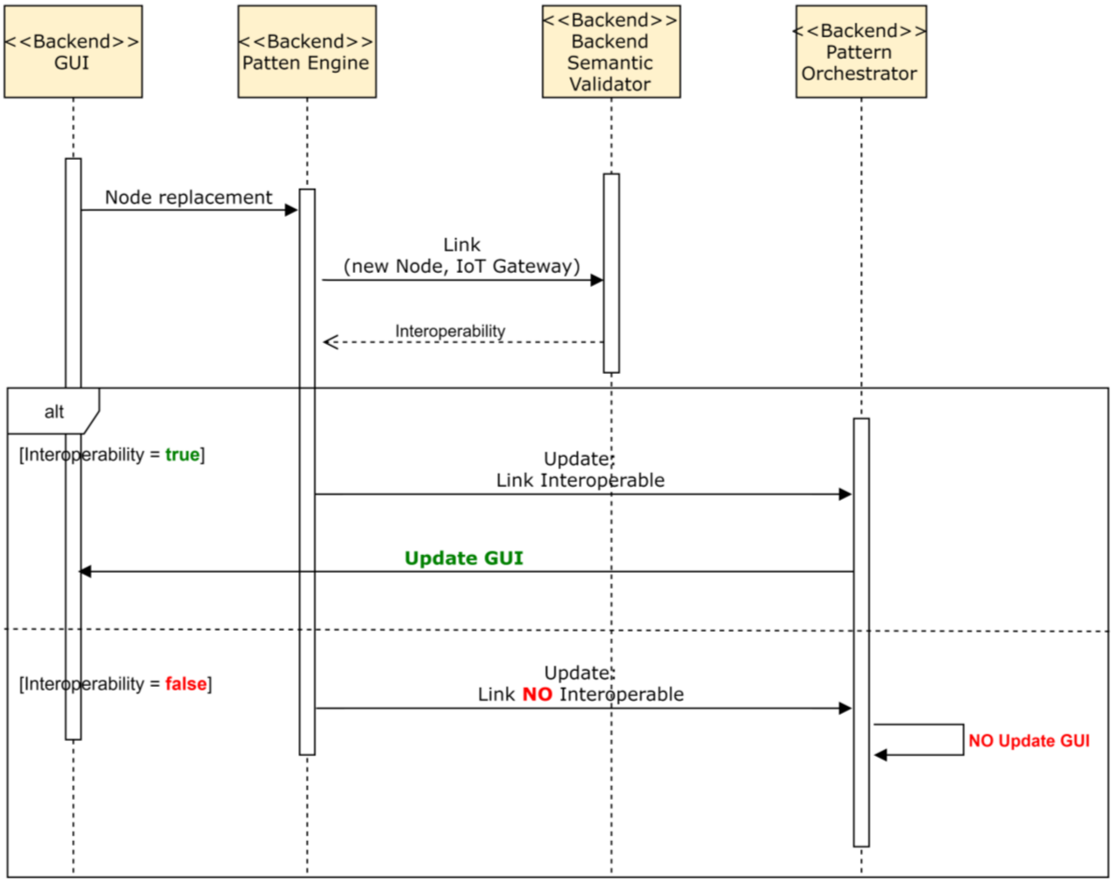


FIGURE 36 UC3 INTEROPERABILITY VALIDATION APPROACH

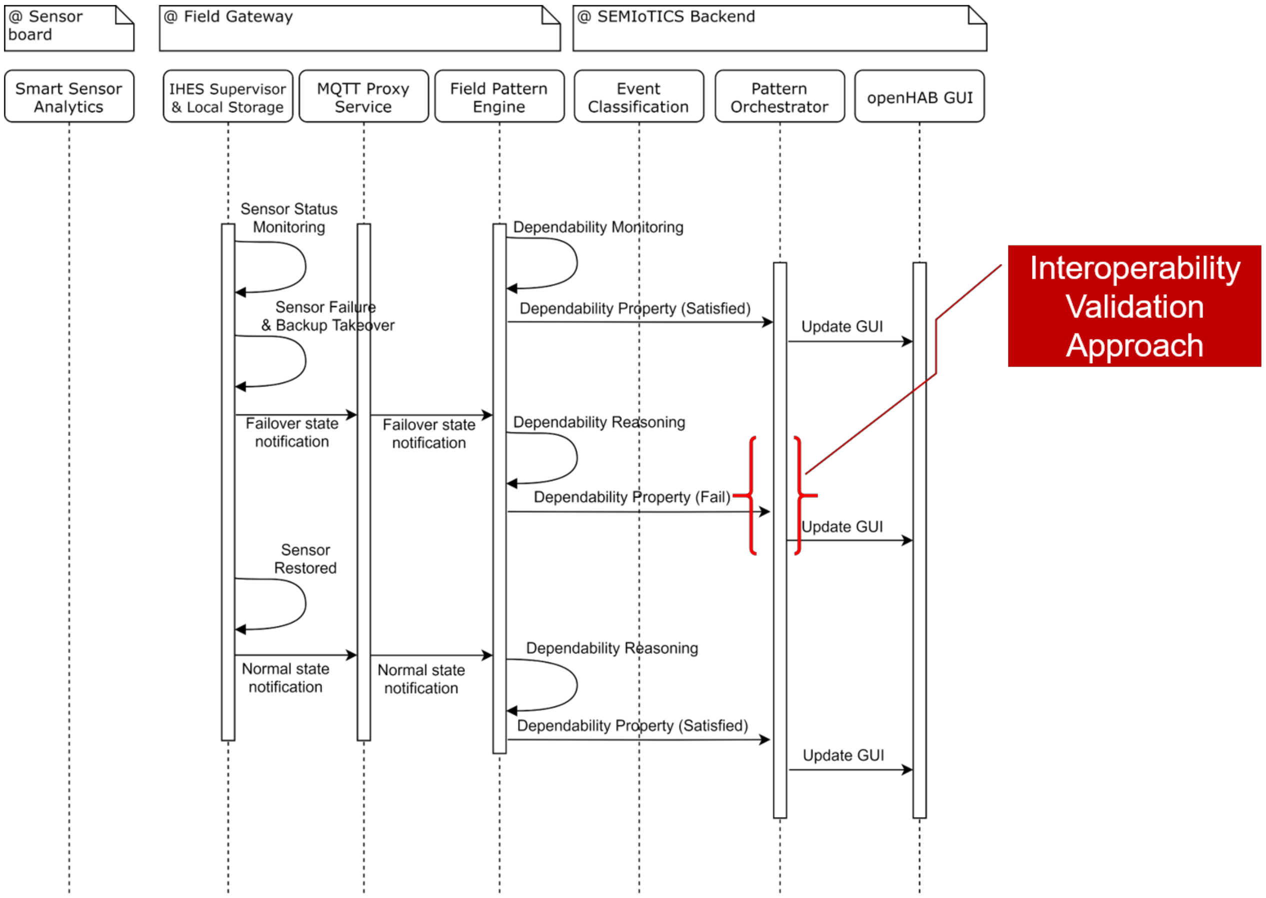

@ Sensor board
@ Field Gateway
@ SEMIoTICS Backend
Smart Sensor Analytics
IHES Supervisor & Local Storage
MQTT Proxy Service
Field Pattern Engine
Event Classification
Pattern Orchestrator
openHAB GUI
Sensor Status Monitoring
Sensor Failure & Backup Takeover
Failover state notification
Failover state notification
Sensor Restored
Normal state notification
Normal state notification
Dependability Monitoring
Dependability Property (Satisfied)
Update GUI
Dependability Reasoning
Dependability Property (Fail)
Update GUI
Dependability Reasoning
Dependability Property (Satisfied)
Update GUI
Interoperability Validation Approach

FIGURE 37 UC3 SEQUENCE FLOWS (FIGURE 103 – D4.8) – INTEROPERABILITY VALIDATION PARTICIPATION

10 VALIDATION

This chapter summarizes the validation features of SEMIoTICS that are related to the semantic interoperability and the various topics that are covered in this deliverable.

10.1 Related Project Objectives and Key Performance Indicators (KPIs)

The objectives related to Task 4.4 and their mapping to D4.11 content are summarized in the Table 9.

TABLE 9 TASK 4.4 OBJECTIVES

T4.4 Objectives	D4.11 Chapters
Semantics • Definition of semantic annotations for the SPDI patterns defined in T4.1 • Development of data transformation techniques and validation mechanisms to ensure end-to-end semantic interoperability • Definition of the mappings between datatypes used in SEMIoTICS, to ensure that data flow is possible between smart objects that are linked in the composition structure defined by the pattern	2, 3, 4, 7
Semantic Mediators • Definition of SEMIoTICS semantic mediators' mechanisms, with the purpose of resolving, if possible, conflicts among the semantic models used in the semantic annotations of the patterns • The semantic mediators' mechanisms will rely on ontology alignment and data transformation techniques • The mechanisms to validate end-to-end semantic interoperability will rely on the inclusion of interoperability conditions in the patterns and be based on the use of semantic reasoners or rule engines, as well as logic programming; the development of validation mechanisms will be driven by the way semantic interoperability conditions are defined	2, 5, 8
Operation • Definition of data flows between SEMIoTICS architecture components • Identification of the elements responsible for logical and structural data transformation • Identification of the SEMIoTICS architecture elements that are responsible for resolving semantic differences and provide an implementation of them to support the usage scenarios	2, 6, 8, 9

The overall deliverable constitutes the initial contribution towards fulfilling the project's requirements regarding **SEMIoTICS's objective 2** ("Development of semantic interoperability mechanisms for smart objects, networks and IoT platforms") and the relevant **KPI 2.2** ("Delivery of data type mapping and ontology alignment and transformation techniques that realize semantic interoperability") and **KPI 2.3** ("Validated semantic interoperability between the SEMIoTICS framework and 3 IoT platforms, including FIWARE").

Furthermore, the satisfaction of Task 4.4 and the mapping measurement of the corresponding KPIs is evaluated with a UC-specific scenario including data flow which is possible between smart objects and is linked in the composition structure defined by the SPDI patterns of Task 4.1 (see Section 9).

10.2 SEMIoTICS Interoperability Requirements

The general SEMIoTICS' requirements (**D2.3**) that are covered by the presented semantic interoperability mechanisms are summarized in the next table.

TABLE 10 TASK'S OBJECTIVES

Requirements (D2.3)	Description	D4.11 Sections
R.GP.1	End-to-end connectivity between the heterogeneous IoT devices (at the field level) and the heterogeneous IoT Platforms (at the backend cloud level)	Section 6
R.UC1.8	Semantic and robust bootstrapping/registration of IIoT sensors and actuators with IIoT gateway MUST be supported	Subsection 2.2.1
R.UC1.9	Semantic interaction between use-case specific application on IIoT Gateway and legacy turbine control system MUST be supported	Section 8, Section 9
R.UC1.12	Standardized semantic models for semantic-based engineering and IIoT applications SHALL be utilized	Subsection 4.2, Subsection 4.3
R.UC2.3	The SEMIoTICS platform SHOULD guarantee proper connectivity between the various components of the SARA distributed application. The SARA solution is a distributed application not only because it uses different cloud services (e.g. AREAS Cloud services, AI services) from different remote computational nodes, but also because the SARA application logic itself is distributed across various edge nodes (SARA Hubs).	Section 6, Subsection 9.2
R.UC2.6	The SEMIoTICS platform SHOULD allow the SARA solution to retrieve the resources exposed by registered devices via their object model (i.e. a data structure wherein each element represents a resource, or a group of resources, belonging to a device). The SEMIoTICS platform SHOULD support at least the OMA LWM2M object model.	Subsection 3.2, Subsection 9.2
R.UC2.11	The SEMIoTICS platform SHOULD allow a SARA component to request a group of devices to take/initiate an action (e.g. turn on/off a light bulb).	Subsection 2.2, Subsection 9.2
R.UC3.1	IoT Sensing unit shall be able to embed environmental (e.g. temperature, pressure, humidity, light) and inertial sensors (accelerometer, gyroscope).	Subsection 2.2, Subsection 9.3
R.UC3.15	A use case specific serialized message protocol is required to coordinate the gateway and its associated units and exchange data / events / anomalies between them. JSON will be the preferred serialization format adopted.	Subsection 5.1.1, Subsection 9.3

11 CONCLUSION

This deliverable, being the final output of Task 4.4 ("End-to-End Semantic Interoperability") presented the landscape for accomplishing semantic interoperability in the IoT domain. To do so, the state-of-the-art approaches were reviewed, including techniques and technologies for semantics, data mappings, ontologies alignment, semantic reasoning, etc.

The main outcome is the proposal of the semantic interoperability mechanisms that can be deployed in across the two main SEMIoTICS's layers (field, backend) and the provision of the required common representation and meaning of data. Moreover, a full set of Adaptor Nodes, which will be responsible for resolving, if possible, conflicts among the semantic annotations, are described in subsection 5.1.3, along with their implementation details in subsection 8.3.1.

Additionally, this deliverable describes the final thoughts towards the integration of SEMIoTICS with other IoT platforms. Then, the translation of Recipes into SPDI Patterns and semantic interoperability patterns are analysed. The overall process is further analysed in Deliverable 4.1 and its follow-up, D4.8.

Finally, both the initial and the final mechanisms/designs that are responsible for resolving semantic differences have described. Hence, the elements that are responsible for resolving semantic differences and provide an implementation of them to support the SEMIoTICS Usage Scenarios are presented in Section 9.

To sum up, the mechanisms and designs of data transformation techniques and validation mechanisms to ensure end-to-end semantic interoperability presented by this deliverable. Particularly, the final implementation of mentioned algorithms, the API for the connectivity to external IoT platforms and the integration of the involved components will be analysed in Task 4.6 – Implementation of SEMIoTICS backend API. Also, the demonstration and validation of the above semantic interoperability and adaptation techniques in the SEMIoTICS use cases will be described in Task 5.4 – Demonstration and validation of IWPC- Energy scenario, Task 5.5 – Demonstration and validation of SARA-Health scenario and Task 5.6 – Demonstration and validation of IHES-Generic IoT scenario.

REFERENCES

[1] Hatzivasilis, G., Askoxylakis, I., Alexandris, G., and Fysarakis, K., 2018. The Interoperability of Things. 23rd IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD 2018), IEEE, Barcelona, Spain, 17-19 September, 2018, pp. 1-7.

[2] Hatzivasilis, G., Fysarakis, K., Soultatos, O., Askoxylakis, I., Papaefstathiou, I. and Demetriou, G., 2018. The Industrial Internet of Things as an enabler for a Circular Economy Hy-LP: a novel IIoT protocol, evaluated on a Wind Park's SDN/NFV-enabled 5G Industrial Network. Computer Communications, Elsevier, vol. 119, pp. 127-137.

[3] Thuluva, A. S., BrÖring, A., Medagoda, G. P., Don, H., Anicic, D. and Seeger, J., 2017. Recipes for IoT applications. IoT Conference, ACM, Linz, Austria, October 22-25, pp. 1-8.

[4] Bröring, A., Schmid, S., Schindhelm, C.-K., Kheli, A., Kabisch, S., Kramer, D., Phuoc, D., Mitic, J., Anicic, D. and Teniente, E., 2017. Enabling IoT ecosystems through platform interoperability. IEEE Software, IEEE, vol. 34, issue 1, pp. 54-61.

[5] Ganzha, M., Paprzycki, M., Pawlowsji, W., Szmeja, P. and Wasielewska, K., 2016. Semantic technologies for the IoT – an Inter-IoT perspective. 1st International Conference on the Internet-of-Things Design and Implementation (IoTDI), IEEE, Berlin, Germany, pp. 271-276.

[6] Serra, J., Pubill, D., Antonopoulos, A. and Verikoukis, C., 2014. Smart HVAC control in IoT: energy consumption minimization with user comfort constraints. The Scientific World Journal, Hindawi, vol. 2014, article ID 161874, pp. 1-11.

[7] Vilalta, R., Mayoral, A., Pubill, D., Casellas, R., Martinez, R., Serra, J., Verikoukis, C. and Munoz, R., 2016. End-to-end SDN orchestration of IoT services using an SDN/NFV-enabled edge node. Optical Fiber Communications Conference and Exhibition (OFC), Anaheim, CA, USA, pp. 1-3.

[8] Fysarakis, K., Hatzivasilis, G., Papaefstathiou, I. and Manifavas, C., 2016. RtVMF – A secure Real-time Vehicle Management Framework with critical incident response. IEEE Pervasive Computing Magazine (PVC) – Special Issue on Smart Vehicle Spaces, IEEE, vol. 15, issue 1, pp. 22-30.

[9] Hatzivasilis, G., Papaefstathiou, I. and Manifavas, C., 2017. SCOTRES: secure routing for IoT and CPS. IEEE Internet of Things (IoT) Journal, IEEE, vol. 4, issue 6, pp. 2129-2141.

[10] Internet of Things IoT Semantic Interoperability: Research Challenges, Best Practices, Recommendations and Next Steps, European Research Cluster on the Internet of Things (IERC-AC4), March 2015.

[11] Cousin, P., Serrano, M, and Soldatos J., 2015. Internet of Things Research on Semantic Interoperability to Address Manufacturing Challenges Available at: https://doi.org/10.1002/9781119081418.ch3 [Accessed 21 Feb. 2018].

[12] Dadzie, A.-S. and Rowe, M., 2011. Approaches to visualising linked data: a survey. Semantic Web, IOS Press, vol. 2, issue 2, pp. 89-124.

[13] Hernandez-Serrano J., et al., 2017. On the Road to Secure and Privacy-Preserving IoT Ecosystems. InterOSS-IoT, LNCS 10218, pp. 107–122.

[14] Guarino N., Oberle Daniel, Staab S., 2009. What is an ontology? In Handbook on ontologies, pp. 1-17.

[15] Yachir, A., Djamaa B., Mecheti A., Amirat Y., Aissani M., 2016. A Comprehensive Semantic Model for Smart Object Description and Request Resolution in the Internet of Things. Procedia Computer Science, pp. 147-154.

[16] Hachem, S., Teixeira, T., Issarny, V. 2011. Ontologies for the Internet of Things. Proceedings of the 8th Middleware Doctoral Symposium, ACM.

[17] Haller, A., et al., 2018. The SOSA/SSN ontology: a joint WeC and OGC standard specifying the semantics of sensors, observations, actuation, and sampling. Semantic Web, IOS Press, vol. 1-0X, pp. 1-19.

[18] Zeng, D., Guo, S. and Cheng, Z., 2011. The Web of Things: a survey. Journal of Communications, Academy Publisher, vol. 6, no. 6, pp. 424-438.

[19] Soldatos, J. et al., 2015. OpenIoT: Open source Internet-of-Things in the Cloud. Interoperability and Open-Source Solutions for the Internet of Things, Springer, LNCS, vol. 9001, pp. 13-25.

[20] Ganzha, M. et al., 2017. Semantic interoperability in the Internet of Things, as overview from the INTER-IoT perspective. Journal of Network and Computer Applications, Elsevier, vol. 81, issue 1, pp. 111-124.

[21] Cameron, J. D., Ramaprasad, A. and Syn T., 2015. An ontology of mHealth. 21st Americas Conf]rence on Information Systems (AMCIS), Puerto Rico, pp. 1-11.

[22] Daniele, L., den Hartog, F. and Roes, J., 2015. Created in close interaction with the industry: the smart appliances reference (SAREF) ontology. International Workshop Formal Ontologies Meet Industries (FOMI), Springer, LNBIP, vol. 225, pp. 100-112.

[23] Bajaj, G., et al., 2017. A study of existing ontologies in the IoT-domain, HAL Archives, hal-01556256, pp. 1-24.

[24] Semantic Web Standards and Ontologies in the Medical Sciences and Healthcare [online] Available at: http://what-when-how.com/medical-informatics/semantic-web-standards-and-ontologies-in-the-medical-sciences-and-healthcare/ [accessed 23 March 2018].

[25] Bicer V., Laleci G., Dogac A., Kabak Y., 2005. Artemis message exchange framework: semantic interoperability of exchanged messages in the healthcare domain. ACM SIGMOD.

[26] Nardon F.B., Moura L.A., 2004. Knowledge sharing and information integration in healthcare using ontologies and deductive databases. MEDINFO 2004, pp. 62-66.

[27] Bjorklund M., 2016. The YANG 1.1 Data Modeling Language, August 2016. RFC 7950.

[28] Medved J., Varga R., Tkacik A., and Gray K. 2014. Opendaylight: Towards a model-driven SDN controller architecture, In 2014 IEEE 15th International Symposium on, pp. 1–6. IEEE.

[29] Enns R., et al, 2011. Network Configuration Protocol (NETCONF), IETF RFC 6241.

[30] Bierman A. et al., 2017. RESTCONF Protocol, IETF RFC 8040.

[31] Drools Rules Engine – CISCO [online] Available at: https://www.cisco.com/c/en/us/td/docs/net_mgmt/active_network_abstraction/3-6_sp1/administrator/ruleseng.pdf [Accessed 15 Mar. 2018].

[32] Szilagyi I. and Wira P., 2016. Ontologies and Semantic Web for the Internet of Things - a survey,"IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society, Florence, pp. 6949-6954.

[33] Krötzsch M., Simancík F., Horrocks I., 2014. A description logic primer, ˇ in Perspectives on Ontology Learning, ser. Studies on Semantic Web, Amsterdam, The Netherlands: IOS Press.

[34] Gyrard A., Bonnet C., Boudaoud K. 2018. Helping IoT application developers with sensor-based linked open rules, in Proc. 7th Int. Workshop Semantic Sensor Netw., pp. 105–108.

[35] Su X. et al. 2016. Stream reasoning for the Internet of Things: Challenges and gap analysis, in Proc. 6th Int. Conf. Web Intell. Min. Semantics (WIMS), Nîmes, France, Jun.

[36] Daga E., Panziera L., Pedrinaci C. 2015. Basil: A cloud platform for sharing and reusing SPARQL queries as Web APIs, in Proc. Int. Semantic Web Conf. (Posters & Demos).

[37] Fafalios, P., Yannakis, T. and Tzitzikas, Y. 2016. Querying the Web of data with SPARQL-LD. International Conference on Theory and Practice of Digital Libraries (TPDL), Hannover, Germany, 5-9 September, Springer, LNCS, vol. 9819, pp. 175-187.

[38] IoT Broker: Available at: https://www.slideshare.net/FI-WARE/iot-broker [Accessed 21 Feb. 2018].

[39] Seeger, J., Bröring A., Pahl M.-O., Sakic. E. 2019. Rule-Based Translation of Application-Level QoS Constraints into SDN Configurations for the IoT. EuCNC 2019, 18.-21. June, Valencia, Spain. IEEE.

[40] Seeger, J., Deshmukh R.A., Bröring A. 2018. Running Distributed and Dynamic IoT Choreographies. Global Internet of Things Summit (GIoTS 2018), 4.-7. June 2018, Bilbao, Spain. IEEE.

[41] Thuluva, A.S., Bröring A., Medagoda Hettige Don G.P., Anicic D., Seeger J. 2017. Recipes for IoT Applications. Proceedings of the 7th International Conference on the Internet of Things (IoT 2017), 22.-25. October 2017, Linz, Austria. ACM.

[42] Seeger, J., Bröring A., Pahl M.-O., Sakic E. 2019. Rule-Based Translation of Application-Level QoS Constraints into SDN Configurations for the IoT. EuCNC 2019, 18.-21. June, Valencia, Spain. IEEE.

[43] E. Sakic, Kulkarni V., Theodorou V., Matsiuk A., Kuenzer S., Petroulakis N. E., Fysarakis K.. 2018. Virtuwind–an sdn-and nfv-based architecture for softwarized industrial networks, in International Conference on Measurement, Modelling and Evaluation of Computing Systems. Springer, 2018, pp. 251–261.

[44] Haroon A., Shah M.A., Asim Y., Naeem W., Kamran M., Javaid Q. 2016. Constraints in the IoT: The World in 2020 and Beyond. International Journal of Advanced Computer Science and Applications(ijacsa).

[45] Hellman R. 2009. Barries to organizational interoperability – The Norwegian case. IADIS International Conference e-Society.