SEMIoTICS

Deliverable D4.4
Semantic Interoperability Mechanisms for IoT (first draft)

Deliverable release date	Initial 31.05.2019, revised 25.11.2019
Authors	1. Eftychia Lakka, George Hatzivasilis, Nikolaos Petroulakis (FORTH), 2. Darko Anicic, Arne Broering (SAG) 3. Mirko Falchetto (ST) 4. Lukasz Ciechomski (BS)
Responsible person	Eftychia Lakka (FORTH)
Reviewed by	Darko Anicic (SAG), Konstantinos Fysarakis (STS), Juan David Parra (UP), Jordi Serra (CTTC), Mirko Falchetto (ST), Kostas Ramantas (IQU), Urszula Rak (BS)
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 revised
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 14
3 Background and Related Work 16
3.1 The Basis for Semantics 16
3.2 Semantic Models for Smart Objects 18
3.3 Ontologies 21
3.3.1 Overview 21
3.3.2 Ontologies in Healthcare Domain 21
4 SEMIoTICS Semantics 23
4.1 Data Mappings in SEMIoTICS 23
4.2 Ontologies in SEMIoTICS 23
5 Semantic Mediator mechanisms 24
5.1 Ontology Alignments & Data Transformation Techniques 24
5.1.1 Data Mapping 24
5.1.2 Ontology Alignment 25
5.2 Semantic Reasoning 28
6 SEMIoTICS Intergation Approach with Other IT Platforms 30
6.1 SEMIoTICS Interfaces 30
6.2 Integration with FIWARE 30
7 End-To-End Interoperability Verification Mechanisms 36
7.1 Translation of Recipes into SPDI Patterns 36
7.2 Interoperability Patterns 40
8 Semantic interoperability mechanisms - Implementation 42
9 Validation 45
9.1 Related Project Objectives and Key Performance Indicators (KPIs) 45
9.2 SEMIoTICS Interoperability Requirements 45
10 Conclusions 47

10.1 Summary 47
10.2 Future Work 47
References 48

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
OWL	Web Ontology Language
P2P	Peer to Peer
QoS	Quality of Service
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
SSN	Semantic Sensor Network
SWE	Sensor Web Enablement
TD	W3C Things Description
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 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 shared vocabulary. Thus, the meaning of the data is exchanged along 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 particular 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 are 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 the Chapter 6.

Then, a common and generic Application Programming Interface (API) will be established in D4.6 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 real-time

[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/

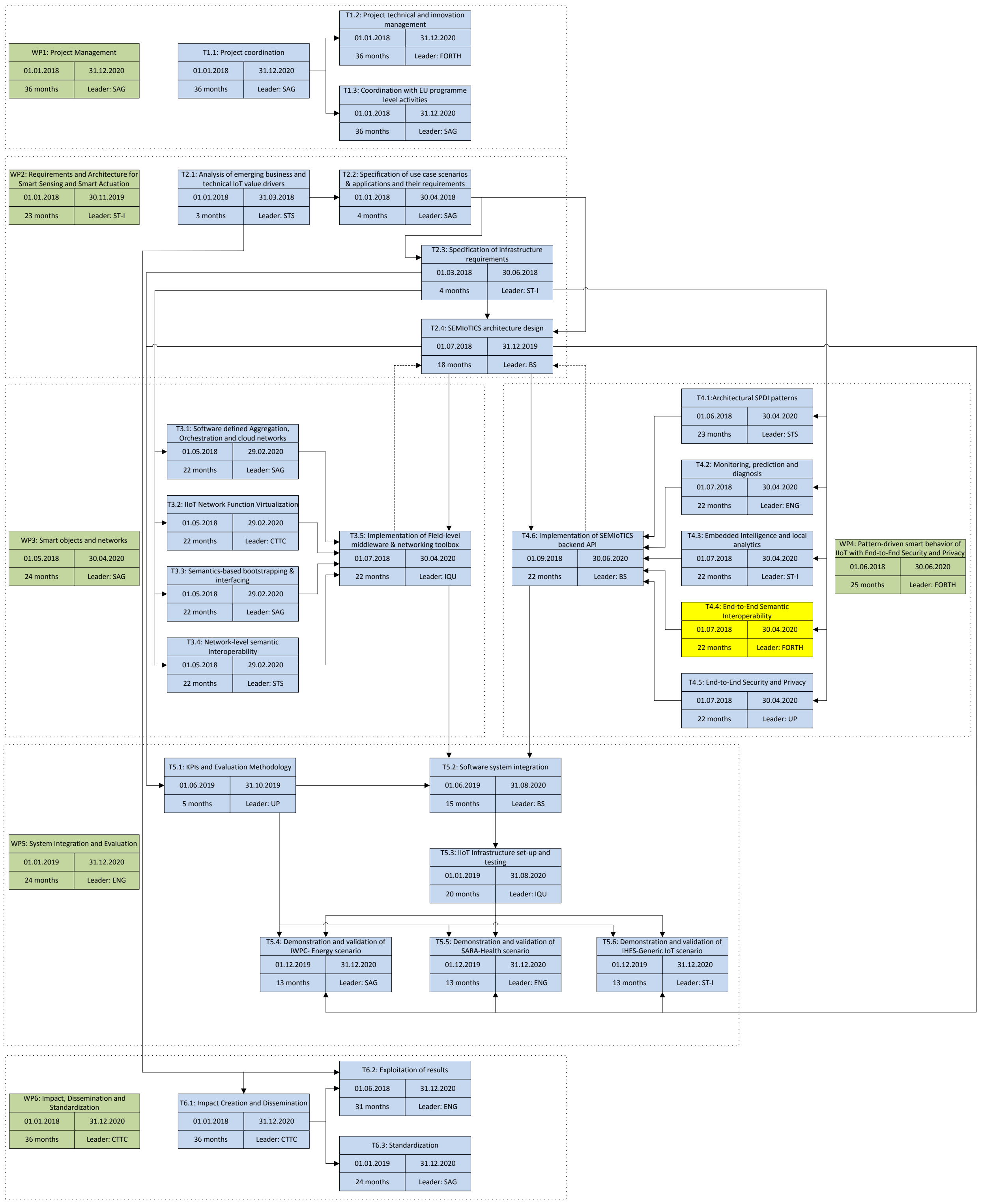
[5] https://github.com/iot-schema-collab & http://iotschema.org/

management of the underlying IoT ecosystem (see D4.1) 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 proposed implementation for of semantic interoperability mechanisms is presented. Finally, in Chapter 9 the validation of the project objectives, key performance and interoperability mechanisms are also presented.

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 and the semantic ontologies.
- **Chapter 5** defines the implementation of the concrete semantic interoperability mechanism.
- **Chapter 6** provides the interfaces for integrating with the SEMIoTICS framework and sketches the first steps towards the integration/interplay with the FIWARE platform.
- **Chapter 7** details the verification methods and details the related annotations for the Security, Privacy, Dependability, and Interoperability (SPDI) patterns with the RECIPE tool.
- **Chapter 8** outlines the application of the proposed process in the scenario of end-to-end interoperable service composition.
- **Chapter 9** 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 10** summarizes 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
WP3: Smart objects and networks
01.05.2018
30.04.2020
24 months
Leader: SAG
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
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
WP4: Pattern-driven smart behavior of IIoT with End-to-End Security and Privacy
01.06.2018
30.06.2020
25 months
Leader: FORTH
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
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
T6.2: Exploitation of results
01.06.2018
31.12.2020
31 months
Leader: ENG
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
T6.3: 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 a system to work with or use the components of another system. 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 main 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 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 D3.3.

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, it is also the exchange of web data, due to enable 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 realize generic IoT solutions at a global scale. Particularly, there are some high-level interoperability issues that 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 the majority of them.

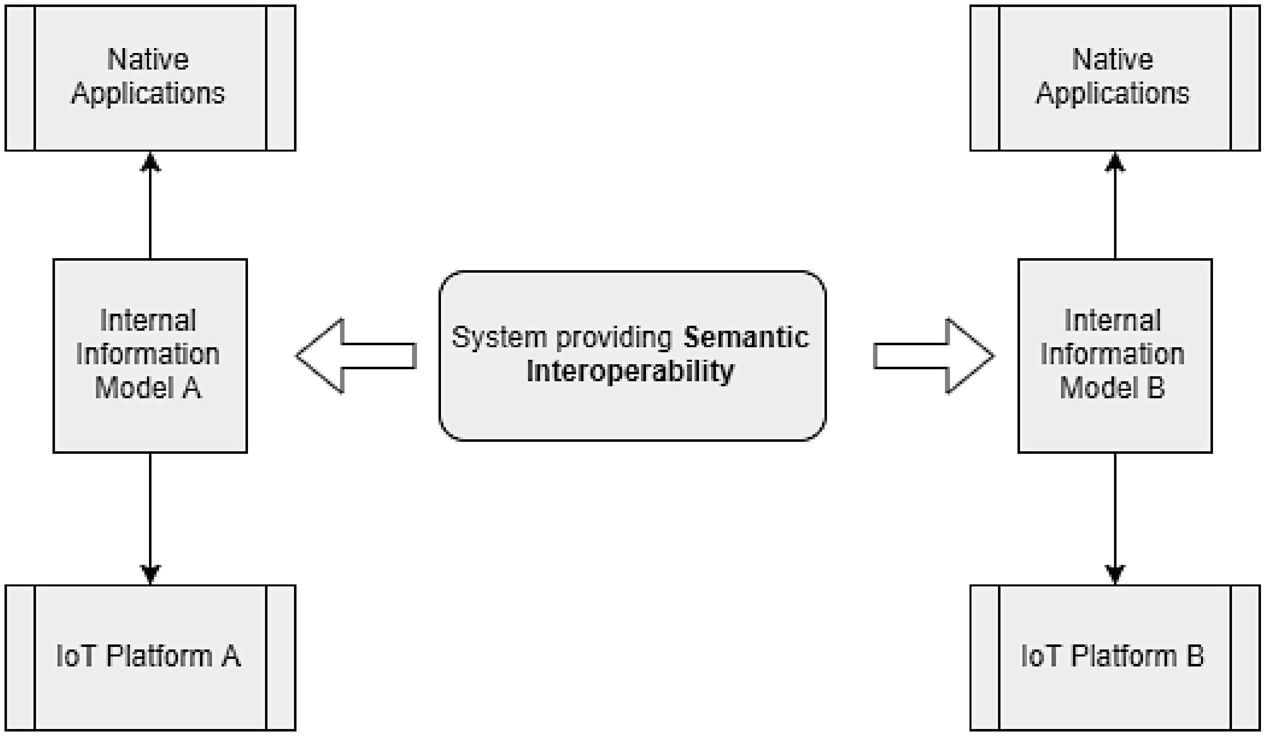

Native Applications
Native Applications
Internal Information Model A
System providing Semantic Interoperability
Internal Information Model B
IoT Platform A
IoT Platform B

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 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

As a motivating example, we consider the 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 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 sensors in the rooms. Three types of sensory devices are modelled (Figure 2**Error! Reference source not found.**):

- 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 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/

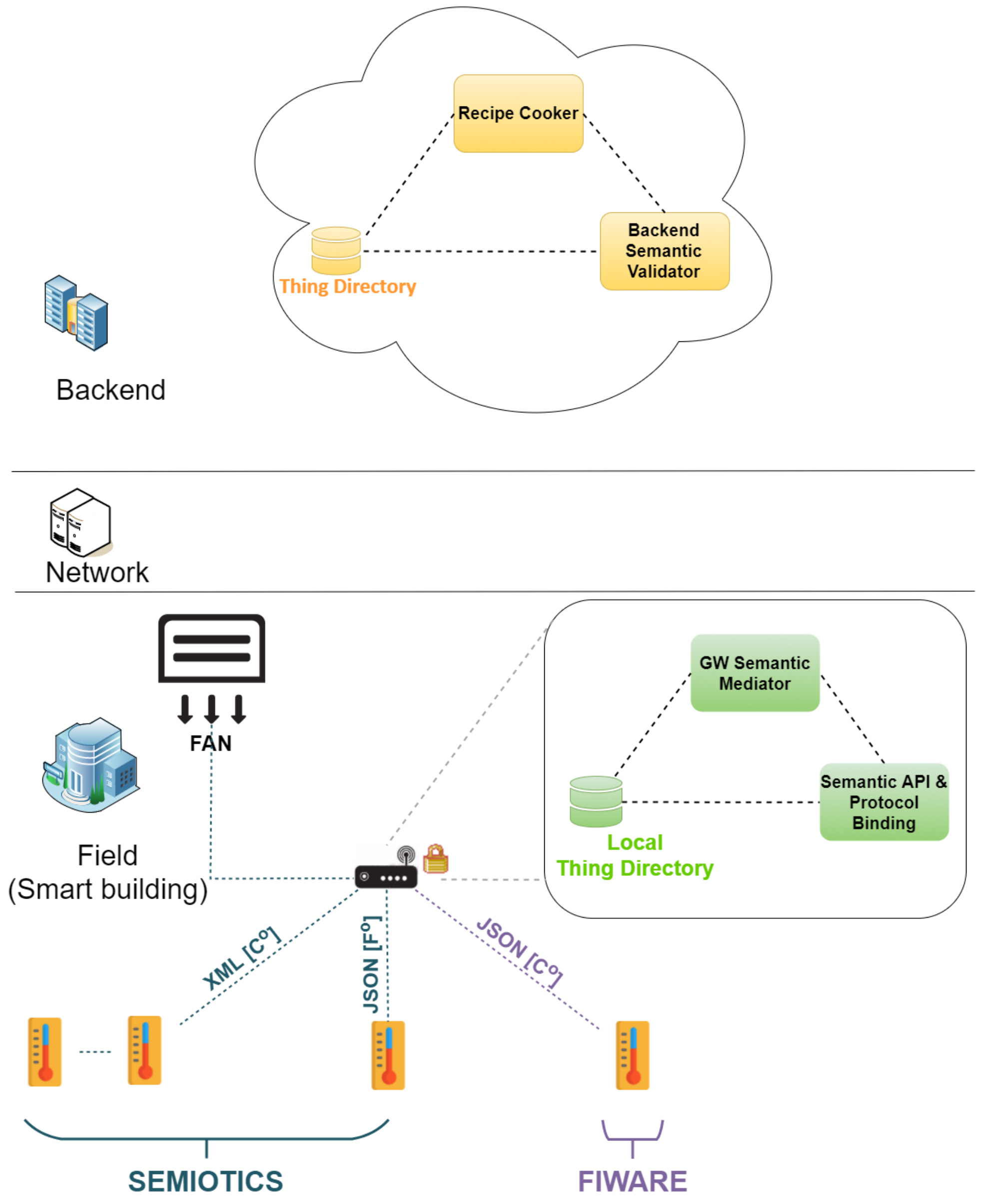


Figure 2 The Smart Sensing Interoperability Scenario

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 2.4) 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^{o} and uses the XML format to transmit data (Figure 3). However, the actuator (fan) requires data in F^{o} 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).

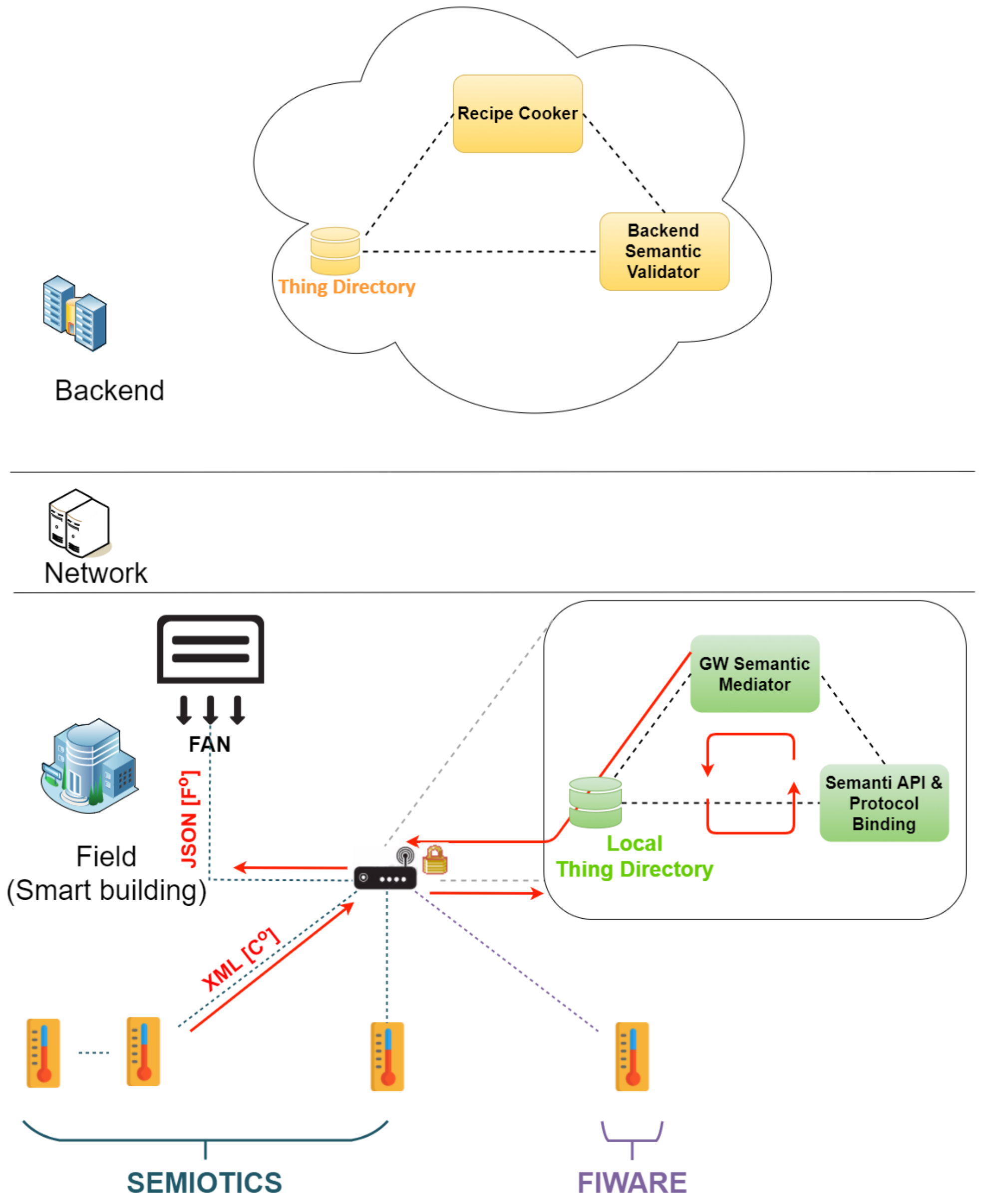


Figure 3 horizontal scenario – In field layer

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 to 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).

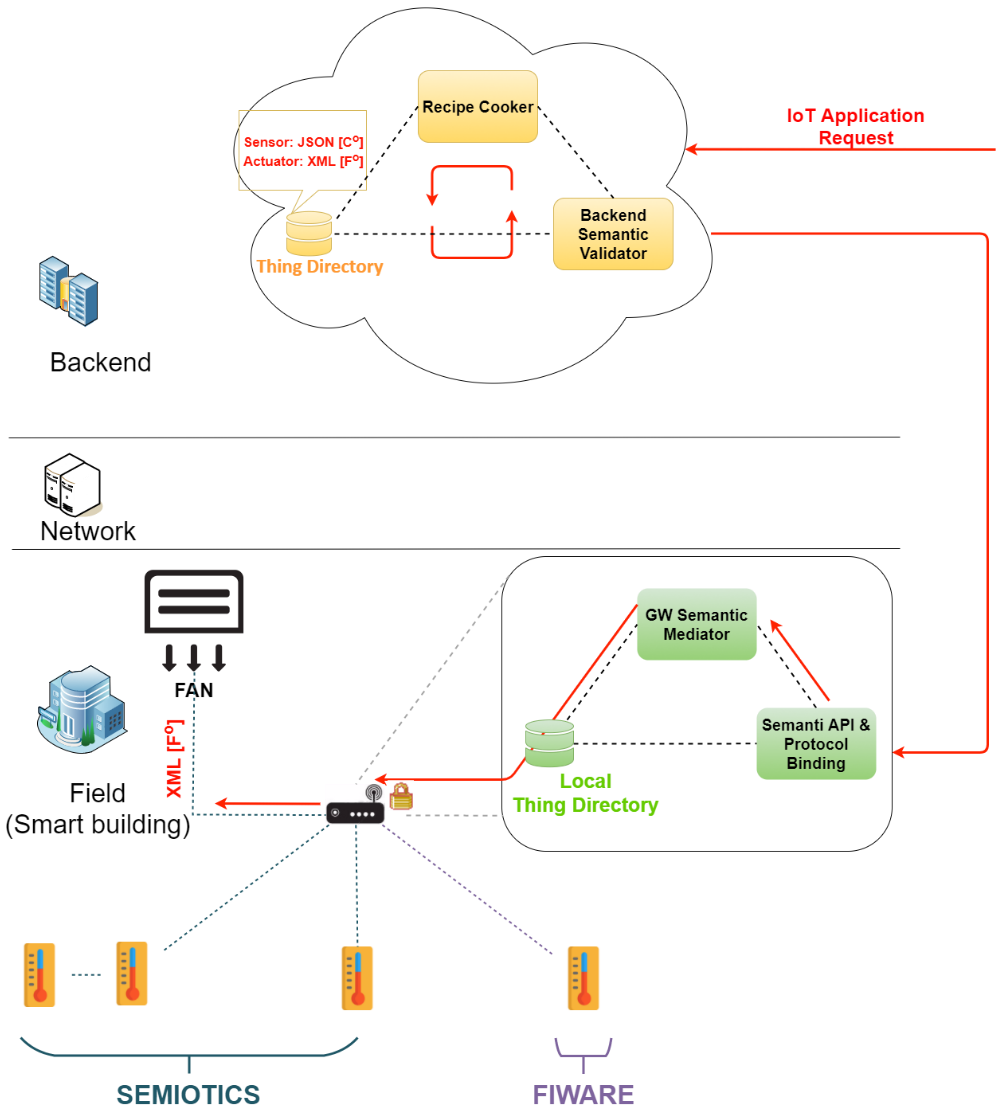


Figure 4 vertical SCENARIO – IN backend LAYER

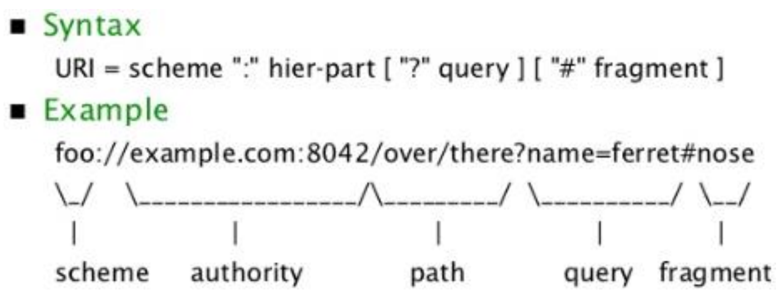
In cases where the abovementioned communication between the field and the backend must be **encrypted**, the semantic functionality is performed in the cloud by the end-point that decrypts and processes the data.

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

A **Uniform Resource Identifier (URI)** provides a simple and extensible means for identifying a resource (RFC 3986[7]). 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.

```
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 describes a technician that can repair the sensors in the motivating example (European sensors that measures 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 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":
"ltGu17tYQWE0MsI3dmqXv2POJF1MykjTOXtn2A5EtMXdmrSmRKUCtvv8jOTmJxJTBxAIgZR0nDfwEUj+
DSa2SUA41NRK6+plPGsj5fvCCFK5rwM8KFVPTDP8CfBG5CSsQW3faf3oudu5yjNCbxuHUFNvL6UMkD
EeyP1MCcGOR0s="
  }
}
```

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” 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 particular 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 for example:

- 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 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.

[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

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. 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 [14], [15].

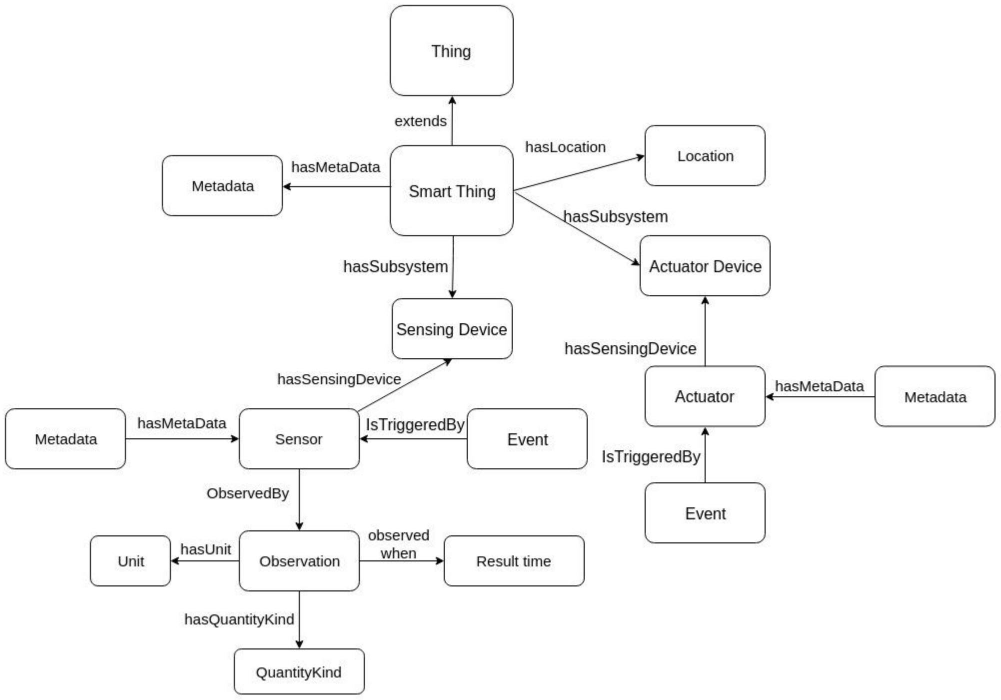


Figure 6: extendED model based on SSN ontology

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 [16]. 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 [17].

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) [18] and INTER-IoT [19], 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 [20], 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) [21] and the MyOntoSens [22].

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 [23]. 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 [24] that enable universal forms of knowledge representation integrate heterogeneous information, answer complex queries, and pursue data integration and knowledge sharing in healthcare [25].

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 few of the standards currently used for interoperability in the semantic web [23].

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 [23].

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 basic ontology for any medical	US National Library of Medicine
ICD-10	Classification of diagnosis codes; is 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 design for the IoT domain by the World Wide Web Consortium (W3C) community is 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 particular brownfield semantic standard into another IIoT standard. The SEMIoTICS IoT Gateway is able to utilize the mapping knowledge and thus get 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 ease 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 D3.3).

5 SEMANTIC MEDIATOR MECHANISMS

This chapter describes in more details 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

- 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 [26]. Additionally, it can be used to describe other network constructs, such as services, protocols, policies and customers [27]. 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 [28] and RESTCONF [29] 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 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.

- Development Function:

An alternative solution for mapping between XML and JSON is the implementation of 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).

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

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

5.1.2 Ontology Alignment

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.

- 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 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 4 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.

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

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

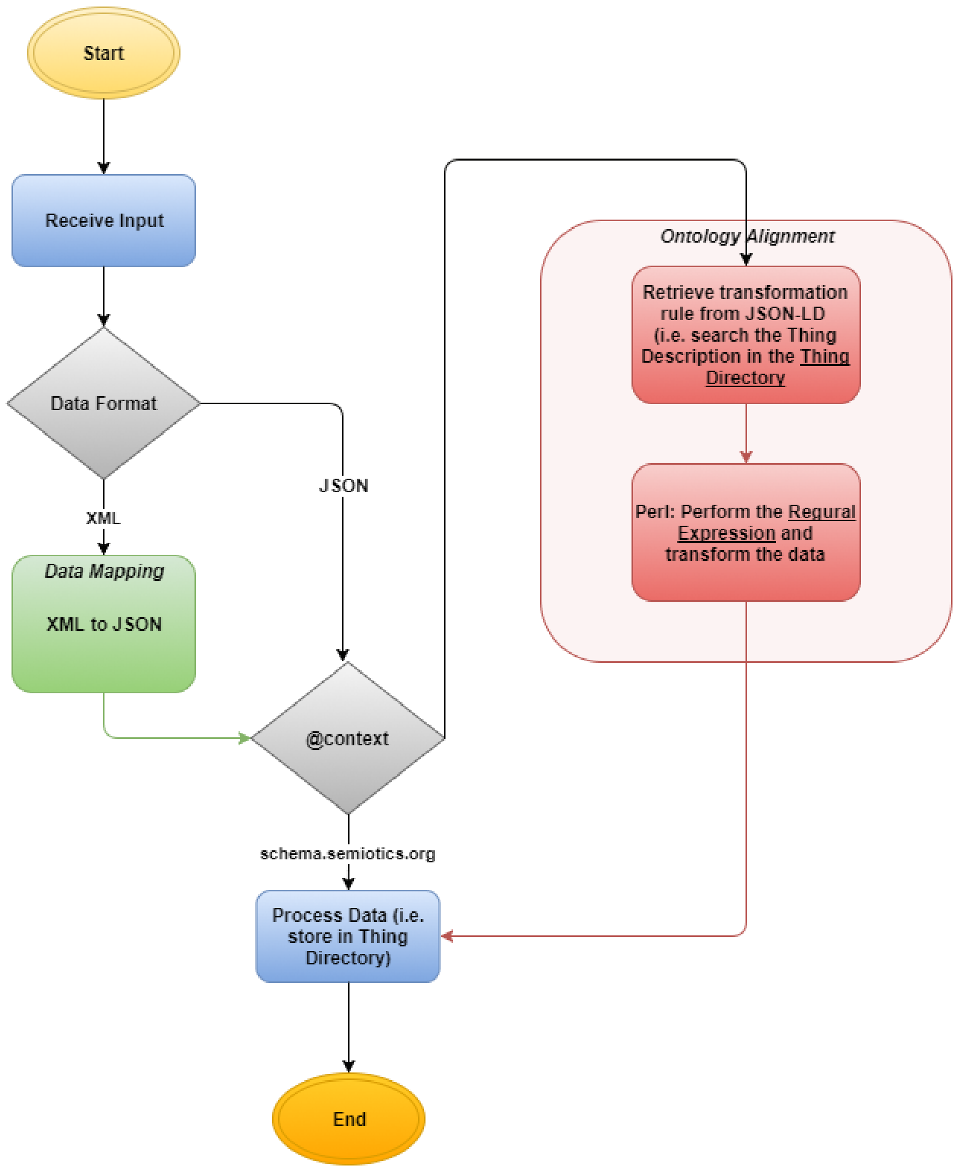

Start
Receive Input
Data Format
XML
JSON
Data Mapping
XML to JSON
@context
schema.semiotics.org
Ontology Alignment
Retrieve transformation rule from JSON-LD (i.e. search the Thing Description in the Thing Directory)
Perl: Perform the Regural Expression and transform the data
Process Data (i.e. store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

Figure 7 SEMANTIC INTEROPERABILITY PROCEDURE: DATA MAPPING AND ONTOLOGY ALIGNMENT WITH JSON RE – ACTIVITY DIAGRAM

- 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 5). From the developing point of view [30]:

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

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.
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 5 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 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.

- Extension Recipe Model – Adaptor Nodes in Node-RED Platform:

The Node-RED[17] 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 in 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 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 architecture. At the moment, the first added nodes that have already implemented, provide the functionality of temperature value transformation from the Celsius to the Fahrenheit scale and vice versa. More Adaptor Nodes will be added in the future to address the SEMIoTICS requirements. All the final extra nodes will be described in the next version of the present deliverable (D4.11).

[17] https://nodered.org/

5.2 Semantic Reasoning

The term of semantic alignments has been used in the literature to adapt and transform the data to the declared ontologies [31]. Depending on the expressiveness of the ontologies, reasoning engines can further infer associations and links into the data. Hence, 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 [32]. 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[18], Pellet[19], RacerPro[20], HermiT[21], RIF4J[22] and Fact++[23] are based on different rule languages and can give support for ontologies and OWL. On the other hand, there are some reasoners which support SWRL[24] and RIF[25] rule languages and others have implemented their own human readable rule syntaxes.

Moreover, there are sensor-based linked open rules [33] for sharing and reusing rules in IoT applications based on Jena rules syntax. Besides that, another methodology [34] is used to link and reuse rules on the Web which is expressed as SPARQL queries. Also, in [35] 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:

```
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 6 SPARQL Query for Selecting all the names and email addresses of RDF data that has the type foaf:Person

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

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

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

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

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

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

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

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

The above example builds on the friend-of-afriend 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 [36], SPARQL-LD[26], 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 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 7 SPARQL-LD Query for Selecting the expert Technicians for a specific sensor

Another solution, which is based on SPARQL, is the Thingweb Directory[27]. 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 8) 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[28].

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 8 SPARQL Filter in Thingweb Directory

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

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

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

6 SEMIOTICS INTERGATION APPROACH WITH OTHER IT PLATFORMS

This chapter presents the means of integrating SEMIoTICS with other IoT platforms. More specifically, we describe the interacting interfaces and how they interplay with the smart sensing setting of the FIWARE project.

6.1 SEMIoTICS Interfaces

The SEMIoTICS platform components are able to expose dedicated APIs which are visible outside the platform thanks to a router functionality embedded in the backend platform itself. Each component is able to 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). One the other hands component creator is able to 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.

6.2 Integration with FIWARE

As it was mention 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.

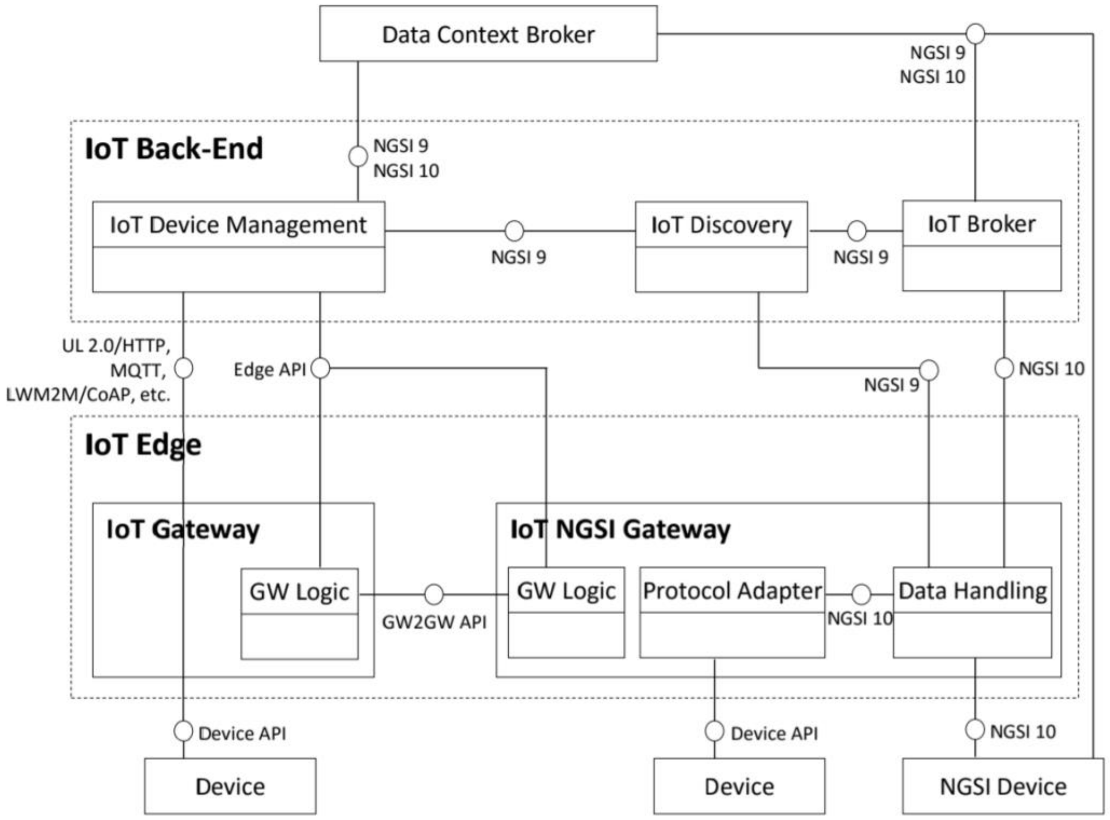


Figure 8 FIWARE IOT SERVICES ENABLEMENT ARCHITECTURE

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 8). Moreover, FIWARE proposes a semantics built upon iot.schema.org, which is the target semantic core for SEMIoTICS. (Data in this core semantics are defined in JSON Schema[29]). This allows the use of the open source FIWARE semantics (schema.fiware.org) instead of working with 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 interface[34] [35]. The NGSI context data has been presented in the image below (Figure 9). 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.

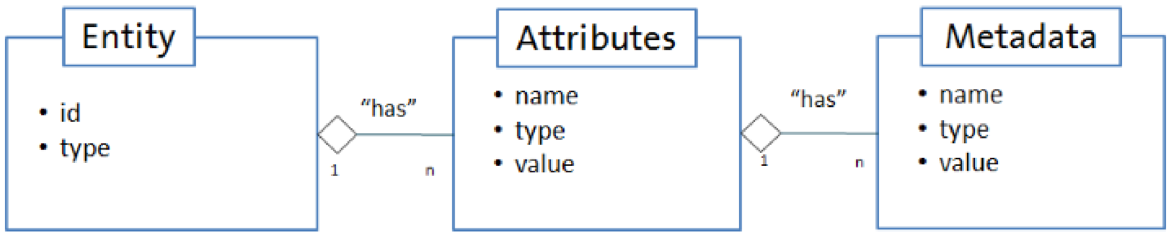


Figure 9 NSGI CONTEXT DATA [34] [35]

[29] http://json-schema.org/

[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/

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

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

FIWARE provides few GEs related to semantic interoperability, that can be applied to the SEMIoTICS project. They 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 [37] 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 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.

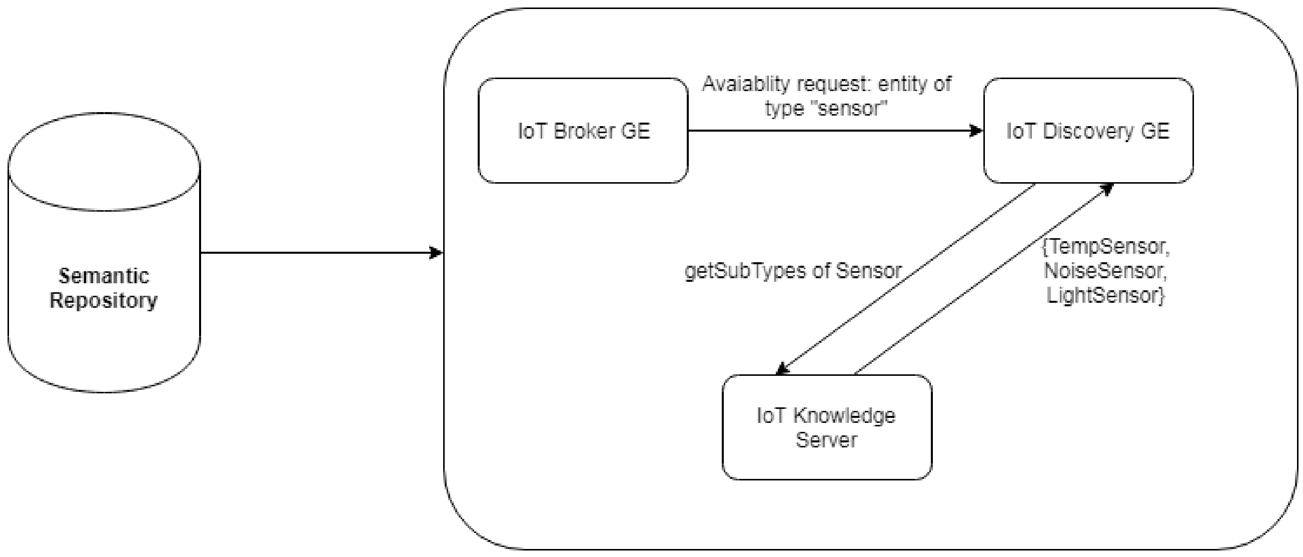


Figure 10 IOT KNOWLEDGE SERVER, IOT BROKER AND IOT DISCOVERY INTEROPERATION EXAMPLE (based on [49])

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.

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

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

IoT Knowledge Server (from the IoT Broker GE) which, as it was 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).

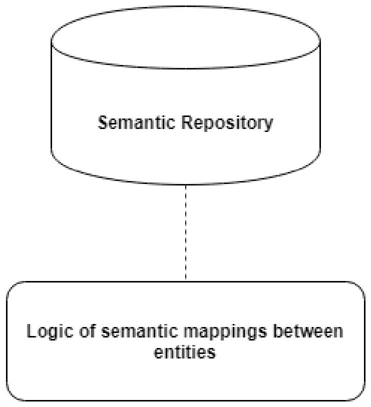


Figure 11 LOGIC OF SEMANTIC MAPPINGS BETWEEN COMPONENTS AS A SEPARATE COMPONENT DEFINING TRANFORMATION MAPPINGS BETWEEN ENTITES

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 component responsible for the logic of semantic mappings has been presented in the Figure 11.

The following JSON-LD code 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 9 FIWARE Temperature JSON-LD Example

Moreover, the D4.11 will include the description of specific scenarios involving SEMIoTICS framework and FIWARE platform to achieve sematic interoperability connection between them and the possibility of interaction with further FIWARE’s GEs. Finally, more details will be given for the interaction of SEMIoTICS with not only

MindSphere[38], based on the initial description of the validation between SEMIoTICS field device and corresponding MindSphere Asset that has already described in D3.3, but also ARES-H, in order to achieve interoperability with these specific targeted external IoT platforms.

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

7 END-TO-END INTEROPERABILITY VERIFICATION MECHANISMS

This chapter analyses the verification mechanisms that ensures 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**.

7.1 Translation of Recipes into SPDI Patterns

Disclaimer: This section is based on a publication accepted at the upcoming EUCNC conference [38].

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 the Figure 12 below.

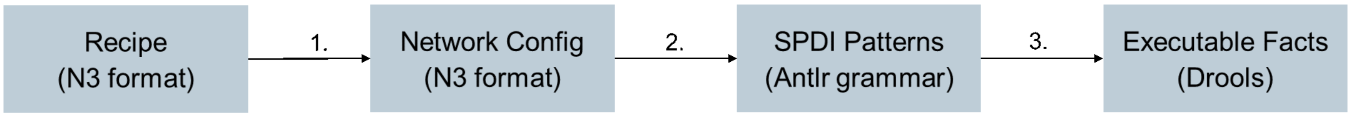


Figure 12 TRANSLATIONS FROM RECIPES TO EXECUTABLE RULES

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 about these 2 models (Recipe and Network model) and their transformation is shown in Figure 13. They are defined as triples in the RDF format and serialized in the N3 format.

On the left side of Figure 13, the model to define abstract IoT compositions as *recipes* is shown. This model is based on our previous work [39], [40]. 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.

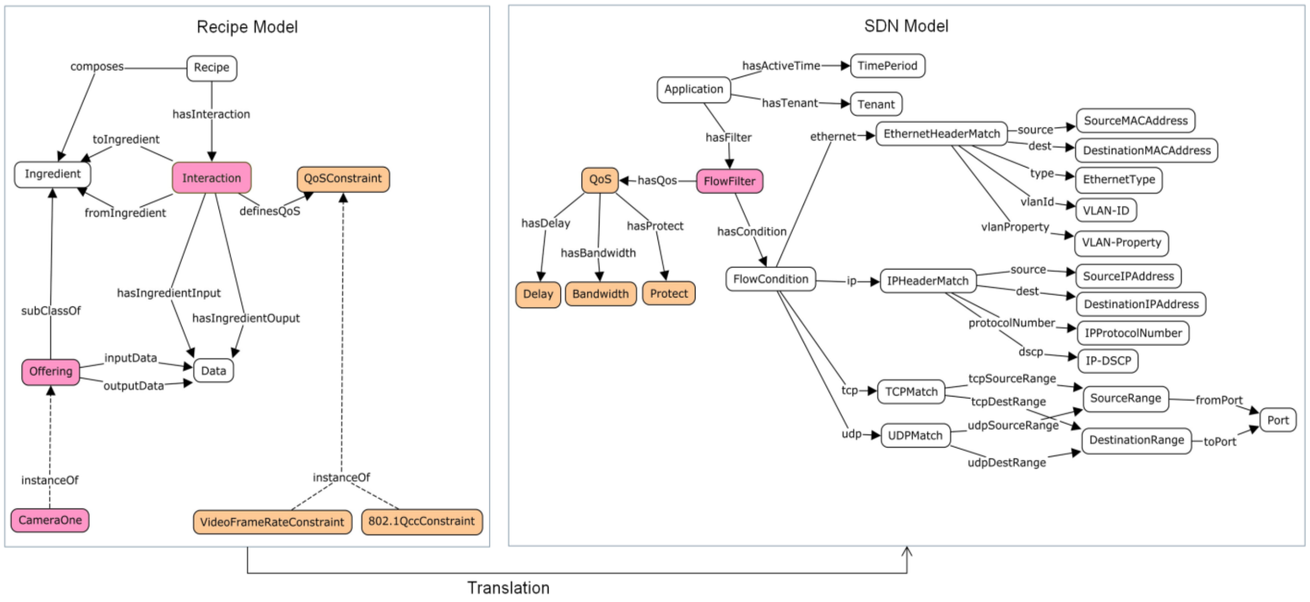


Figure 13 TRANSFORMATION FROM RECIPE TO NETWORK CONFIGUIRATION [41]

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 13. 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 [42]. 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 14 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.

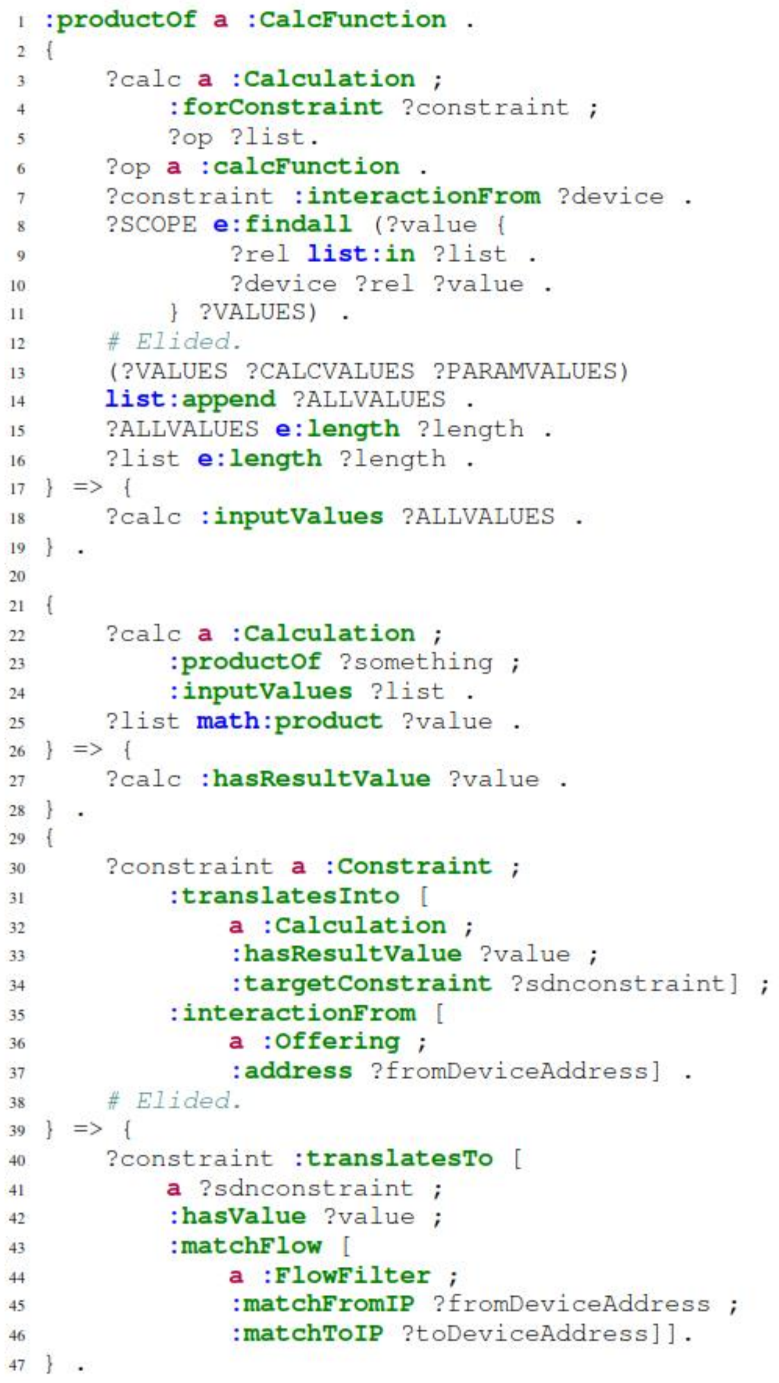
```
:CameraOne a :Offering ;
    :resolutionX 1024 ;
    :resolutionY 786 ;
    :efficiency "0.8"^^xsd:float ;
    :address "192.168.178.25" .

: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 14 DEVICE AND CONSTRAINT DEFINITION IN N3 FORMAT

Figure 15 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 15 TRANSLATION RULES FOR A CAMERA FRAME-RATE CONSTRAINT

Having the above described translation rules available, we have covered the translation step (1.) in Figure 15. 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[39] format. An example output that represents such a SPDI pattern in context of the video camera example above is shown in Figure 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)
```

Figure 16 SPDI PATTERN

7.2 Interoperability Patterns

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, syntactic, semantic, and organizational issues, usually referenced as interoperability layers [43]. Similarly, four levels of interoperability are included in the SEMIoTICS framework: technological, syntactic, semantic and organizational interoperability (see Deliverable 4.1). 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 below). Although, it is worthwhile to note that the organizational interoperability is the most difficult challenge of all the interoperabilities. These complexities are associated not only with formal agreements on collaboration but also with practical approaches to organizational interoperability [44].

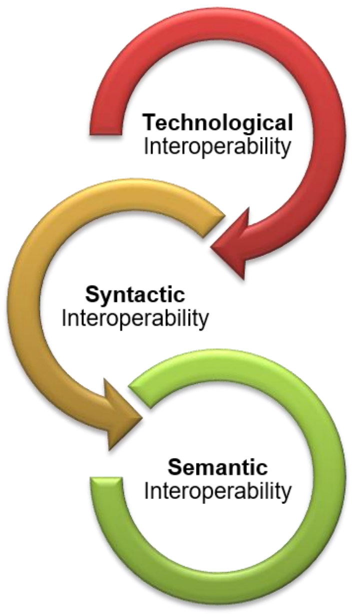

Technological Interoperability
Syntactic Interoperability
Semantic Interoperability

Figure 17 PATTERN DEFINITION - NTEROPERABILITY LAYERS IN SEMIoTICS FRAMEWORK

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

Based on the initial version of the language for specifying SPDI patterns in D4.1, a set of patterns rules for interoperability are proposed for the SEMIoTICS system. Specifically, the technological, the syntactic and the semantic pattern definition are composed the pattern specification rule for interoperability issue in SEMIoTICS. Following the structure of the Drools rules engine, the above specification is described in CODE 10.

```
rule "Interoperability"
when
        $A: Placeholder($input : operation.inputs,
        $intData : parameters.outputs)
        $B: Placeholder(parameters.inputs == $intData,
        $output : parameters.outputs)
        $ORCH: Link(firstActivity == $A, secondActivity == $B)
        $OP: Req( propertyName == "Interoperability",
        subject == $ORCH, satisfied == false)
        $SP: PropertyPlan (properties contains $OP)
then
        PropertyPlan newPropertyPlan = new PropertyPlan($SP);
        newPropertyPlan.removeRequirement($OP);

        Req Technological = new Req($OP,"Technological",ORCH);
        newPropertyPlan.getProperties().add(Technological);
        insert(Technological);
        Req Syntactic = new Req($OP,"Syntactic",ORCH);
        newPropertyPlan.getProperties().add(Syntactic);
        insert(Syntactic);
        Req Semantic = new Req($OP,"Semantic",ORCH);
        newPropertyPlan.getProperties().add(Semantic);
        insert(Semantic);
        insert(newPropertyPlan);
end
```

CODE 10 Interoperability Pattern Rule

8 SEMANTIC INTEROPERABILITY MECHANISMS - IMPLEMENTATION

The first version of SEMIoTICS deliverable D4.4 presents the development of data transformation techniques and validation mechanisms to ensure end-to-end semantic interoperability. In the next version of this deliverable (D4.11) this methodology will be implemented. However, this chapter outlines the SEMIoTICS architecture elements that are responsible for resolving semantic differences and provide a short analysis of the implementation of them. The components of the SEMIoTICS architecture, which are involved in semantic interoperability mechanisms, are demonstrated in Figure 18. Particularly, the Backend Semantic Validator, the Recipe Cooker, the Thing Directory from the backend layer and the Semantic API & Protocol Binding, the GW Semantic Mediator and the Local Thing Directory from field layer are the above-mentioned components.

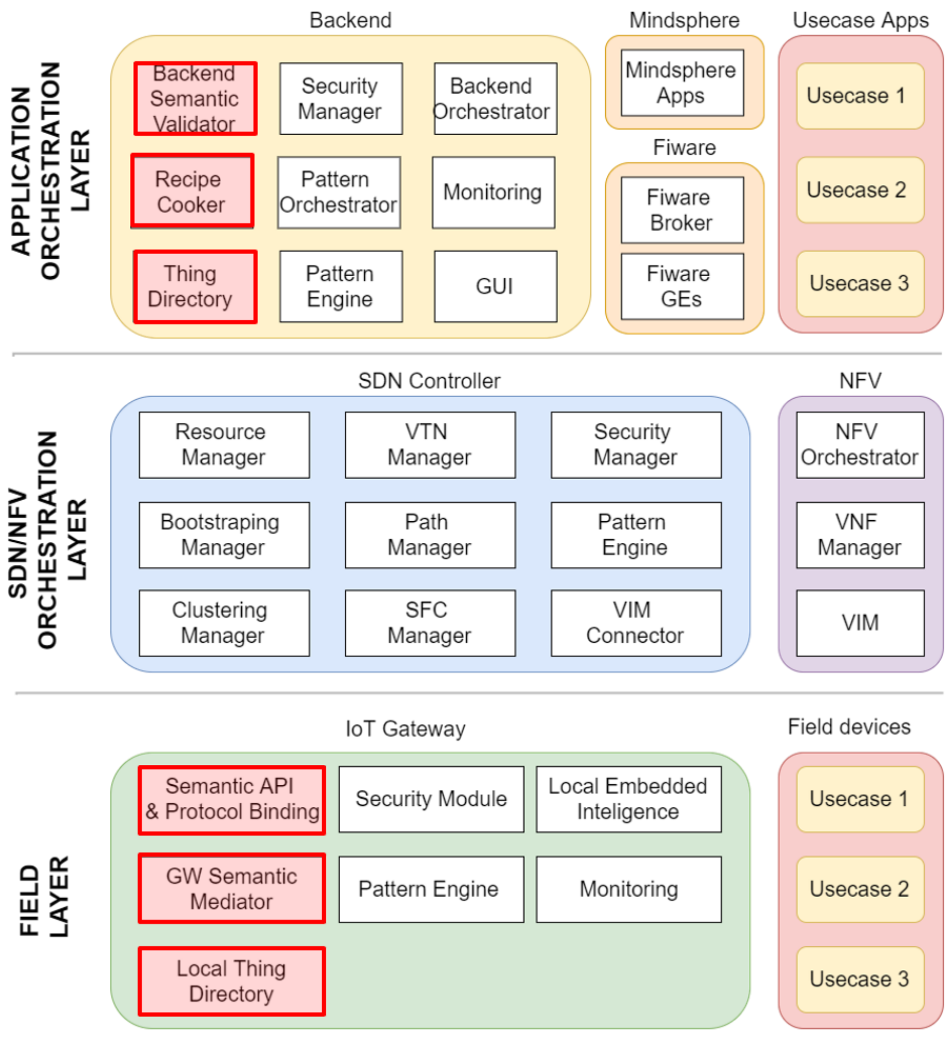


Figure 18 SEMIoTICS ARCHITECTURE - SEMANTIC INTEROPERABILITY MECHANISMS COMPONENTS

In Figure 19, a sequence diagram depicts the procedure of the semantic interoperability mechanisms between the application orchestrator layer and the field layer. As stated in Section 2.2, the aim is the connection between two Things (Sensor and Actuator), by an IoT application which sends request in the backend. This motivated scenario can be applicable in UC2 and UC3 using not only the specific sensor (thermometer) but also any other sensor (e.g. for pressure, humidity, light) that aims to interact with an actuator.

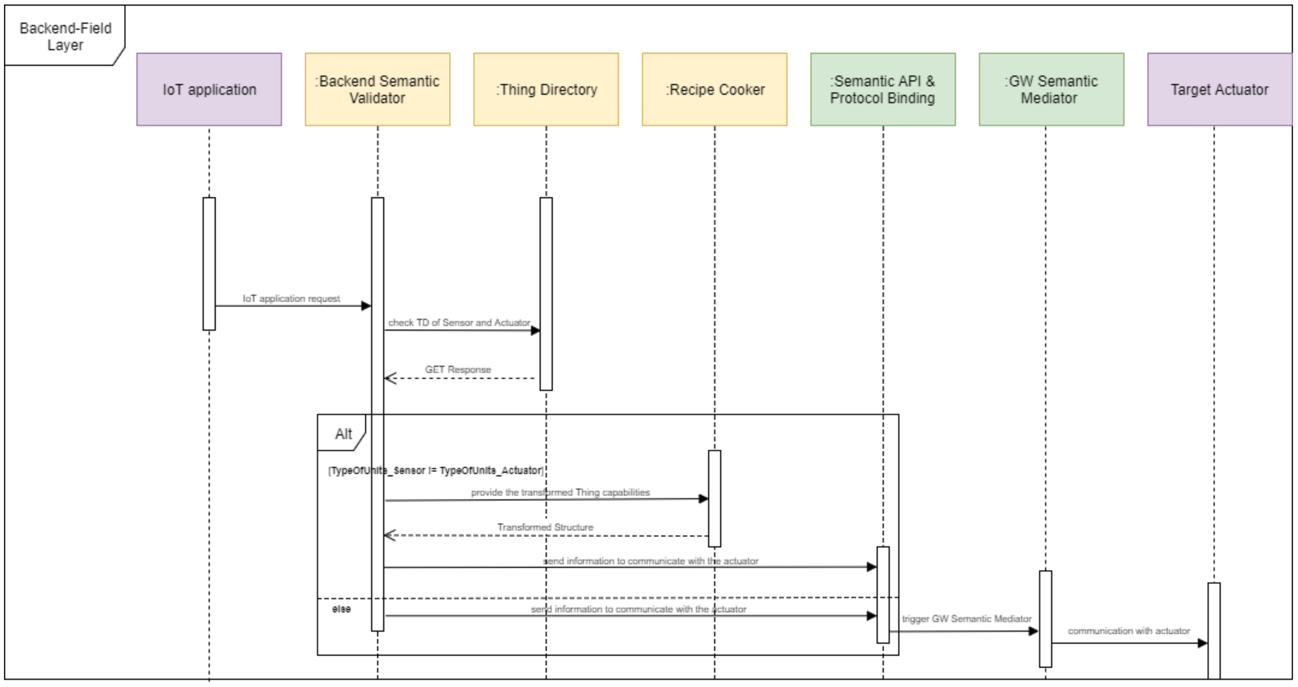

Backend-Field Layer
IoT application
:Backend Semantic Validator
:Thing Directory
:Recipe Cooker
:Semantic API & Protocol Binding
:GW Semantic Mediator
Target Actuator
IoT application request
check TD of Sensor and Actuator
GET Response
Alt
[TypeOfUnits_Sensor != TypeOfUnits_Actuator]
provide the transformed 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 19 SEQUENCE DIAGRAM FOR SEMANTIC INTEROPERABILITY MECHANISMS

In fact, the BSV receives the request from the IoT application, using the grpc[40] 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[41] (Figure 20). 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 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.

[40] https://grpc.io/

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

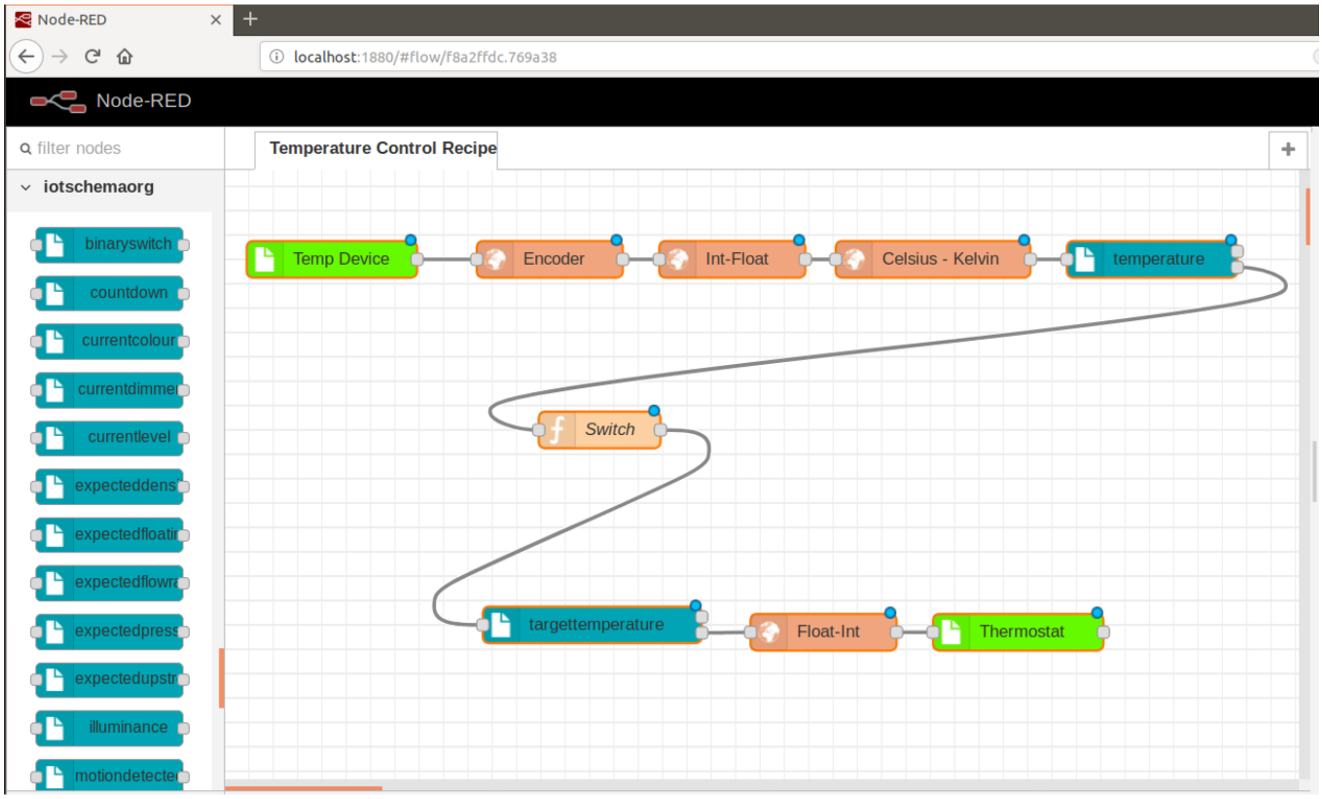


Figure 20 TEMPERATURE CONTROL EXAMPLE (GITHUB.COM)

9 VALIDATION

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

9.1 Related Project Objectives and Key Performance Indicators (KPIs)

The objectives of the related T4.4 and their mapping to D4.4 content are summarized in the following table.

Table 4 Task's Objectives

T4.4 Objectives	**D4.4 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
SMs • 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	2, 6, 8

The overall deliverable constitutes the initial contribution towards fulfilling the project's requirements regarding **SEMIOTIC'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 T4.4 and the mapping measurement of the corresponding KPIs will be 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 in T4.1.

9.2 SEMIoTICS Interoperability Requirements

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

Table 5 Task's Objectives

Requirements (D2.3)	**Description**	**D4.4 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	Section 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
R.UC1.12	Standardized semantic models for semantic-based engineering and IIoT applications SHALL be utilized	Section 4.2
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
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.	Section 3.2
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).	Section 2.2
R.UC3.1	IoT Sensing unit shall be able to embed environmental (e.g. temperature, pressure, humidity, light) and inertial sensors (accelerometer, gyroscope).	Section 2.2
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.	Section 5.1.1

10 CONCLUSIONS

10.1 Summary

This deliverable presents the landscape for accomplishing semantic interoperability in the IoT domain. To do so, the state-of-the-art approaches are reviewed (Chapters 2-3), 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 provide the required common representation and meaning of data (Chapter 5).

Moreover, this deliverable describes the initial thoughts towards the integration of SEMIoTICS with other IoT platforms (Chapter 6). Then, the translation of Recipes into SPDI Patterns and patterns are analysed (Chapter 7). The overall process is further analysed in D4.1.

An example of semantic interoperation is described involving smart temperature sensing. The integration process will be facilitated by the SEMIoTICS API that will be detailed in D4.6. Particularly, we demonstrate the application of various interoperability methods in a smart sensing scenario, involving semantic interoperability from the field to the backend system (Chapter 8).

10.2 Future Work

To sum up, initial mechanisms and designs that are responsible for resolving semantic differences, in this deliverable, will serve as inputs to the second deliverable of Task 4.4 (D4.11). Particularly, the final development of data transformation techniques and validation mechanisms to ensure end-to-end semantic interoperability in SEMIoTICS framework will be included in D4.11. Also, in the final version, a set of patterns rules for interoperability, which are proposed for the SEMIoTICS system (D4.1), will be described in further details. Moreover, a full set of Adaptor Nodes, which will be responsible for resolving, if possible, conflicts among the semantic annotations, will be described and presented in the D4.11. Finally, further analysis for the validation of semantic interoperability between the SEMIoTICS framework and 3 IoT platforms (FIWARE, MindSphere and OpenHAB) and the semantic descriptions for all the types of smart objects (**KPI-2.1**), which are necessary for the SEMIoTICS usage scenarios, will be provided by the D4.11.

REFERENCES

[1] Hatzivasilis, G., Askoxylakis, I., Alexandris, G., and Fysarakism, 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-AC$_4$), 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] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.
[22] Bajaj, G., et al., 2017. A study of existing ontologies in the IoT-domain, HAL Archives, hal-01556256, pp. 1-24.
[23] 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].
[24] Bicer V., Laleci G., Dogac A., Kabak Y., 2005. Artemis message exchange framework: semantic interoperability of exchanged messages in the healthcare domain. ACM SIGMOD.
[25] Nardon F.B., Moura L.A., 2004. Knowledge sharing and information integration in healthcare using ontologies and deductive databases. MEDINFO 2004, pp. 62-66.
[26] Bjorklund M., 2016. The YANG 1.1 Data Modeling Language, August 2016. RFC 7950.
[27] 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.
[28] Enns R., et al, 2011. Network Configuration Protocol (NETCONF), IETF RFC 6241.
[29] Bierman A. et al., 2017. RESTCONF Protocol, IETF RFC 8040.
[30] 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].
[31] 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.
[32] 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.
[33] 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.
[34] 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.
[35] 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).
[36] 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.
[37] IoT Broker: Available at: https://www.slideshare.net/FI-WARE/iot-broker [Accessed 21 Feb. 2018].
[38] 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.
[39] 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.
[40] 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.
[41] 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.
[42] 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.
[43] 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).
[44] Hellman R. 2009. Barries to organizational interoperability – The Norwegian case. IADIS International Conference e-Society.