Symbiosis of smart objects across IoT environments

688156 - symbIoTe - H2020-ICT-2015

D2.4 – Revised Semantics for IoT and Cloud resources

The symbIoTe Consortium

Intracom SA Telecom Solutions, ICOM, Greece
Sveučiliste u Zagrebu Fakultet elektrotehnike i računarstva, UNIZG-FER, Croatia
AIT Austrian Institute of Technology GmbH, AIT, Austria
Nextworks Srl, NXW, Italy
Consorzio Nazionale Interuniversitario per le Telecomunicazioni, CNIT, Italy
ATOS Spain SA, ATOS, Spain
University of Vienna, Faculty of Computer Science, UNIVIE, Austria
Unidata S.p.A., UNIDATA, Italy
Sensing & Control System S.L., S&C, Spain
Fraunhofer IOSB, IOSB, Germany
Ubiwhere, Lda, UW, Portugal
VIPnet, d.o.o, VIP, Croatia
Instytut Chemii Bioorganicznej Polskiej Akademii Nauk, PSNC, Poland
NA.VI.GO. SCARL, NAVIGO, Italy

For more information on this document or the symbIoTe project, please contact:
Sergios Soursos, INTRACOM TELECOM, souse@intracom-telecom.com

Document Control

Title: Revised Semantics for IoT and Cloud resources

Type: Public

Editor(s): Michael Jacoby

E-mail: michael.jacoby@iosb.fraunhofer.de

Author(s): Michael Jacoby (IOSB), João Garcia (UW), Antonio Paradell (ATOS/WL), Luca de Santis (NAVIGO), Matteo Pardi (NXW), Karl Kreiner (AIT), Szymon Mueller (PSNC), Svenja Schröder (UNIVIE), Marcin Płóciennik (PSNC), Ivana Podnar Žarko (UNIZG-FER)

Doc ID: D2.4-v1.2

Amendment History

Version	Date	Author	Description/Comments
v0.0	01/05/2017	Michael Jacoby (IOSB)	TOC created
v0.1	08/06/2017	Michael Jacoby (IOSB)	Updated structure Updated with content from D2.1 Added content to Section 5
v0.2	12/06/2017	Michael Jacoby (IOSB)	Added Section 5.1, 5.2
v0.3	16/07/2017	Michael Jacoby (IOSB) João Garcia (UW) Antonio Paradell (ATOS/WL) Luca de Santis (NAVIGO) Matteo Pardi (NXW) Karl Kreiner (AIT)	Updated Section 5 Updated Section 5.5.2 Updated Section 5.5.5 Updated Section 5.5.3 Updated Section 5.5.4 Updated Section 5.5.4
v0.4	21/07/2017	Szymon Mueller (PSNC) Michael Jacoby (IOSB)	Updated Section 5.7 Formatting of whole document
v1.0	26/06/2017	Svenja Schröder (UNIVIE) Marcin Płóciennik (PSNC) Michael Jacoby (IOSB)	Update of all sections Update of all sections Added Section 5.6.4
v1.1	12/07/2017	Ivana Podnar Žarko (UNIZG-FER) Michael Jacoby (IOSB)	Updated all sections Updated all sections
v1.2	14/07/2017	Sergios Soursos	Final editing and submission-ready version

Table of Contents

Table of Figures

Table of Tables

Table of Listings

(This page is left blank intentionally.)

1 Executive Summary

One of the main objectives of the symbIoTe project is creating a mediation framework to facilitate the discovery and sharing of connected devices across existing and future IoT (Internet of Things) platforms, as well as to enable platform federations and the rapid development of cross-platform IoT applications. The problem is that these devices are managed by different IoT platforms, which are designed for different application domains. The requirement to enable information technology to deal with the semantic of data has been one of the grand challenges in the computer science domain in the past, and probably will be one for the near future. Having this in mind, this document proposes a complete solution to the problem of semantic interoperability: first, we provide an analysis of the problem domain and then present a practical solution to the IoT interoperability problem. Deliverable D2.4 is the revised version of Deliverable D2.1 that focused on the analysis of the general problem and defined the basic approach to interoperability specified by the symbIoTe consortium. D2.4 provides more detailed information on how symbIoTe approaches semantic interoperability thereby going beyond what was stated in deliverable D2.1.

To understand the motivation of this document one may imagine a typical IoT platform for any given domain, for example, a climate control system for a smart home. Such a platform deals with data in a given context where the meaning is predefined, e.g., the scale of a thermostat ranging from 0 to 5, where 0 means *no heating* and 5 *valve is open*. It also has implicit data models like the location of a radiator, which relates to a room and maybe to a heating circuit. Another IoT platform could manage for example self-monitoring devices that collect data within a different context and different predefined meanings of temperature and locations. To make such different IoT-platforms understand each other, the meaning of data and concepts must be explicitly defined. The background chapter on semantic interoperability introduces possible technical solutions for this task. With the developments influenced by the so-called *Semantic Web*, there are established standards, methods and tools available, like the RDF and the OWL format to describe the semantics of data, which will be used by symbIoTe. The document also describes the achievements in the development of semantic mapping, to translate one information model into another that is semantically similar but structurally different. This capability is required for enabling interoperability between platforms, which operate on individual data models.

There are several ways to achieve semantic interoperability. Section 4 explains and discusses the possible approaches, from the simplest approach where everybody shares the same understanding, to the most complex one where everybody may use different concepts and interpretations that are then translated. From a practical perspective, neither the easy nor the complex approach is feasible for most real-life applications. Therefore, symbIoTe proposes an approach where it starts from a set of basic concepts shared across all platforms that are connected via the symbIoTe framework[1]. These basic concepts are sufficient to provide “meta”-understanding about the connected IoT platforms and their resources, so that symbIoTe can provide a generic interoperable mediation service for the IoT domain. To cover the actual meaning of platform-specific data, more detailed platform specific concepts are required. symbIoTe thus proposes an approach that allows multiple extensions to the basic concepts and aims to provide semantic and

[1] We name the chosen approach "Core Information Model with Extensions."

syntactic transformation as a common interoperability service for those extensions. These extensions can be platform-specific but also related to existing standard ontologies.

This deliverable further explains how the approach to semantic interoperability presented in this document is incorporated into the symbIoTe architecture presented in D1.4.

2 Introduction

In the context of this document, semantic can be understood as the meaning of things. Its main purpose in symbIoTe is to enable interoperability, especially semantic interoperability, which is “the ability of computer systems to exchange data with unambiguous, shared meaning” [1] (see Section 3.2 for a detailed definition of semantic interoperability). Some parts of this deliverable are based on the paper “Semantic Interoperability as Key to IoT Platform Federation” [2] as well as the paper “Semantic interoperability in IoT-based automation infrastructures” [3], both of which were authored as part of the academic dissemination within symbIoTe.

2.1 Semantics in symbIoTe

As the overall objective of symbIoTe is to create an interoperability framework for IoT platforms and to enable platform federation, achieving (semantic) interoperability is a key challenge. symbIoTe plans to use semantic technologies to bridge the semantic gap between existing and future IoT platforms with a goal to enable interoperability on a higher level, the semantic level, which is a step forward in comparison to state of the art solutions that primarily focus on syntactic interoperability.

2.2 Purpose of the document

The purpose of deliverable D2.4 “Revised Semantics for IoT and Cloud resources” is to document the outcomes and results regarding semantics in the symbIoTe project. The role of semantics within symbIoTe is described and approaches to achieve semantic interoperability are identified and discussed. A detailed description on how symbIoTe achieves semantic interoperability and the created information models are presented. This document also specifies the impact of semantics on the architecture and provides a realistic vision that goes beyond the project scope.

This document is the revised version of D2.1 “Semantics for IoT and Cloud Resources” and therefore can be seen as an updated version of it. Most changes are related to Section 5, especially with respect to the information models.

2.3 Document Scope

This document reports the work accomplished in T2.1 “Semantics for IoT and Cloud Resources” with a strong focus on semantic interoperability. It presents a theoretical analysis of the problem domain of semantic interoperability together with multiple possible approaches to address this challenge. Furthermore, it discusses to what degree these approaches are suitable to be used within symbIoTe, and justifies a decision to use the Core Information Model with Extensions. Thereby, it strongly focuses on the definition of information models.

Although it is briefly mentioned in the document, syntactic interoperability and how it is addressed and realized by symbIoTe is not the primary subject to this deliverable.

2.4 Relation to other deliverables

The content of this deliverable was motivated and influenced by the outcome of tasks T1.3 “System requirements” and T1.4 “System architecture” which has been published in D1.2 “Initial Report on System Requirements and Architecture” resp. D1.4 “Final Report on System Requirements and Architecture”.

The work done in T2.1 “Semantics for IoT and Cloud Resources” and its outcome that is documented in this deliverable has impact on multiple tasks, as semantic interoperability is a core functionality of symbIoTe. The need for new functionality and components realizing semantic interoperability directly influenced T1.4 “System architecture”, T2.2 “Virtual IoT environment” and T3.3 “Specification & Implementation of IoT Federation”. Furthermore, T2.3 “Implementation of symbIoTe domain-specific enablers”, T3.2 “Security and Access Scopes” and T4.1 “Local Registration, Discovery and Interoperability of Smart Objects” are influenced indirectly by the outcome of this deliverable.

2.5 Deliverable Outline

This deliverable is structured as follows: Section 3 introduces the term semantic interoperability and provides background information on current semantic web technologies and standards. In Section 4, the problem of achieving semantic interoperability between multiple IoT platforms is explained in detail, together with a number of possible approaches addressing it. Section 5 gives an outline of the approach chosen by the symbIoTe consortium and justifies the decision. The information models created and used within symbIoTe are presented in detail. Further, we explain how the approach to semantic interoperability affects the system architecture and provide a vision going beyond the project scope. The document closes with Section 6 presenting conclusions and next steps.

3 Background

3.1 What are semantics and why do we need it?

According to the Merriam-Webster dictionary, *semantics* is the study of meaning, especially “the meaning or relationship of meanings of a sign or set of signs” [4]. A sign here is a fundamental linguistic unit that designates an object or relation and therefore semantics can be understood as the mapping of signs to their meaning. For example, a sign could be a word, e.g. the word ‘table’, or a URI (Uniform Resource Identifier) like ‘http://www.example.com/table’ and the corresponding meaning would be the concept of a table that is defined by its properties and context specifying that it is a constructed thing which has some legs, a solid top and is normally found in a man-made environment. In every act of communication, the involved actors, human or computers, require a shared understanding of things. A typical way to express semantics is via a taxonomy or an ontology, which is “an explicit specification of a [shared] conceptualization” [5] (explicit because all relevant elements must be explicitly named in order to avoid misinterpretations; shared because there must be a common agreement within a specific domain of interest).

Knowing this, it is obvious that semantics plays an important role in every act of communication. Without it, we would not be able to understand the meaning of the exchanged information, or at least we could not be sure that we are having the same understanding of it as our communication partner. To give an example image two people having a conversation about the weather and one says “…it had 40 degrees outside”. In this case, the semantics are not clearly defined as degrees could refer to degrees Fahrenheit or to degrees Celsius. Communication might work by chance if both conversation partners e.g. come from the same country and have the same understanding of degree in the context of temperature but as long as they do not use a shared or agreed upon vocabulary they cannot be sure that they both have the same understanding of the exchanged information.

Transferring this example to the technical level of IoT platforms communication where data is exchanged, processed and interpreted by machines this clear and formal definition of meaning is even more important. This is because machines (most of the time) do not have any capability of reasoning to come up with a probably correct interpretation of the meaning of received data from others (mainly because they don’t have the information they need for this task, in the given example this would be the cultural background of the person speaking). Therefore, semantics is essential to bring multiple IoT platforms (which were created by different persons with different cultural backgrounds and) with focus on different domains together as they most probably will have different understandings of things and interoperability most likely will fail.

3.2 Semantic Interoperability

A common understanding of the concept of interoperability is described in the Levels of Conceptual Interoperability Model (LCIM) [6]. This definition is derived from simulation theory, but it has a much broader applicability. This definition distinguishes seven levels of interoperability that are grouped in three parts [7]:

- Integratability contends with the physical/technical realms of connections between systems, which include hardware and firmware, protocols, etc.
- Interoperability contends with the software and implementation details of interoperations, including exchange of data elements based on a common data interpretation, etc.
- Composability contends with the alignment of issues on the modelling level. The underlying models are purposeful abstractions of reality used for the conceptualization being implemented by the resulting simulation systems.

For computer systems, the ability to have a clear and formalized way to express the meaning of things is an indispensable precondition to achieve semantic interoperability. Bringing both terms together, semantic interoperability can be defined as “the ability of computer systems to exchange data with unambiguous, shared meaning” [1].

3.3 Semantic Web

The term Semantic Web was first used by Tim Berners-Lee in his article “The semantic web” from 2001 stating that “the Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation” [8].

The main concept of the Semantic Web is to extend the existing World Wide Web from a web of (interlinked) documents to a web of (interlinked) machine-readable and processable data. This is achieved through a family of very specific technology standards driven by the World Wide Web Consortium[2] (W3C). As the concept and the ideas of the Semantic Web are essential to understand the problem of and the proposed solutions to semantic interoperability presented in this document, the basic technologies and standards are explained hereafter.

3.3.1 Resource Description Format (RDF)

RDF[3] is the (metadata) data model for the Semantic Web and therefore can be seen as its cornerstone technology. All data on the Semantic Web is represented in RDF, even the schema description. The main advantage of RDF is its innate flexibility compared to the tabular data model of relational databases and the tree-based data model of XML.

As shown in Figure 1, data in RDF is often depicted as a labelled, directed graph where the nodes represent *resources* (depicted as ovals) or *literals* (depicted as rectangles) and the labelled edges represent *relations* (often also called *predicates*). This representation clearly shows the power of RDF to represent data without previously defining its structure, unlike with relation databases.

[2] https://www.w3.org/
[3] https://www.w3.org/TR/2004/REC-rdf-primer-20040210/

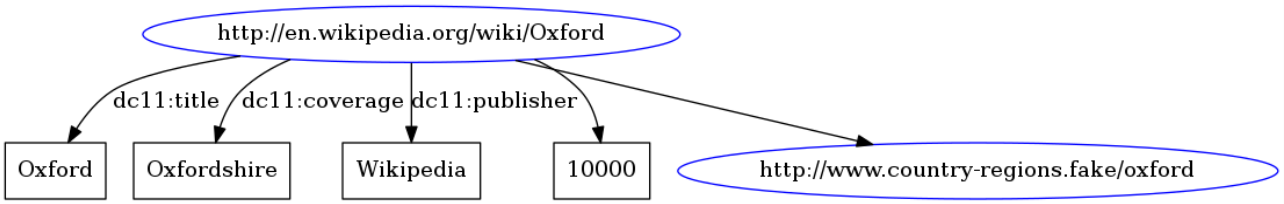


Figure 1 Example RDF data depicted as graph.

All data in RDF is described as a *triple* (also often called *statement* or *3-tuple*) of the form *(subject, predicate, object), e.g. (<http://en.wikipedia.org/wiki/Oxford> dc11:title "Oxford"^^xsd:string)*. Subjects and predicates are resources that are represented by an URI (Universal Resource Identifier). Objects can also be a resource or a literal (which basically is another name for value). Datatypes of literals can be defined using XSD datatypes[4]. The use of URIs to identify resources (which can be seen as an atomic piece of information) allows globally unique addressing even between different databases and thus allows global interlinking of information.

Collections of triples are called a graph. For better data management (e.g. access control, simplified updating, trust), large collections of RDF data are usually segmented into different named graphs. Triples stored in a named graph are often referred to as quads as they are of the form *(graph, subject, predicate, object)*. Databases designed to store RDF data are referred to as triple (or quad) stores.

RDF is an abstract (metadata) data model, which means there are multiple serialization formats that can be used to represent RDF data. The most popular are RDF/XML[5], N-Triples[6], Turtle[7], TriG[8], RDFa[9], Notation3 (N3)[10] and JSON-LD[11].

3.3.2 RDF Schema (RDFS) & Web Ontology Language (OWL)

RDFS[12] and OWL[13] are RDF schema languages, which are used to define meta models for RDF data. These meta models are often referred to as vocabularies or ontologies, which are explained in detail in the next section. Both, RDFS and OWL are themselves expressed using RDF.

3.3.2.1 Vocabularies and Ontologies

The terms vocabulary and ontology are terms used very frequently in the context of Semantic Web but are often defined and thus used differently. Generally, they are used to describe a set of triples with a strong logical cohesion, i.e. belonging to a certain domain. The W3C states, “there is no clear division between what is referred to as vocabularies

[4] https://www.w3.org/TR/swbp-xsch-datatypes/
[5] https://www.w3.org/TR/rdf-syntax-grammar/
[6] https://www.w3.org/TR/n-triples/
[7] https://www.w3.org/TR/turtle/
[8] https://www.w3.org/TR/trig/
[9] https://www.w3.org/TR/rdfa-primer/
[10] https://www.w3.org/TeamSubmission/n3/
[11] https://www.w3.org/TR/json-ld/
[12] https://www.w3.org/TR/rdf-schema/
[13] https://www.w3.org/TR/owl2-overview/

and ontologies. The trend is to use the word ontology for more complex, and possibly quite formal collection of terms, whereas vocabulary is used when such strict formalism is not necessarily used or only in a very loose sense" [9].

For the rest of the document we will use the term ontology and refer to it as a set of triples defining a meta model. This means ontologies only contain information about general concepts and no data of concrete instances (often called individuals). The main idea behind developing ontologies is to structure data in a clear and machine-readable way to have a common understanding of things as well as to enable inference (making implicit knowledge explicit through reasoning).

3.3.2.2 RDFS

RDFS is the most basic schema language of the Semantic Web. It is a very minimalistic set of classes and properties used to describe classes of and relations between objects. RDFS also distinguishes between classes and individuals (instances of classes). The most important classes are listed in Table 1.

Table 1 Most important classes of RDFS.

Class Name	Description
rdfs:Resource	all things declared by RDF are resources
rdfs:Class	describes the concepts of a class
rdfs:Literal	describes the concept of a literal
rdfs:Property	the class for properties

Table 2 Most important properties defined in RDFS.

Property Name	Description
rdfs:domain	defines to which subjects does a property applies
rdfs:range	defines the set of values a property can accept
rdf:type	used to state that a resource is an instance of a class
rdfs:subClassOf	defines one class as a subclass of another class
rdfs:label	provides a human-readable version of resource's name
rdfs:comment	provides a human-readable description of resource
rdfs:seeAlso	link to another resource that might provide additional information

3.3.2.3 OWL

OWL is another RDF schema language, which is more expressive than RDFS and can express quite subtle ideas about data. It is very efficient as it comes in various flavours, called profiles, each with a different level of expressivity and therefore complexity and computational power needed for inference. It includes everything RDFS provides and adds many new classes and properties like

- owl:TransitiveProperty
- owl:unionOf
- owl:sameAs

- owl:inverseOf
- owl:hasValue

moreover, some properties to model meta-meta-data like

- owl:import
- owl:versionInfo
- owl:deprecatedProperty

3.3.3 SPARQL Protocol and RDF Query Language

SPARQL[14] is the de facto standard query language for RDF data and quite similar to the query language for relational data SQL.

```
1  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3  PREFIX dbp: <http://dbpedia.org/ontology/>
4
5  SELECT ?city ?popTotal
6  FROM <http://example.com/dataset.rdf>
7  WHERE
8  {
9      ?city rdf:type <http://dbpedia.org/class/yago/CitiesInTexas> .
10     ?city dbp:populationTotal ?popTotal .
11 }
12 ORDER BY ?popTotal
```

Listing 1 An example SPARQL query.

Looking at the overall structure of the example SPARQL query in Listing 1, we see that it is quite similar to SQL. One main difference is the format of the WHERE clause as with SPARQL it consists of a list of so-called triple patterns. These triple patterns are normal triples which can contain a variable (starting with a '?') on every position. When executed, the variables in the triple patterns are bound to concrete values whereat all occurrences of the same variable are bound to the same value. This concept is called graph pattern matching and the results of the query are all possible valid combinations of values bound to all mentioned variables. In SPARQL, there exist multiple query types as specified in Table 3.

Table 3 SPARQL query types.

Query Type	Description
SELECT	returns a list of bindings which is basically a table like in SQL
CONSTRUCT	returns a RDF graph which is basically a list of triples
DESCRIBE	returns information about a single resource. What will be returned is not generally defined but rather implementation dependent
ASK	returns true if the query has at least one result, otherwise false

[14] https://www.w3.org/TR/sparql11-overview/

3.4 Information Models of Existing Platforms used in symbIoTe

In this section, we highlight the problem of enabling interoperability between multiple platforms within symbIoTe by providing a practical example. Therefore, we present the information models of two existing platforms, openUwedat from AIT and Symphony from NXW that will be used within symbIoTe. These real world examples demonstrate which services are needed when different IoT platforms want to exchange information.

3.4.1 openUwedat

openUwedat is not a closed system but rather a library and framework for arbitrary time series oriented applications. It does not use a single data model, but rather adapts the data model to the needs of specific applications. Nevertheless, there is a core data model that applies to all applications and there are extensions that apply to individual applications. Thus, the description of the data model is split into two parts, the general information model and the information model used for the symbIoTe installation.

3.4.1.1 General Information Model

Like depicted in Figure 2, the core data model of openUwedat is constructed around *datapoints*. These datapoints are sources and destinations of time series data.

Each datapoint can be queried to emit *TimeSeries* data. A TimeSeries is mainly a container for *Slots*. TimeSeries object are comparable to the *Observations* collection of the OData interface, which is currently used for syntactic interoperability in symbIoTe.

Each Slot has a reference time and zero or more *values* assigned to it. Thus, it is closely related to symbIoTe's concept of an *Observation*, as defined in Section 5.2.

Each slot's value can be any type (restricted to Java types at the moment). This is related to symbIoTe's idea of an observation value.

TimeSeries have *Properties*. They are addressed in a dictionary style by using key-strings. For this reason, the property system is easily extendable with new properties needed for particular applications.

There is a set of Properties within the core model whose existence is mandatory or at least strongly recommended:

- ValueKeys: This key describes which values are available within a slot.
- Units: A description of the Units of Measurements (UoM) related to each value.

Datapoints also have properties. The only mandatory property is *ObservedProperty*.

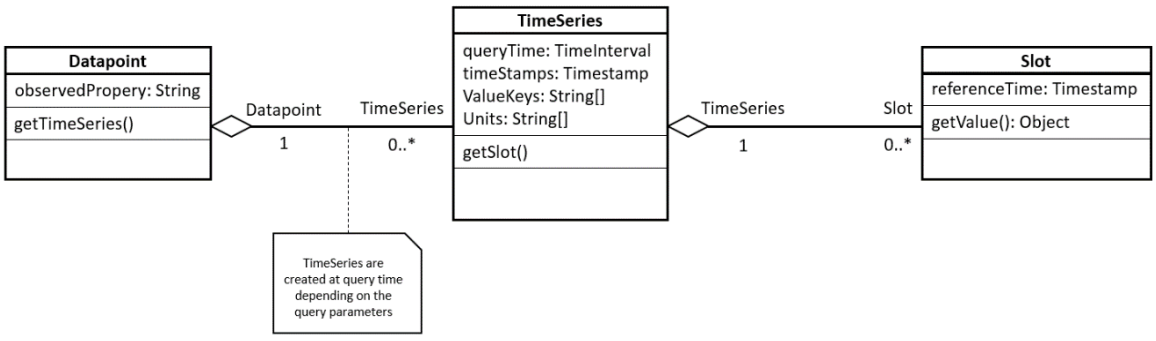


Figure 2 The core Information Model of openUwedat.

3.4.1.2 Information Model of the SymbIoTe installation

For symbIoTe most of the core model was already covered by openUwedat's core model.

Two important extensions were needed nevertheless:

- We needed a simple ID that can be exposed via the OData interface. This was added to the set of properties.
- Each datapoint (aka Sensor or Resource) has a concept of location. For the symbIoTe use case, this concept is simple as we are dealing with fixed stations. Therefore, we just added a property *location*, which is composed of longitude, latitude and altitude.

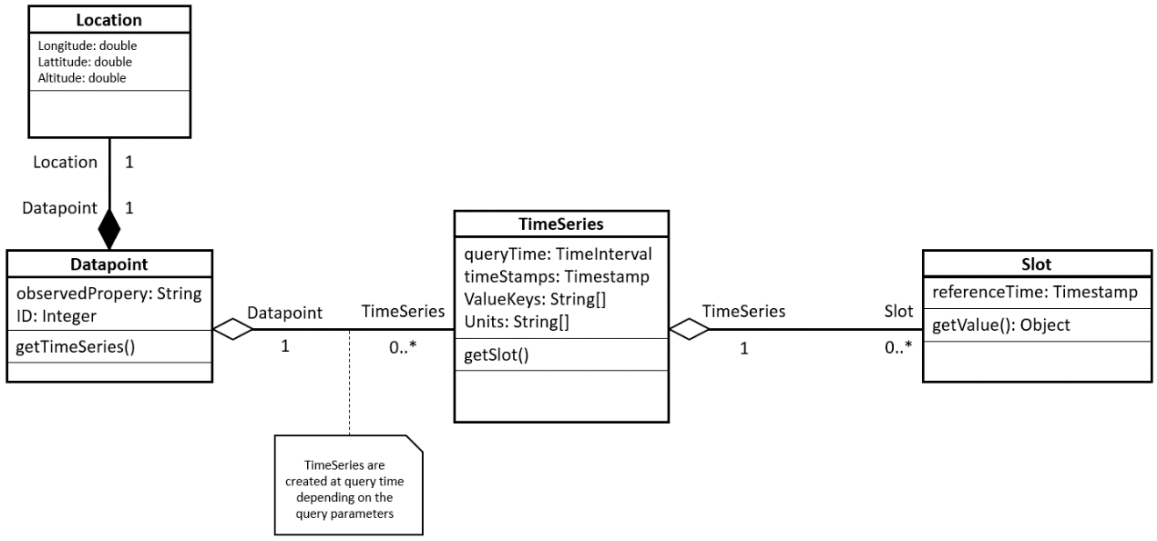


Figure 3 openUwedat Information Model for SymbIoTe.

3.4.2 Symphony

Symphony is the NXW platform for the integration of home/building control functions, devices and heterogeneous subsystems. Symphony is a service-oriented middleware integrating several functional subsystems into a unified IP–based platform. As hardware/software compound, Symphony encompasses media archival and distribution, voice/video communications, home/building automation and management, and energy management. The platform owns a generalized abstract model for all the Internet Connects Objects (e.g., smartphones, printers, sensors, actuators, etc.), managing a set of context-driven decisions/actions. This leads to a quite complex data model, partitioned into subcomponents, one for each service provided by the platform.

Figure 4 depicts a high-level diagram that explains the Symphony platform data model.

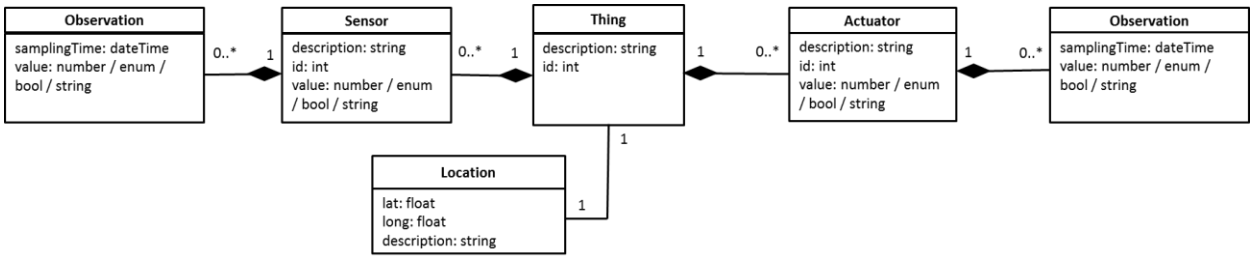


Figure 4 The Symphony data model.

Since Symphony integrates a large number of sensors and actuators, two examples are shown below:

- a temperature sensor (depicted in Figure 5)

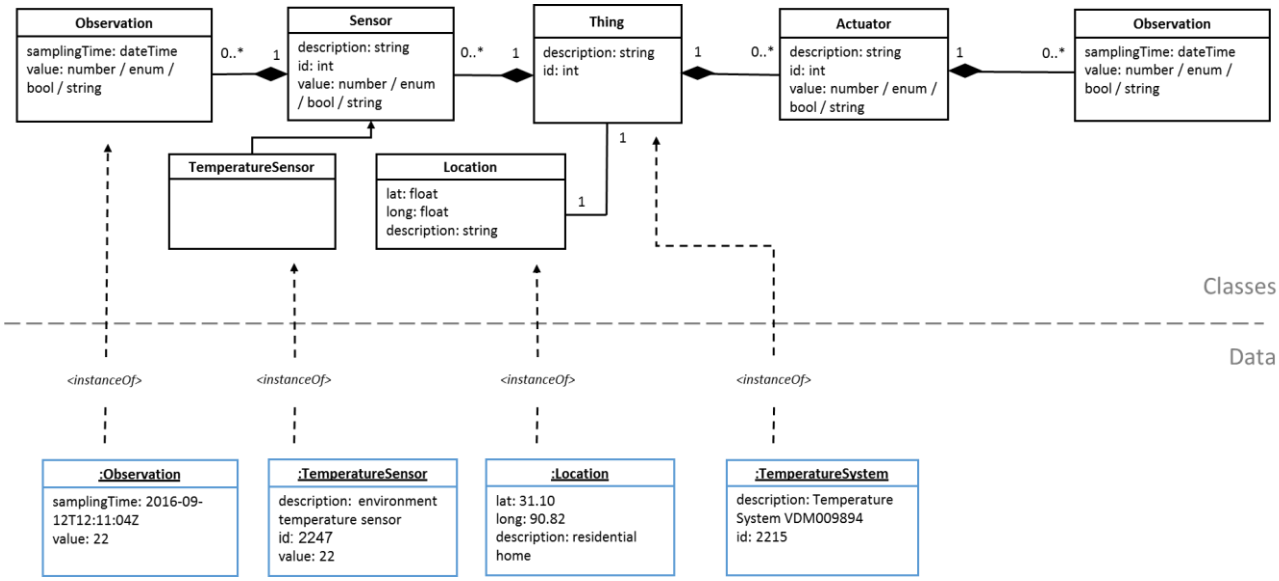


Figure 5 Data model for a temperature sensor in Symphony.

- a linear load actuator (depicted in Figure 6)

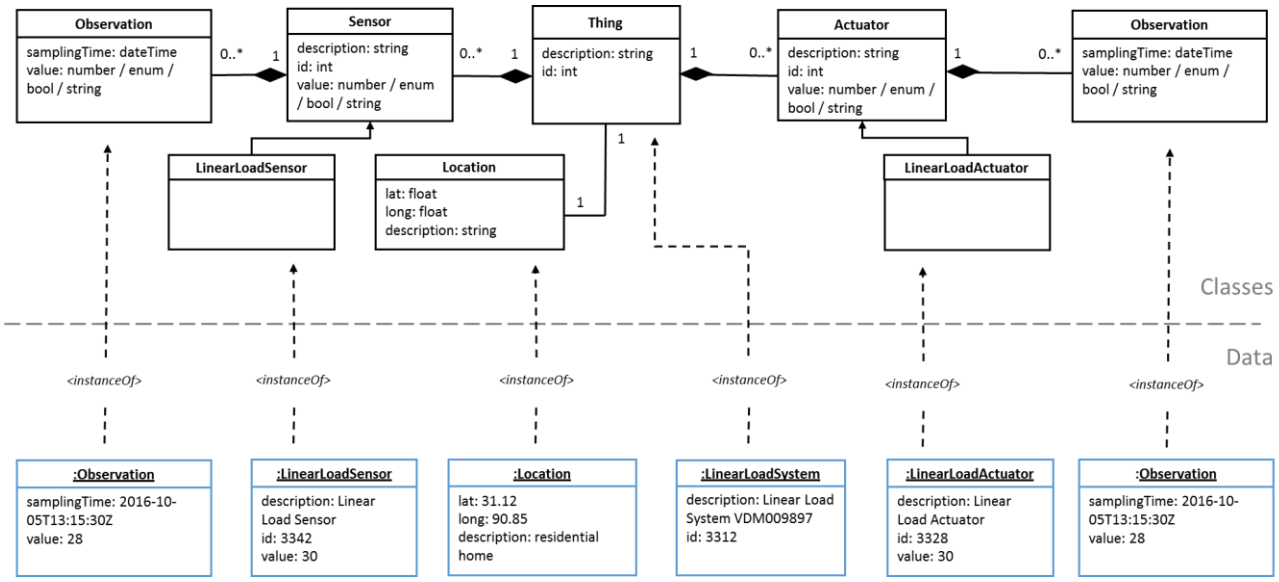


Figure 6 Data model for a linear load actuator in Symphony.

4 Achieving Semantic Interoperability

In this section, we introduce the problem of semantic interoperability between multiple IoT platforms. Furthermore, we present multiple possible approaches how semantic interoperability can be achieved on a general level together with their advantages and disadvantages, which is the outcome of the analysis on semantic interoperability within the symbIoTe project.

4.1 Problem

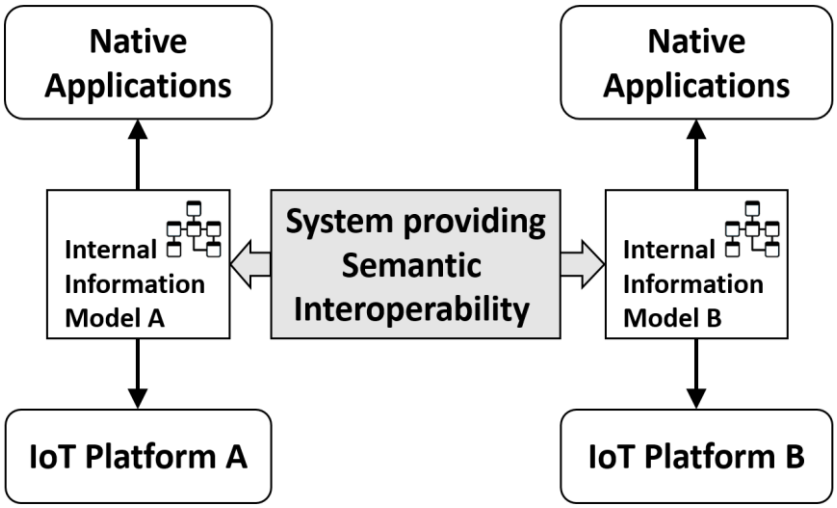


Figure 7 Schematic representation of the problem of semantic interoperability between different IoT platforms.

Figure 7 depicts the general problem of semantic interoperability between multiple IoT Systems. Each IoT platform models its data using an internal information model that is used by native applications of this specific platform. As different IoT platforms are designed by different bodies/communities/companies and often focus on different aspects within IoT they tend to have platform-specific information models that differ in multiple aspects. Closing this semantic gap means enabling semantic interoperability.

4.2 Possible Approaches

As outcome of our research on the problem domain of how to achieve semantic interoperability, we identified a possible solution space, which is depicted in Figure 8. It can be thought of as a line between two opposed approaches, which are, on the one side, using a single core information model that all platforms must use and, on the other side, using completely independent platform-specific information models for each platform, which then need to be aligned using semantic mapping techniques. In between, there exists a large, not clearly defined number of intermediate solutions. In the following, we present the two basic solutions together with three representative intermediate solutions. These approaches are motivated by and in-line with the concepts presented by Wache et al. [10] and Choi et al. [11].

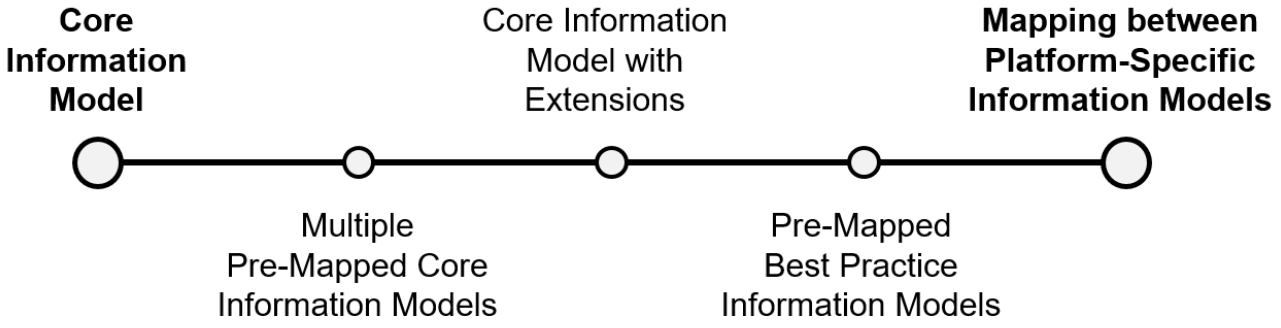


Figure 8 Solution space for possible approaches to semantic interoperability.

4.2.1 Core Information Model

The most widespread approach amongst existing platforms is to use a single core information model that all platforms must comply with. This means that a platform can only expose data that fits into this core information model, as custom extensions are not permitted. If a platform needs to expose data that does not fit into the core information model the platform cannot expose this data and cannot inter-operate with others. From our perspective this is rather some form of standardization than an approach to enable true semantic interoperability that will also work without adaption (e.g. changing the "standard"/Core Information Model) when new IoT platforms covering eventually new domains will emerge.

Pros

- easy to implement and use since the data from all platforms follows the same information model
- resulting system easy to use for app developers who only need to know one information model

Cons

- finding/defining an information model all platforms can agree upon may be difficult
- information model tends to become complex as it must comprise all data that should be exchangeable between platforms
- will always exclude some platforms whose internal information model does not fit the core information model
- no way to integrate future platforms with information models not compatible to the core information model without breaking the existing system

4.2.2 Multiple Pre-Mapped Core Information Models

Based on the single core information model approach this one tries to make it more easy and convenient for platform owners to integrate their internal information model by supporting not only a single core information model but also multiple ones. To achieve that a large number of existing platforms can easily participate it would be a good idea to choose well-established information models (e.g. the Semantic Sensor Network Ontology [12] (SSN) or the oneM2M ontology [13]) as core information models. To ensure interoperability between platforms using different core information models the supported models are already mapped to each other. As it will not always be possible to map two core information models completely there will be some degree of information loss if

platforms conform to different core information models. On the other hand, if they comply with the same model, they will be fully interoperable.

Pros

- flexible approach as further core information models and mappings can be added over time
- does not enforce use of one single core information model which excludes less platforms from participating

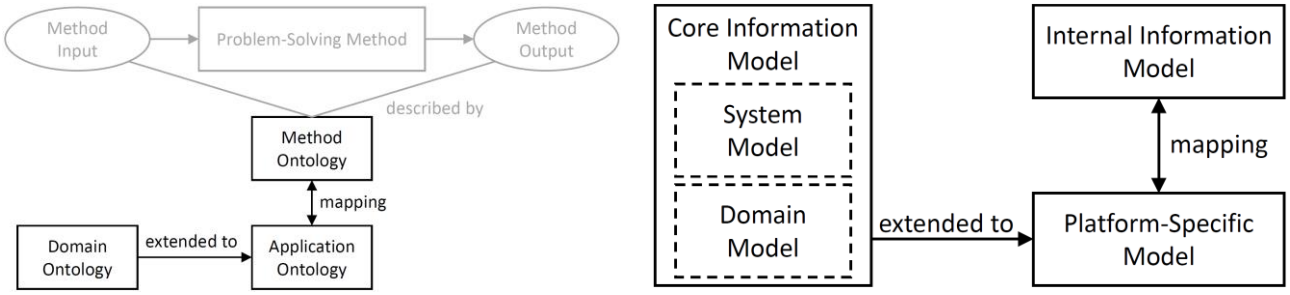
Cons

- may still exclude some platforms whose information model does not match any of the core information models

4.2.3 Core Information Model with Extensions

This approach is based on an information model that is designed to be as abstract as possible but at the same time as detailed as needed. Therefore, the core information model should try to only define high-level classes and their interrelations, which act as extension points for platform-specific instantiations of this information model. These platform-specific instantiations either use the provided classes directly or they can define a subclass, which can hold any platform-specific extensions to the core information model, e.g. additional properties. Besides the high-level classes, the core information model may also contain properties the system needs that will be very general properties like *ID* or *name* in most of the cases.

This approach resembles an approach for a model-driven knowledge engineering system (KES) presented by Studer et al. shown in Figure 9a where a domain ontology is extended to an application ontology which is mapped to a method ontology that is finally used to define in- and output of a method used to solve a problem. The core information model with extensions can be very closely matched to this approach as depicted in Figure 9b. The main difference is, that there exists not only a domain ontology that is extended but rather the core information model, which contains the domain model and the system model (which can be seen as a platform-specific extension of the domain model to the system that provides the interoperability). The application ontology corresponds to the platform-specific model, which is a platform-specific extension to the core information model, and the method ontology corresponds to the internal information model of the platform as depicted in Figure 9.



(a) Ontology-based approach for a model-driven KES (based on Figure 6 from [43]).

(b) Conceptual structure of the Core Information Model with Extensions approach.

Figure 9 Structural similarity between an ontology-based model-driven KES and the Core Information Model with Extensions.

This results in an information model that has a minimalistic core that all platforms must conform to and extension points to realize custom requirements. Two platforms using different extensions can directly understand each other in terms of the core information model and when they need also to understand the custom extensions, they must define a semantic mapping between their extensions.

Pros

- provides basic interoperability between platforms by defining minimalistic core information model
- provides full flexibility by custom extensions, i.e. no platforms are excluded
- high acceptance from adopter-side as it combines basic out-of-the-box interoperability (by the core information model) with support for complex scenarios (through extensions and semantic mapping)

Cons

- requires semantic mapping when custom extensions need to be understood by different platforms
- defining a semantic mapping can be a complex task and requires additional work from developers/platform owners
- design of the core information model is a complex task

4.2.4 Pre-Mapped Best Practice Information Models

Essentially, this is the same approach as *Multiple Core Information Models* but with one small but significant modification: the provided information models are no longer seen as core information models but rather as *best practice* information models. Hence, platforms must not be compliant to any of the provided information models as in the previous approach but can choose their information model freely. If they choose to re-use one of the provided best practice information models they will gain instant interoperability to other platforms also aligned with one of the best practice information models.

Pros

- no limitations on information model, hence does not exclude any platform
- best practice information models make usage for inexperienced platform owner more easy

- better and broader interoperability due to already aligned best practice information models

Cons

- no initial interoperability between platforms as long as no mapping is defined when no pre-mapped information model is used
- defining a semantic mapping can be a complex task and requires additional work from developers/platform owners

4.2.5 Mapping between Platform-Specific Information Models

In this approach, there is not anything like one or more core information models. Instead, every platform independently provides its own information model. Interoperability is only achieved through mapping between these platform-specific information models.

Pros

- not limited only to a fixed set of information models but rather supports all possible information models
- mappings can be added iteratively increasing the degree of interoperability

Cons

- no initial interoperability between platforms as long as no mapping is defined
- defining a semantic mapping can be a complex task and requires additional work from developers/platform owners
- the system does not understand any of the data it is processing

4.3 Comparison of Approaches

Section 4 presents the analysis of possible approaches for the semantic interoperability. It is required to decide how independent IoT platforms should exchange meta-information and thus create a pool of IoT data sources, resources and services available to the applications. Such an interoperability is a crucial functionality for symbIoTe because it addresses the need for a common framework across existing and future IoT platforms. The framework will enable discovery and sharing of resources for rapid cross-platform application development. Those applications exploiting multiple data sources and resources will bring new innovative functionalities and lift up IoT to the next technology level.

The first described approach “Core Information Model” seems to be the most suitable to enforce interoperability between IoT platforms. Theoretically, if a shared vocabulary exists there is no need to worry about inconsistency, complexity and performances issues of additional operations like translations (complete or incomplete). Moreover, the development process is simplified because developers can stick to only one standard solution. From a business point of view application development becomes quicker and cheaper. A popular understanding in the IoT domain is, that the quality und usefulness of offered IoT services increases with the number of IoT devices and platforms deployed. However, in the majority of the use cases heterogeneity has to be considered when introducing a service or a product to the market. Nowadays, there are many IoT platforms utilizing different information models to describe their resources, applying policies to share their data and comprising implementation limits preventing smooth integration with other platforms. It would be extremely difficult to convince IoT platform vendors to make deep (and thus expensive) changes in their product. More realistic is to propose the solutions,

which try to find some compromise and balance. Nonetheless, the pursuit of interoperability with the use of all possible platform-specific information models (“Mapping between Platform-Specific Information Models”) is not a straightforward direction either. In this case, semantic mapping and complexity for developers would bring disadvantages like translation performance issues, possible slow progress of application development, incomplete translations and the costs of supporting of new emerging information models.

If two aforementioned approaches are not suitable then one can analyse the three others described in the previous subsections. The first one, “Core Information Model with Extensions”, specifies the very abstract representation of an information model, which may be applicable to any IoT platform. This allows for exchanging at least a set of basic, platform-independent information. Any specific information may be modelled as an extension including required mapping (translation) between extensions of respective platforms. Apart from obvious advantages like flexibility and partial standardization, one should emphasize the downsides. It is not clear if the generic abstract representation is enough useful for interoperability and effective in real use cases. Moreover, certain mappings between extensions may still be complex and incomplete. Regarding the last two approaches, “Multiple Pre-Mapped Core Information Models” and “Pre-Mapped Best Practice Information Models”, they are based on mappings of a subset of information models. They require the implementation of translation mechanisms which results in all related difficulties but may be suitable if it is assumed that the number of information models is limited and stable. Moreover, those models are well known and widely accepted.

While Section 4 discussed all potential approach to semantic interoperability in detail, the next chapter will present the chosen approach within symbIoTe.

5 symbIoTe's Approach to Semantic Interoperability

We analysed the approaches to semantic interoperability presented in Section 4 regarding their suitability for symbIoTe and decided to follow the Core Information Model with Extensions approach mainly due to two reasons. First, symbIoTe needs to have at least some degree of understanding about the resource descriptions exposed by the platforms to be able to provide additional services, e.g., location-based search for sensors. For this, we need platforms to use the same terms to describe all information that is relevant to symbIoTe. This is achieved by having a common, minimalistic Core Information Model (CIM) covering these symbIoTe-relevant terms. Second, this approach gives almost full flexibility to platforms as it allows them to model all non-symbIoTe-relevant information within extensions with the only restriction that they must be extensions of the CIM. Due to the first reason, the approaches Pre-Mapped Best Practice Information Models and Mapping between Platform-Specific Information Models are not suitable as with these approaches, symbIoTe would not be able to understand any of the resource descriptions exposed by the platforms. On the other side, the approaches Core Information Model and Multiple Pre-Mapped Core Information Models are not suitable as they require a complete model of the whole IoT domain to be agreed upon between all platforms. Even if all platforms could agree on such a model, it would massively limit the degree of freedom of the platforms to expose their data as they want or need to. This approach enables symbIoTe to support two patterns for semantic interoperability.

We name the first pattern *interoperability by standardization,* which means that as long as platforms use (partially) the same vocabulary to describe their resources, they are interoperable out-of-the-box. This is the de-facto standard level of semantic interoperability as it can be found in real-world systems. In symbIoTe, we enforce these vocabularies (also called models in our case) to be aligned with our Core Information Model. By this, we enable also out-of-the-box interoperability (in terms of the CIM) between platforms that use different vocabularies. Furthermore, we provide a Best-Practice Information Model (BIM), which is a special kind of Platform-Specific Information Model tailored to the domains of the use cases.

We name the second pattern *interoperability by mapping*. This allows two platform that use completely disjoint (besides the CIM which they both have to extend) vocabularies to interoperate by defining a mapping between their models. symbIoTe uses this mapping information to transparently hide the fact that the other platform uses a completely different vocabulary. From platform owner's side it seems as if the other platform is using the same model.

Supporting the *interoperability by mapping* pattern is a very complex task and goes far beyond the current state-of-the-art[15]. Therefore, this document focuses on the *interoperability by standardization* pattern and explains the idea behind the *interoperability by mapping* pattern but does not provide further technical details on its implementation. Furthermore, at the end of this section, we provide a vision on how an integrated interoperability solution including mapping could be designed to make it useable in practice.

[15] Note that interoperability by mapping is not foreseen in symbIoTe Description of Action.

Figure 10 shows the concept for realizing the Core Information Model with Extensions approach presented in Section 4.2.3. On the left and right hand side, we see two typical vertical IoT silos. Each platform uses its own internal information model to provide a platform-specific API, which is then used by native applications. Between those two vertical IoT silos, we see the symbIoTe interoperability framework depicted in light grey providing interoperability between the two IoT platforms. As proposed in Section 4.2.3 and shown in Figure 9b, symbIoTe uses two central information models: the Core Information Model (CIM) describing domain specific information (matches the Domain Model in Figure 9b) and the Meta Information Model (MIM) describing symbIoTe internal meta information about platforms and resources (matching the System Model in Figure 9b). For a platform to become symbIoTe-compliant, it must expose its data using a Platform-Specific Information Model (PIM), which is the CIM with platform-specific extensions to it. We also depict the key idea of the *interoperability by mapping* pattern depicted by the arrow connecting two platform-specific information models. This allows to define how the platform-specific extension of one platform can be translated into the platform-specific extensions of another platform, and therefore allows to define an arbitrary degree of interoperability between two platforms (i.e. the detail of alignment between two PIMs). When an app or a platform queries symbIoTe to find resources of interest on all available platforms, symbIoTe uses these mappings between platform-specific extensions to re-write the original query to fit the platform-specific information model of each extension and to execute the query against the stored metadata (resource metadata is of course in line with each PIM). Details on the symbIoTe Information Model as well as semantic mapping enabled by SPARQL query re-writing are provided in the following sections.

To enable syntactic interoperability, which is a prerequisite for semantic interoperability, we need platforms to expose their data in a unified way. For that, symbIoTe defines the Interworking Interface, a REST-based interface with JSON payload that is motivated by the Open Data Protocol[16] (OData). The data returned by each platform through the Interworking Interface is based on the platform's PIM. This means, upon a request to a platform the response may contain more information than defined in the CIM. If the requester is only interested in data based on the CIM, any additional fields in the JSON object can be ignored. However, if the requester is another platform using a different PIM and given that there exists a mapping in the Core between those two PIMs, a *data transformation* is required which transforms the original JSON object to another object in line with requester's PIM.

Data transformation cannot be performed by symbIoTe Core Services because this would require the Core to have access to all the data transferred between platforms, which breaches security. The queried platform would be capable of doing this without violating the security concept; however, platform owners might not be willing to provide, maintain and pay the computational resources needed to execute this task just to provide their data in any desired format to anyone consuming data from their platform. Therefore, this task might be performed on the requester-side. For this reason, symbIoTe will provide this functionality as a library that can be used by any consumer.

All information models created in symbIoTe are realized as OWL ontologies and are available on GitHub[17]. They are also hosted on the symbIoTe website under the following

[16] http://www.odata.org/
[17] https://github.com/symbiote-h2020/Ontologies

URLs so that all their URIs are resolvable via HTTP according to the principles to Linked Data [14]:

- Core Information Model https://www.symbiote-h2020.eu//ontology/core
- Meta Information Model https://www.symbiote-h2020.eu//ontology/meta
- Best-Practice Information Model https://www.symbiote-h2020.eu//ontology/bim

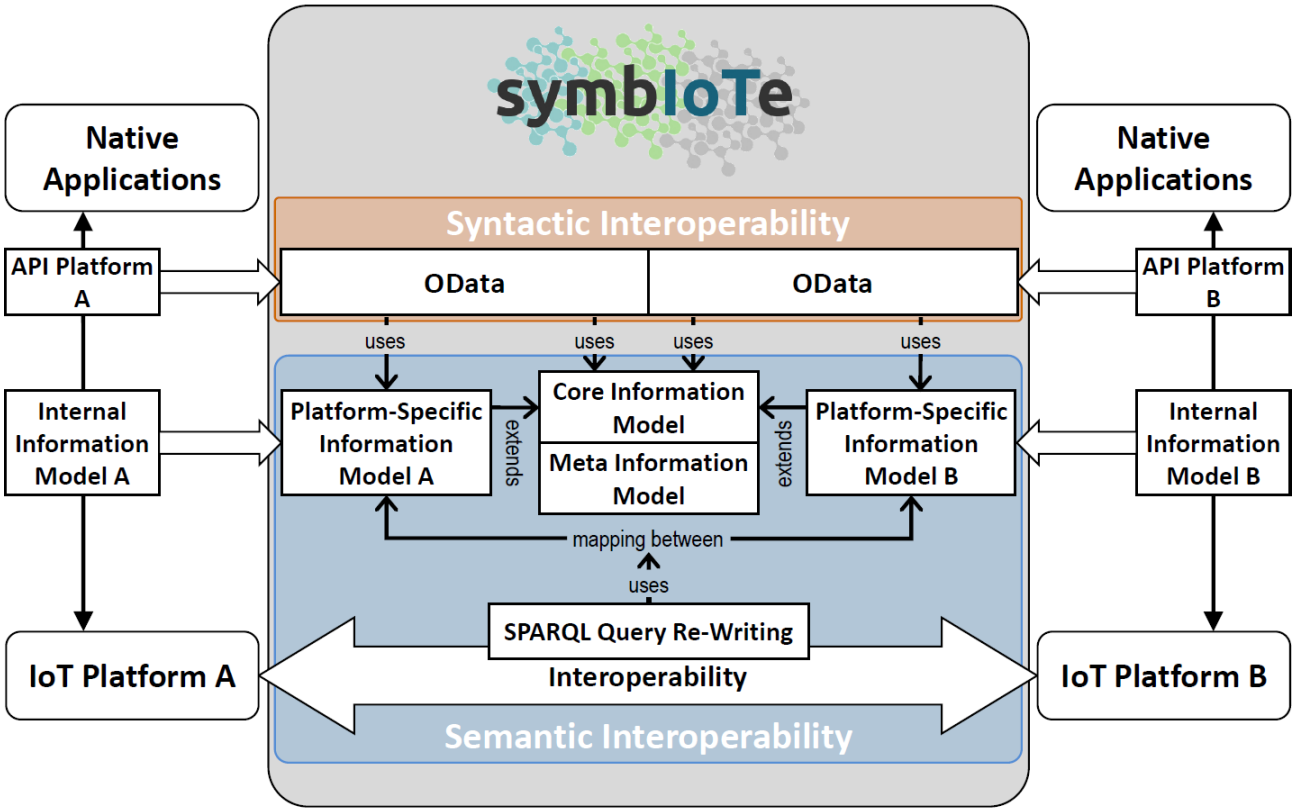


Figure 10 High-level diagram showing how symbIoTe approaches syntactic and semantic interoperability.

5.1 Survey on Information Model Domains

To find out which domains need to be covered by the symbIoTe information models, we conducted a project-internal survey. We started out by collecting requirements about relevant domains and came up with the following twelve domains that are divided into four groups:

- Observation & Measurement
  - Time: Concepts of time instant, time interval, time zones, etc.
  - Location: Basic geo-location like long/lat/alt, polygon, symbolic location.
  - Sensors: Definition of an abstract concept of a sensor.
  - Units of Measurement: Abstract concept, may include some model of units.
- Access & Control
  - Actuators: Definition of an abstract concept of an actuator.

- Services: Services offered by symbIoTe-compliant IoT platforms.
- Protocols: Protocols via which these platforms can be accessed.

- Interoperability
  - symbIoTe infrastructure: Description of symbIoTe-compliant platforms for discovery purpose.
  - Quality of Service / SLA: Concepts describing QoS parameters of interoperability e.g. reliability, availability, response time, etc.
  - Trading & Bartering: Concepts like cost and utility functions, algorithms, etc.
- Security
  - Identity & Access Management: Concepts needed for IAM like roles, access rights, etc.
  - Encryption: Abstract concept, may include concrete encryption algorithms.

We then asked the project partners to state their opinion whether these domains must, should, or could be part of the core ontology. The outcome is shown in Figure 11, which shows that location and time are believed to be the most important concepts together with units of measurement and sensors. For identity & access management, actuators and symbIoTe infrastructure there is a strong position that these concepts should be part of the CIM. Services and Protocols are open for discussion as at most 50% of partners think that this should be an essential element of the CIM. The same goes for Bartering & Trading. Encryption & Quality of Service / SLA is not considered part of the CIM.

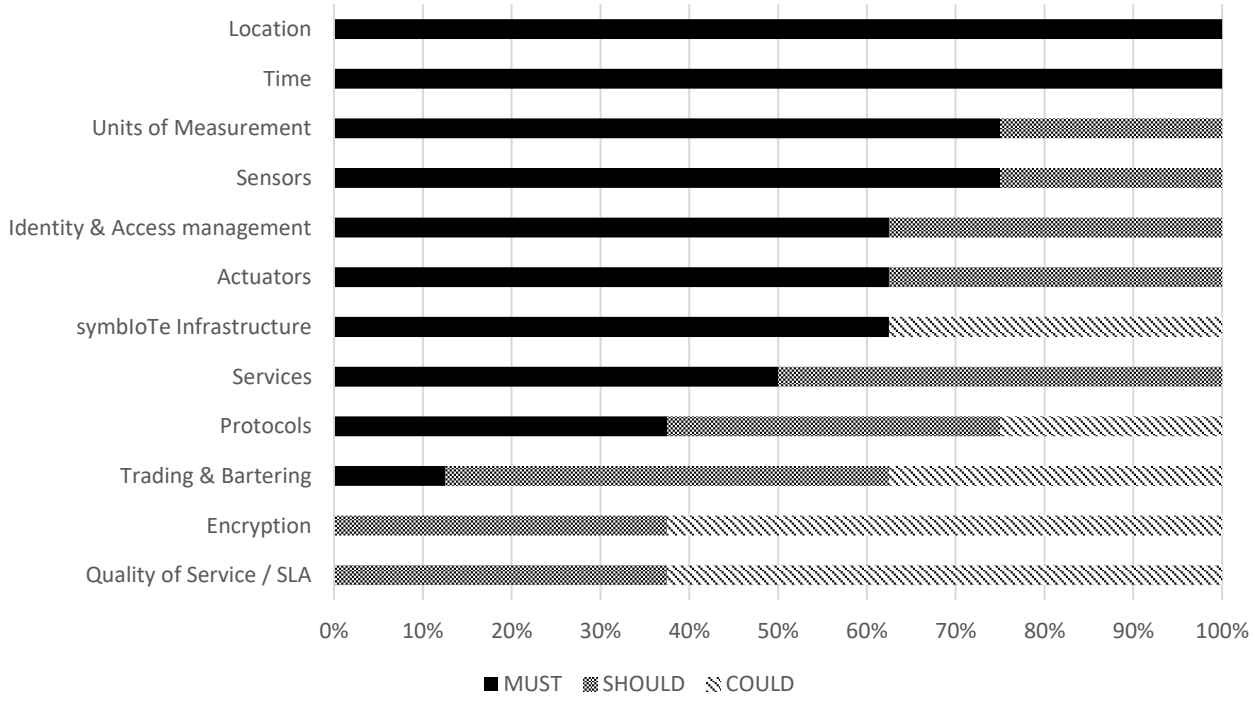


Figure 11 Outcome of the internal survey on which domains must/should/could be modelled within symbIoTe.

5.2 Core Information Model

The Core Information Model (CIM) is the central information model of symbIoTe and serves a two-fold purpose. First, it defines all the terms (classes and their relations) that symbIoTe components, especially the Core, can understand. The second purpose is that it serves as a shared vocabulary between all symbIoTe-compliant platforms and thereby enables out-of-the-box interoperability between them. Therefore, it is designed as a trade-off between a detailed and sophisticated model, which is easy to use out-of-the-box, and a very abstract and general model, which allows for maximal interoperability. As ontology design is always an iterative process, it did undergo multiple changes and extension within seven iterations until the final version 1.0 of the CIM.

The central class of the model is *Resource* which is also used within the Meta Information Model to link a *Resource* belonging to a *Platform* (or more specific, to an implementation of the Interworking Interface; see following section for details). symbIoTe knows different types of resources, which are modelled as subclasses of *Resource* in the CIM. They are divided into *Service* and *Device*, which is further divided into *Sensor*, which can be a *MobileSensor* or a *StationarySensor*, and *Actuator*. For better overview, classes and relations belonging exclusively to one of these domains are highlighted using the same shade of green. Elements defined in other, already existing ontologies are highlighted in light grey. In the following, these three main domains are described in detail.

The sensor domain is strongly influence by the Semantic Sensor Network (SSN) ontology [12] as well as the Sensor-Observation-Sampling-Actuator (SOSA) ontology [15], which is part of the proposal for a re-design of the SSN ontology. Therefore, the basic classes and their relations of the SSN ontology can be found in the CIM: Sensor, Observation, ObservationValue, Property, FeatureOfInterest and UnitOfMeasurement. As the SSN ontology does not allow to define the capabilities of a sensor without having existing observations from that sensor, we added a relation between Sensor and Property as well as one between StationarySensor and FeatureOfInterest to support definition of properties (and feature of interest) a sensor can observe. For modelling observation times, we re-use the W3C Time Ontology[18].

The second major domain covers the actuating part and is based on the combination of two actuator models. From the Actuation-Actuator-Effect ontology pattern[19], which applies the Stimulus Sensor Observation pattern [16] to the actuation domain we took the idea of actuators triggering effects, and from the SOSA ontology the idea of linking actuators to a feature of interest and its properties. Therefore, the classes *FeatureOfInterest* and *Property* belong neither to the sensing nor to the actuating domain, but are rather used as a link between them. For better usability, we introduced the *Capability* class grouping multiple effects of an actuator in a re-usable block, which can be used in a functional interface style as they represent method definitions and link them with their real-world effect.

The third major domain is the service domain. In the CIM diagram, it is rather small as it consists only of a single class, the *Service* class, and four outgoing relations. This is because the most complexity of the service model is related to the modelling of parameters and datatypes, which is a crosscutting domain.

[18] https://www.w3.org/TR/owl-time/
[19] http://ontologydesignpatterns.org/wiki/Submissions:Actuation-Actuator-Effect

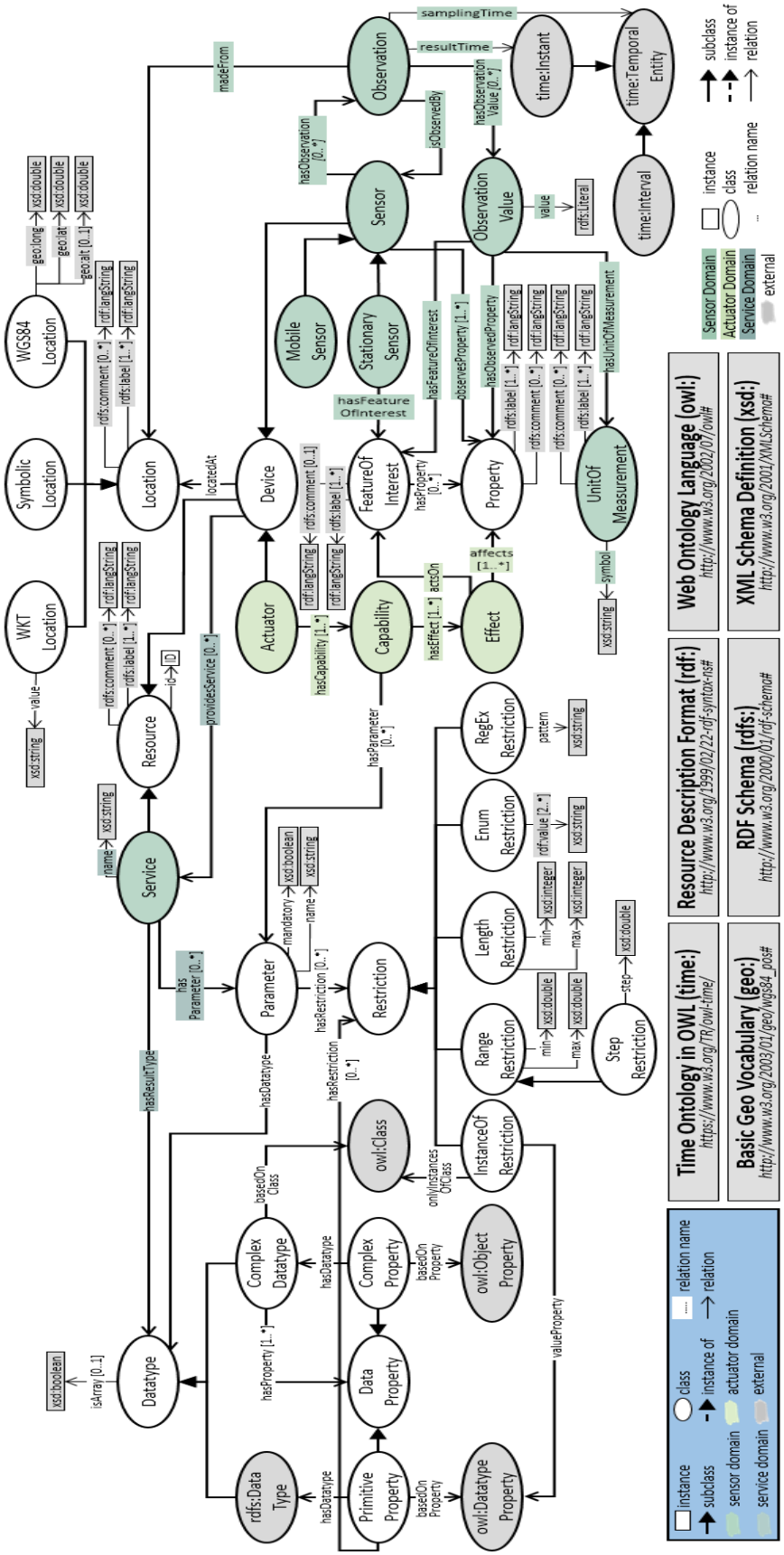


Figure 12 Core Information Model v1.0.

Besides the three major domains, there are even more classes and relations that belong to some crosscutting (sub) domains, i.e., they are being used in a subset of the three major domains. They can roughly be categorized into the domains datatypes, parameters and location. The datatype domain allows definition of primitive and complex datatypes, which can be used as datatype for in- and output parameters. The parameter domain allows definition of input parameters as well as fine-grained restrictions to those parameters. Both are use in the service, as well as the actuation domain.

The location domain allows definition of locations in three different types: by well-known text (*WKTLocation*), GPS position (*WGS84Location*) and as a symbolic location (*SymbolicLocation*). When one of the first two formats is used, symbIoTe can understand which data and devices can be found by location using the symbIoTe Search feature provided by its Core Services.

5.3 Meta Information Model

The Meta Information Model (MIM) is used to store metadata about platforms and resources within the symbIoTe Core components. Like the CIM, it was designed iteratively through three iterations. It started as a model linking platforms, information models and the mappings between information models. With version 0.2, it evolved to a more complex model as depicted in Figure 13.

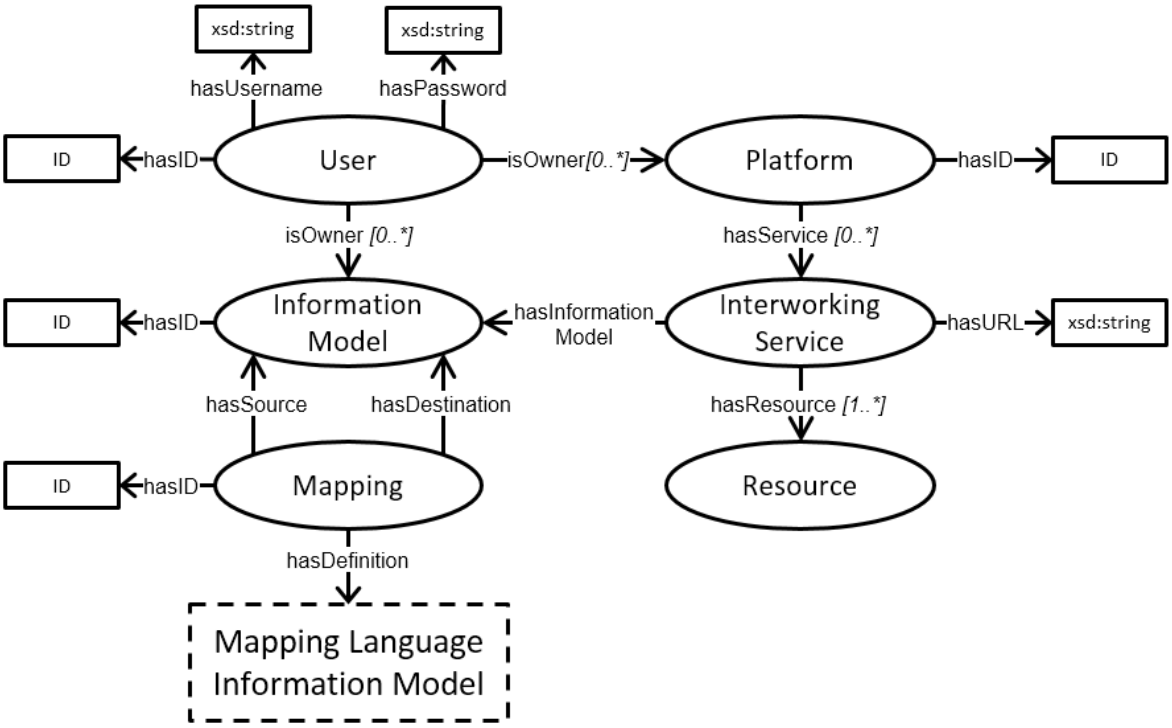


Figure 13 Meta Information Model v0.2.

MIM defines that a user can be the owner of multiple platforms and information models. Each platform can expose multiple implementations of the Interworking Interface (represented by class *InterworkingService*) which uses a specific information model and exposes some resources based on this model. Furthermore, the MIM allows specifying mappings between two information models that are defined using some sort of a mapping language.

The final version (v1.0) of the MIM is depicted in Figure 14. It is the same as v0.2 but with some extensions to support different tasks and functionalities within symbIoTe, e.g. federations and access management that will be presented hereafter.

Access Management is covered by the classes *AccessAttribute* and *MetaAccessAttribute*. *MetaAccessAttribute* allows defining platform-specific attributes together with value restrictions, while the class *AccessAttribute* is used for user-specific instantiation of the attributes carrying also the value of the attribute.

Federation Federations comprise a number of Interworking Interface implementations. They may agree on a common SLA that is stored as a string inside the MIM and can be defined using any suitable domain-specific language.

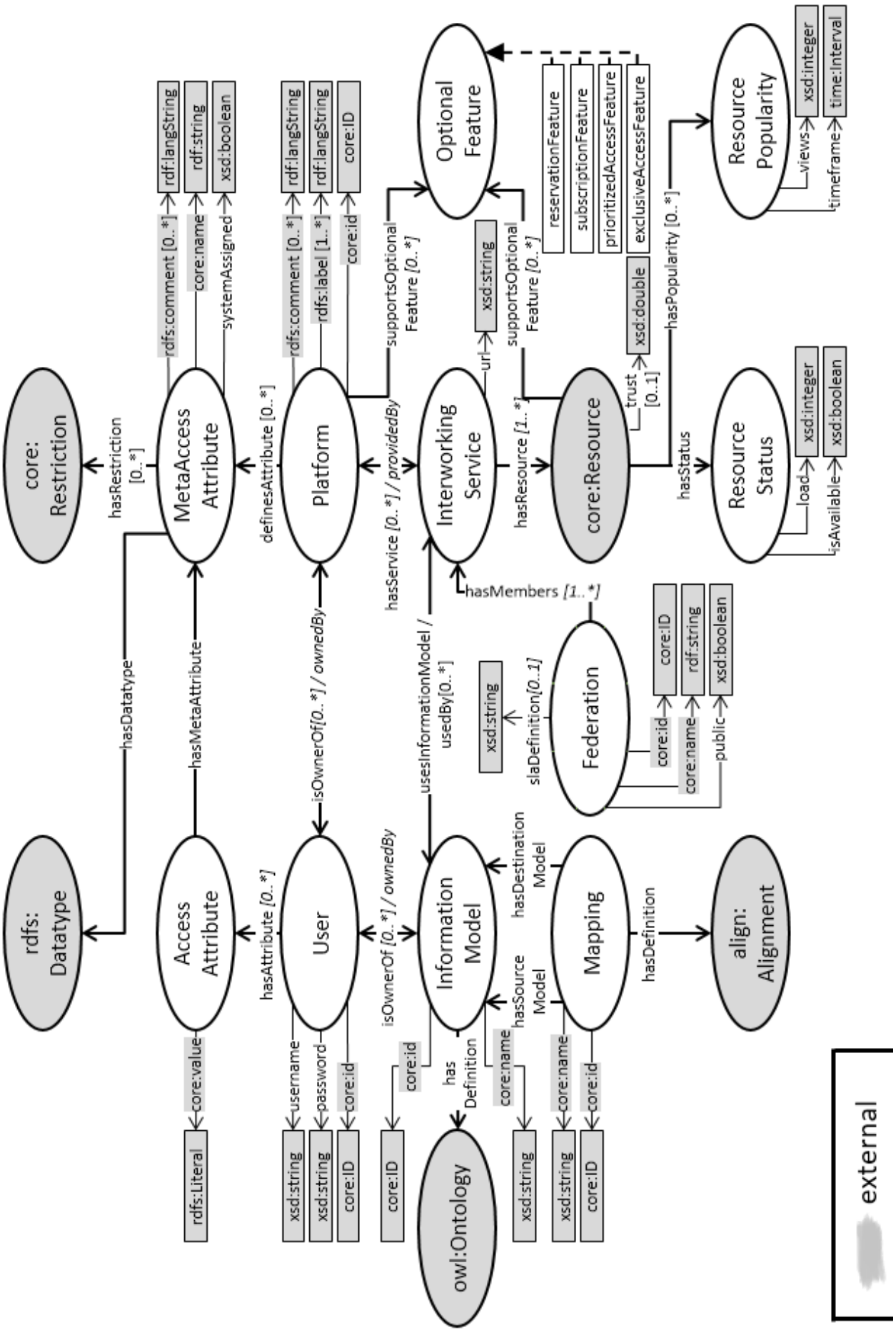
Optional Features are an extension that allows to model if a *Platform* or a *Resource* support different kind of features. This can be for example subscription to resources or prioritized or exclusive resource access.

Resource Status & Resource Popularity extension allows accessing the status and the popularity of a resource determined via monitoring.

Further, some additional changes are worth mentioning. First, each *InformationModel* now has a definition in the form of an OWL ontology definition and second, each Mapping has a definition in the form of an *Alignment*, which links to the ontology of the Alignment API [17].

5.4 Platform-Specific Information Models

As explained in Section 5 and depicted in Figure 10, symbIoTe allows platforms to define custom extensions of the CIM called Platform-Specific Information Models (PIMs). A PIM must be an extension of the CIM, meaning that it is not allowed to alter or change parts of the CIM. PIMs are intended to solely contain additional classes and predicates that are necessary for describing the data provided by a specific platform. This enables platforms to describe their domain with an adequate level of detail. It is possible for multiple platforms to use the same PIM. Actually, such approach leads to *interoperability by standardization* which is encouraged to be used whenever possible.



5.5 Best-Practice Information Model

The Best Practice Information Model (BIM) is a special kind of PIM that is designed to cover the domains of the use cases. As stated previously in Section 5, symbIoTe supports two kinds of interoperability: interoperability by standardization and interoperability by mapping. In four out of five use cases, the interoperability by standardization is used. According to this, the BIM is used as a “standardized” or “agreed-upon” information model throughout those four use cases. The fifth use case, the EduCampus use case, will be realized using the semantic mapping approach described in Section 5.7.3. The symbIoTe use cases are presented in detail in D1.3 “Final Specification of Use Cases and Initial Report on Business Models”.

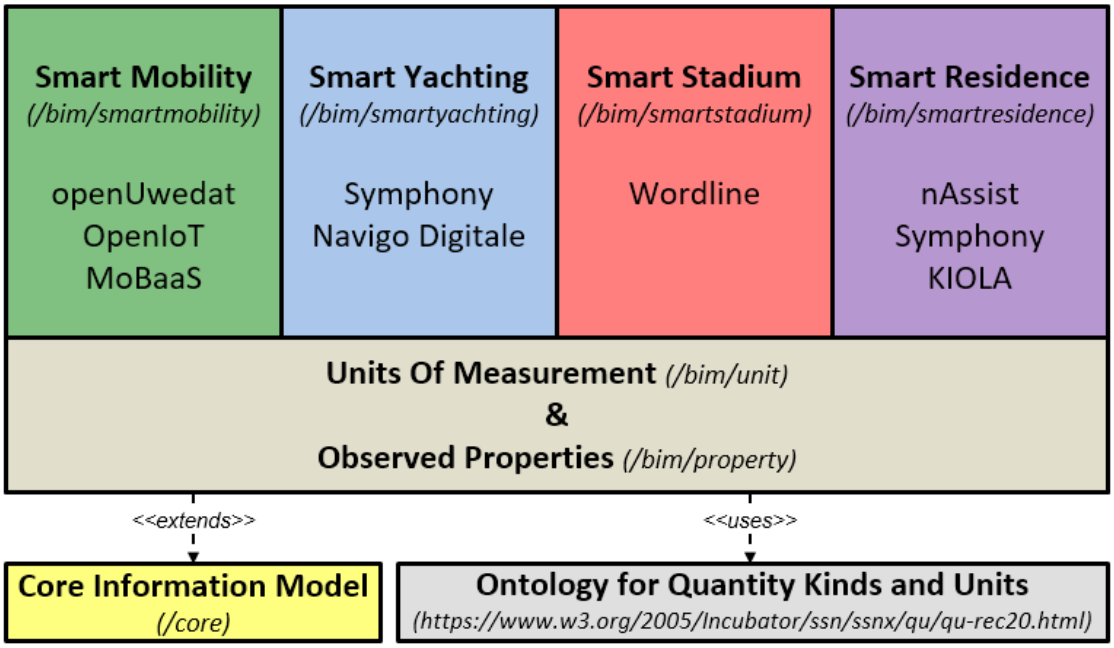


Figure 15 Structure of the Best Practice Information Model

Figure 15 depicts the structure of the BIM. As every PIM, it is an extension of the CIM. Additionally, it uses the Ontology for Quantity Kinds and Units (QU ontology), which is an ontology defining units and physical properties. Based on the QU ontology, the BIM provides a common layer defining additional units of measurements and observed properties that are shared across all of the use case domains.

On the top of that common layer, we have four separate parts of the BIM, one for each of the four use case domains: Smart Mobility, Smart Yachting, Smart Stadium and Smart Residence. Below the use case name stands the relative URI of the model, followed by the list of platforms brought into symbIoTe by partners that will be used within the use case. These parts of the BIM where developed separately by the use case owners in tight collaboration with T2.1 in an iterative manner. They are use-case-specific and cover two aspects: 1) Which data does an application/enabler (the concept of enablers within symbIoTe is explained in detail in D2.3 “Report on symbIoTe Domain-Specific Enablers and Tools”) need to understand from a platform? 2) If an enabler is used, what does an app need to understand about the offered services from an enabler?

It might be puzzling for some readers that these models appear sometimes a bit “disconnected”. The reason for this is that it is not a model in the classical sense of an

UML diagram but rather a collection of terms, called a vocabulary, need to describe the domain and needed for the services provided in the use cases. Although one does not see it at a first glance, these models are completely connected (between their components). This means that each element is reachable from each other element by some path. The main reason it may appear unconnected is that many connections are only created upon usage of the model, e.g. registering resources of a platform that is using the BIM within symbIoTe. Another reason is, that the models are quite complex and displaying them in a graphical way is not always appropriate, as it would get confusing. Therefore, at some points, the models are depicted in a partially informal and incomplete way (e.g. by including tables into the figures). For better understanding of connections that are made at registration time of resources, we also provide a number of small RDF examples how the payload of such a registration could look like.

In the following sections, we will present the different parts of the BIM in detail. However, they are not explained and described down to the smallest detail, as this would go beyond the scope of this document. Please refer to the ontology definitions available on GitHub[20] for the most detailed information about the BIM (and other information models in symbIoTe).

5.5.1 Units Of Measurement & Observed Properties

The BIM re-uses an existing ontology called Ontology for Quantity Kinds and Units (QU ontology) for describing units of measurement and observable properties. It is partially based on the OMG SysML QUDV (Quantities, Units, Dimensions and Values) which is part of the OMG SysML 1.2 standard [18]. It has been made available by the W3C Semantic Sensor Network Incubator Group[21]. In symbIoTe, it is used as a structured collection of units and observable properties.

5.5.1.1 Units of Measurement

To identify which units of measurement are needed within the BIM, we gathered information about which units are used by existing platforms or needed within a use case from platform and from use case owners. Figure 16 shows all the needed units and the alignment of the QU ontology with the CIM. Units already defined within the QU ontology are displayed in light grey whereas units defined by the BIM are display in white. The newly defined units are aligned within the taxonomy of the QU ontology.

5.5.1.2 Observed Properties

For observed properties, we also gathered the needs of platform and use case owners and tried to re-use what is already present within the QU ontology. Figure 17 shows only some of these properties as the Smart Mobility use case relies on the list of air quality pollutants as defined by the European Environment Agency[22] which comprises around 500 different observable properties for air quality. Furthermore, it shows the alignment of the QU ontology with the CIM through making *qu:QuantityKind* a subclass of *core:Property*. As the

[20] https://github.com/symbiote-h2020/Ontologies
[21] https://www.w3.org/2005/Incubator/ssn/
[22] http://dd.eionet.europa.eu/vocabulary/aq/pollutant/

QU ontology uses SKOS to classify properties, we decided to also do this for the observed properties within the BIM and added respective instances of *skos:Scheme*.

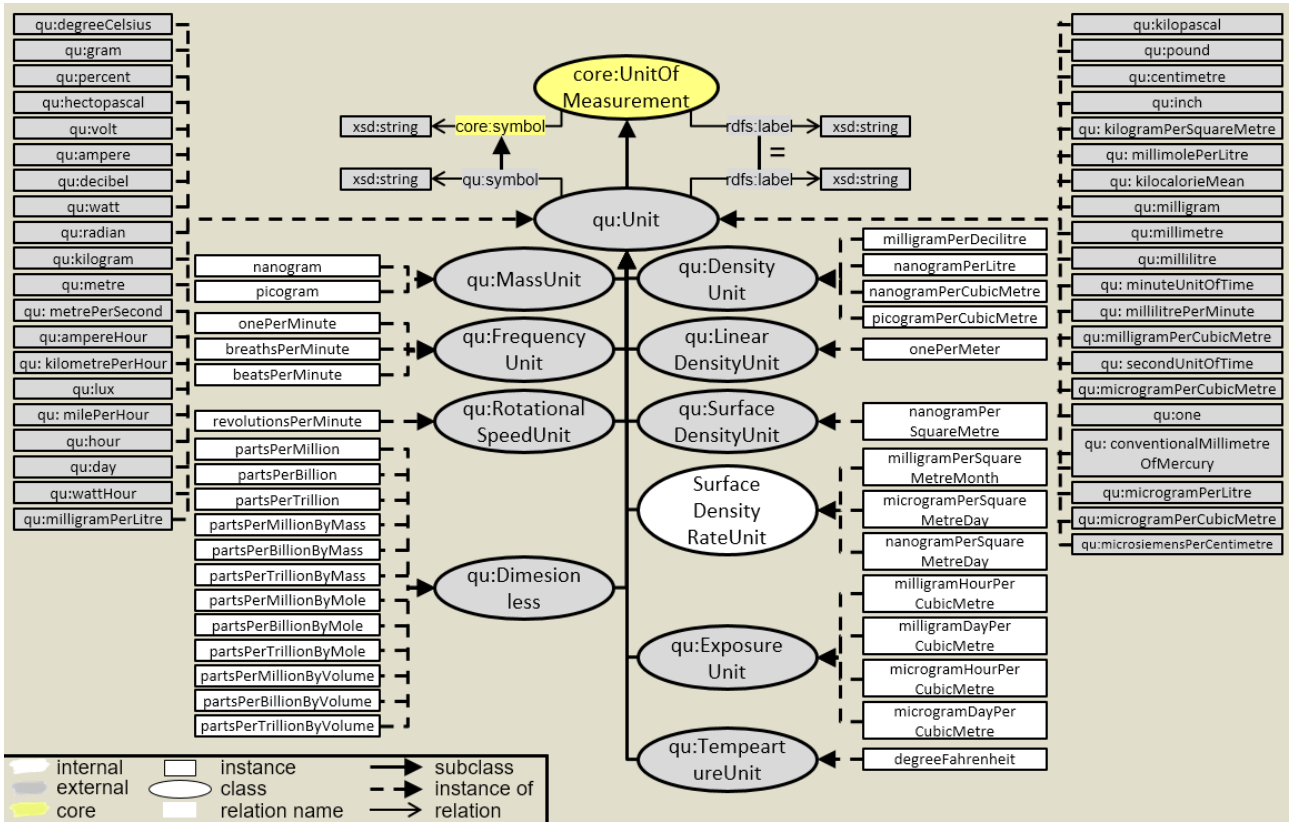


Figure 16 Additional units of measurements defined in the common layer of the BIM.

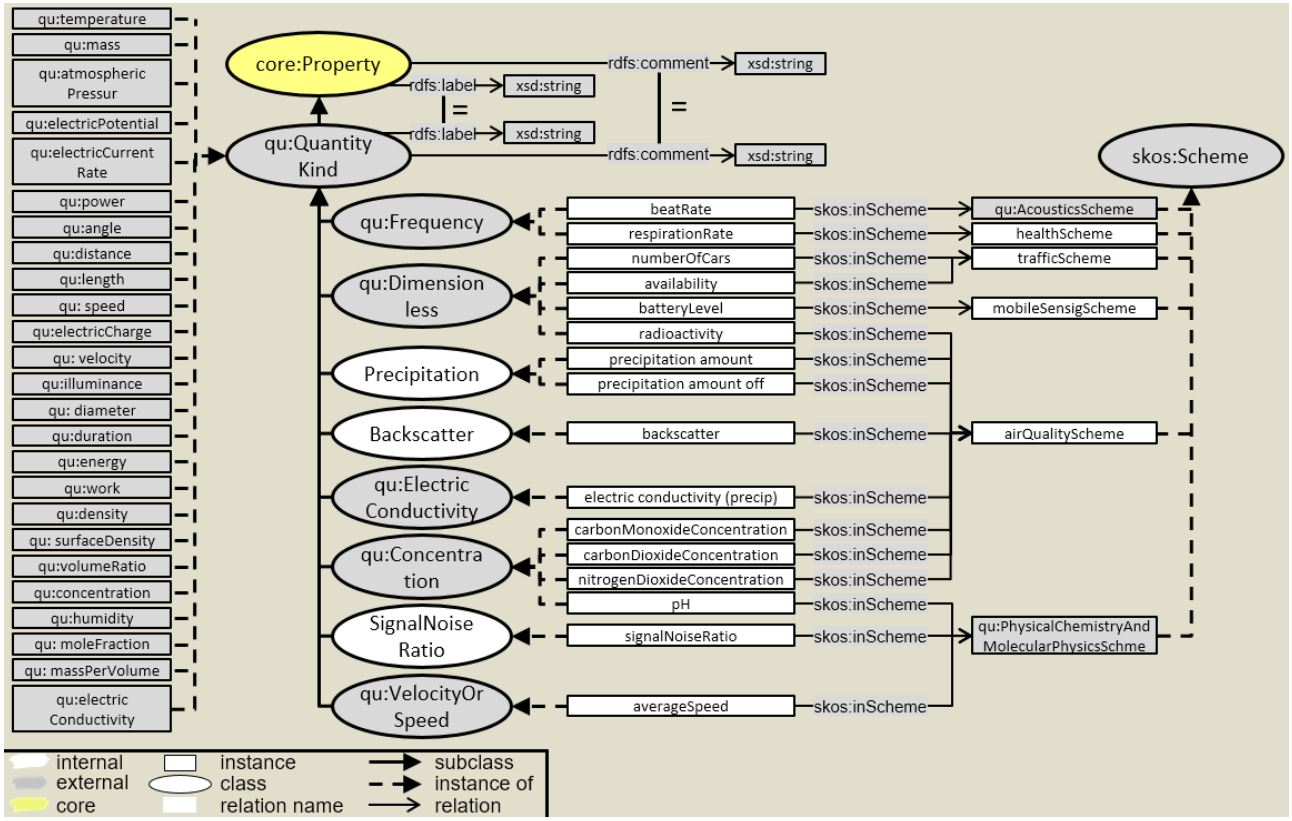


Figure 17 Additional observed properties defined in the common layer of the BIM.

5.5.2 Smart Mobility Domain

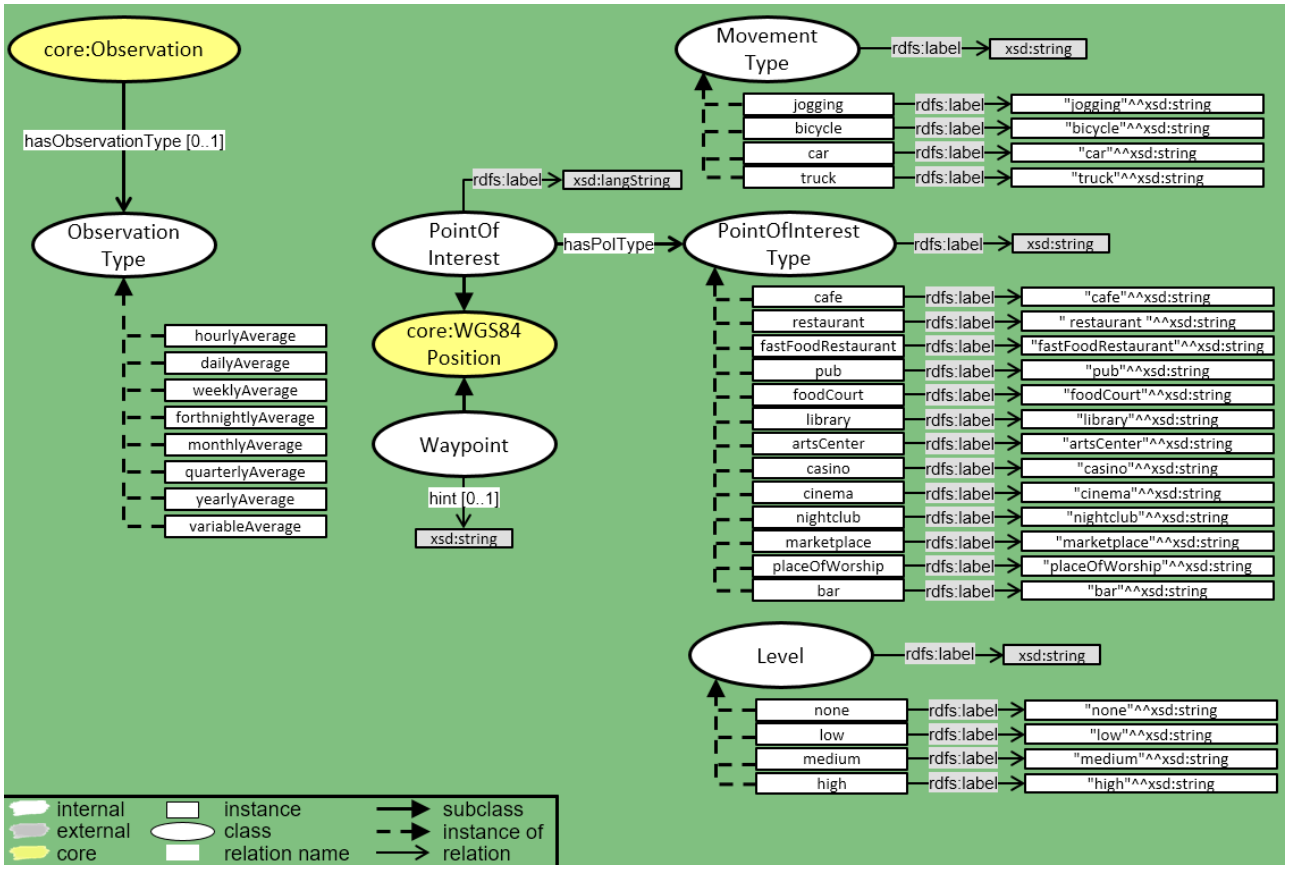


Figure 18 The Smart Mobility domain model of the BIM.

The Smart Mobility and Ecological Routing use case addresses the problem of inefficient transportation and poor air quality that many European cities face nowadays. This use case offers the ecologically most preferable routes for motorists, bicyclists and pedestrians based on the available traffic and environmental data acquired through various platforms. This scenario is extremely relevant for people who travel within the major European cities, since a constant exposure to pollutants can cause severe health problems. It is also of interest to the municipalities' governing bodies that, by helping their citizens to avoid these health problems, they can reduce health care costs. Additionally, the use case will provide a way for users to search for Points of Interests, filtered by certain factors such as air quality, noise pollution and parking availability. symbIoTe will empower this use case by providing platform interoperability, allowing developers to easily access and handle data from different platforms and domains in the same manner.

The Smart Mobility domain model is depicted in Figure 18. It is rather simple as the basic functionality needed from the platforms for this use case is air quality sensing (which is already cover by the CIM). The only extension needed to the sensing domain is the added optional *ObservationType* of an observation. The classes *MovementType*, PointOfInterest, *PointOfInterestType*, *Level* and *Waypoint* are only used to describe in- and output parameter of the services offered by the enabler for this use case.

Listing 2 shows an example definition for the registration of a stationary CO2 sensor. Besides the definition of the observed property *bim:carbonDioxideConcentration* only terms already defined in the CIM are used.

```
sensorX a owl:NamedIndividual ;
    a core:StationarySensor ;
    core:observesProperty bim:carbonDioxideConcentration ;
    core:locatedAt [ a WGS84Location ;
        geo:lat "48.2081743"^^xsd:double ;
        geo:long "16.3738189"^^xsd:double
    ] ;
    core:hasFeatureOfInterest [ a WGS84Location ;
        geo:lat "48.2081743"^^xsd:double ;
        geo:long "16.3738189"^^xsd:double
    ] .
```

Listing 2 Example stationary CO2 sensor definition.

Listing 3 and Listing 4 are more complex examples of an RDF specification used to register the *calculateGreenRoute* and *pointOfInterestSearch* services offered by the enabler within symbIoTe.

The *calculateGreenRoute* service computes a route between two points taking into account the transportation method preferred by a user. It takes three parameters as input:

- **start** and **end**, which have a complex datatype based on *core:WGS84Position* but using only its *geo:lat* and *geo:long* attributes. They represent the locations of the start and the end of the desired route.
- **movementType** is a string-valued parameter which only accepts the label-values of instances of the class *MovementType*, which represents the preferred method of transportation of the user.

The return type of the service is again a complex datatype based on the class *Waypoint* containing the attributes *geo:lat* and *geo:long* from *WGS84Position* together with an optional hint of type string.

The *pointOfInterestSearch* searches for PoIs within a certain area following user's preferences. It takes as input parameters:

- **location**, representing the centre of the area where the user wants to find PoIs and represented as a complex datatype based on *core:WGS84Position*
- **radiusInMeter** an integer value that represents the radius in meters around the point defined by location that should be considered
- **POIType** a string-valued parameter representing the type of PoI the user is looking for; realized as complex data with a restriction to instances of the class *PointOfInterestType*
- **propertyType** a string-valued parameter that indicates the type of sensor which will aid in the search of the PoI
- **propertyLevel** a string-valued parameter which only accepts the label-values of instances of the class *Level* and it indicates the level of a certain amount of an observed property indicated in *propertyType*

The service will return an array of objects of the class *PointOfInterest*, with their respective name, position and type.

```
calculateGreenRouteService a owl:NamedIndividual ;
    a core:Service ;
    core:name "calculateGreenRoute"^^xsd:string ;
    core:hasParameter [ a core:Parameter ;
        core:name "start"^^xsd:string ;
        core: mandatory "true"^^xsd:boolean ;
        core:hasDatatype [      a core:ComplexDatatype ;
            core:basedOnClass core:WGS84Location ;
            core:hasProperty [ a core:PrimitiveProperty
                core:hasDatatype xsd:double ;
                core:basedOnProperty geo:lat
            ] ;
            core:hasProperty [ a core:PrimitiveProperty
                core:hasDatatype xsd:double ;
                core:basedOnProperty geo:long
    ]]] ;
    core:hasParameter [ a core:Parameter ;
        core:name "end"^^xsd:string ;
        core: mandatory "true"^^xsd:boolean ;
        core:hasDatatype [      a core:ComplexDatatype ;
            core:basedOnClass core:WGS84Location ;
            core:hasProperty [ a core:PrimitiveProperty
                core:hasDatatype xsd:double ;
                core:basedOnProperty geo:lat
            ] ;
            core:hasProperty [ a core:PrimitiveProperty
                core:hasDatatype xsd:double ;
                core:basedOnProperty geo:long
    ]]] ;
    core:hasParameter [ a core:Parameter ;
        core:name "MovementType" ;
        core:mandatory "false"^^xsd:boolean ;
        core:hasDatatype xsd:string ;
        core:hasRestriction [ a InstanceOfRestriction ;
        core:onlyInstancesOfClass bim:MovementType ;
        core:valueProperty rdfs:label
    ]] ;
    core:hasReturnType [ a core:ComplexDatatype ;
        core:basedOnClass bim:Waypoint ;
        core:hasProperty [ a core:PrimitiveProperty
            core:hasDatatype xsd:double ;
            core:basedOnProperty geo:lat
        ] ;
        core:hasProperty [ a core:PrimitiveProperty
            core:hasDatatype xsd:double ;
            core:basedOnProperty geo:long
        ] ;
        core:hasProperty [ a core:PrimitiveProperty
            core:hasDatatype xsd:string ;
            core:basedOnProperty bim:hint
    ]] .
```

Listing 3 Example registration of a *calculateGreenRoute* service offered by the enabler.

```
poiSearchService a owl:NamedIndividual ;
      a core:Service ;
      core:name "pointOfInterestSearch"^^xsd:string ;
      core:hasParameter [ a core:Parameter ;
            core:name "location"^^xsd:string ;
            core: mandatory "true"^^xsd:boolean ;
            core:hasDatatype [      a core:ComplexDatatype ;
                  core:basedOnClass core:WGS84Location ;
                  core:hasProperty [ a core:PrimitiveProperty
                        core:hasDatatype xsd:double ;
                        core:basedOnProperty geo:lat
                  ] ;
                  core:hasProperty [ a core:PrimitiveProperty
                        core:hasDatatype xsd:double ;
                        core:basedOnProperty geo:long
      ]]] ;
      core:hasParameter [ a core:Parameter ;
            core:name "radiusInMeter"^^xsd:string ;
            core: mandatory "true"^^xsd:boolean ;
            core:hasDatatype xsd:string
      ] ;
      core:hasParameter [ a core:Parameter ;
            core:name "pointOfInterestType" ;
            core:mandatory "false"^^xsd:boolean ;
            core:hasDatatype xsd:string ;
            core:hasRestriction [ a InstanceOfRestriction ;
                  core:onlyInstancesOfClass bim:PointOfInterestType ;
                  core:valueProperty rdfs:label
      ] ;
      core:hasParameter [ a core:Parameter ;
            core:name "propertyType"^^xsd:string ;
            core: mandatory "false"^^xsd:boolean ;
            core:hasDatatype xsd:string ;
            core:hasRestriction [ a InstanceOfRestriction ;
                  core:onlyInstancesOfClass core:Property ;
                  core:valueProperty rdfs:label

      ]] ;
      core:hasParameter [ a core:Parameter ;
            core:name "propertyLevel" ;
            core:mandatory "false"^^xsd:boolean ;
            core:hasDatatype xsd:string ;
            core:hasRestriction [ a InstanceOfRestriction ;
                  core:onlyInstancesOfClass bim:Level ;
                  core:valueProperty rdfs:label
      ]] ;
      core:hasReturnType [ a core:ComplexDatatype ;
            core:basedOnClass bim:PointOfInterest ;
            core:hasDatatype [      a core:ComplexDatatype ;
                  core:basedOnClass core:WGS84Location ;
                  core:hasProperty [ a core:PrimitiveProperty
                        core:hasDatatype xsd:double ;
                        core:basedOnProperty geo:lat
                  ] ;

            core:hasProperty [ a core:PrimitiveProperty
                        core:hasDatatype xsd:double ;
                        core:basedOnProperty geo:long
```

Listing 4 Example registration of a *pointOfInterestSearch* service offered by the enabler.

5.5.3 Smart Yachting Domain

The Smart Yachting use case consists of the following two showcases:

- Smart Mooring aims to automate the mooring procedure of the port, which is quite a bureaucratic and tedious process (Marinas operate in strongly regulated contexts). For this showcase, Navigo is working to integrate its PortNet application, which is a workflow management system that controls and supervises a certain number of Port's authorization procedures.
- Automated Supply Chain aims to identify the needs for goods and services on board, by using IoT sensors, so that automated requests for offers can be issued in the marketplace service in the Port, implemented by another application developed by Navigo infrastructure (Centrale Acquisti).

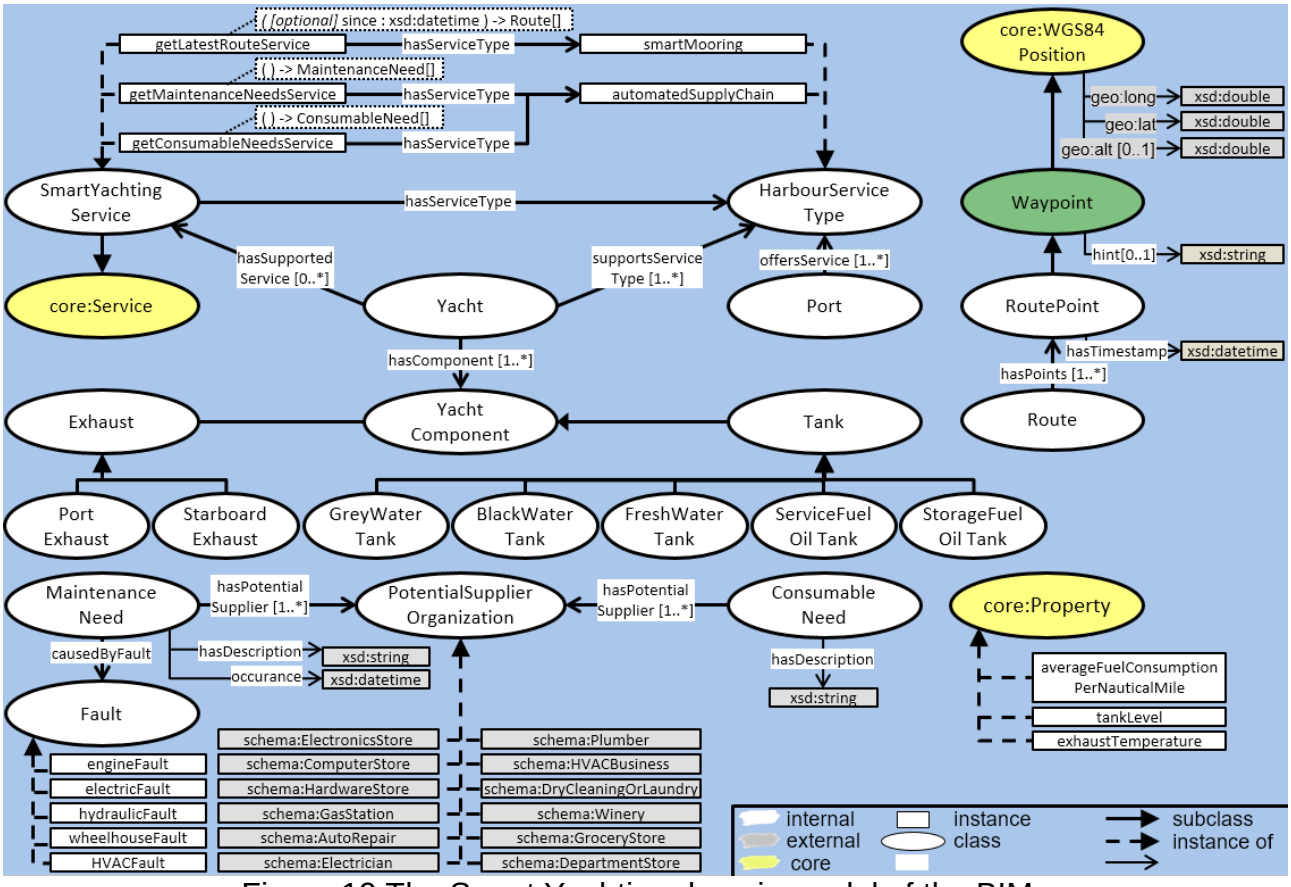


Figure 19 The Smart Yachting domain model of the BIM.

To better understand the model, it makes sense to provide a brief description of how these two showcases will work. They will both operate in a context that involves the following conditions:

- The Port has an IoT Platform.
- The Yacht has an IoT Platform on board.
- The platforms interoperate through symbIoTe.
- No need for the platforms to be the same, as long as they are both symbIoTe enabled (e.g. a vessel might have a simplified system like Navigo Digitale Yacht, whereas a yacht might have Nextworks' Symphony on board).

- The Port IoT Platform connects with sensors and "devices" in its area through both LoRaWAN (Long Range Wide Area Network) and WiFi.
- Navigo's applications, the aforementioned PortNet and Centrale Acquisti, will be both integrated in the symbIoTe ecosystem through Enablers. This should facilitate other actors (e.g. other ports with different applications) to implement the Smart Yachting use case by simply rewriting the integration logic between their applications and the enablers.

symbIoTe will allow to automatically retrieve yacht data that is essential to process the mooring workflow or the supply chain request.

For Smart Mooring it is also assumed that the Yacht is seen as a Smart Device and the port is a Smart Space (Level 3 compliance as explained in D1.4 "Final Report on System Requirements and Architecture"). Moreover, we assume that the Yacht maintains its ID when moving between Ports: the Yacht will be therefore seen as an example of a Roaming Device, achieving Level 4 symbIoTe compliance.

The Smart Yachting domain model is depicted in Figure 19. As shown, the model revolves around the concepts of Yacht, Port and Services. It was indeed important to provide a formal description of the kinds of services that a Yacht, on the one hand, supports, and a Port, on the other, offers. The services implemented in the Smart Yachting use case have been described as entities of the *HarbourServiceType* class. This allows the model to be easily extended over time.

Accessing yacht's machine data through symbIoTe will be different in the two showcases.

For Smart Mooring, the workflow application needs, on the one hand, to simply retrieve a set of machine data from the yacht sensors; on the other it also needs the latest route of the yacht (that is, the route from the latest port to the present position), which is of course a much richer type of information.

For the former case, the model introduces the concept of *YachtComponent* which features two main subclasses, one – *Exhaust* – to provide an indication of the current temperature of the two classic yacht exhausts, and the other – *Tank* – to measure the level of the different tanks within the boat.

Exhaust and *Tank* are superclasses that generalize the actual sensors that provide measurements: in this regard, each of their subclasses constitutes a FeatureOfInterest, linked to a specific Property of the yacht for which a machine value (*tankLevel*, *exhaustTemperature*) can be measured and provided.

Route on the other hand is a time-ordered sequence of geographic coordinates. Each position (*RoutePoint*) has an associated timestamp plus an actual geo-position. A specific service must be provided on the yacht side to provide the latest route (*getLatestRouteService*).

As said, the Automated Supply Chain showcase will be based on identifying, through IoT sensors, the needs for goods and services on board of a Yacht, so that automated requests for offers can be issued on the marketplace platform of the Port.

We identified two main kinds of needs, *MaintenanceNeed* and *ConsumableNeed*. The former will be always associated to a particular instance of the class *Fault* (e.g. *engineFault*, *electricFault*, etc). A *ConsumableNeed* more generically identifies a resupply

need on board, e.g. the need to purchase wine if sensors in a smart fridge detect a low number of bottles.

From a technical viewpoint, the problem of easily and effectively classifying the needs, (those necessary to the Yacht and those offered by local suppliers in the Port' Supply Chain platform) has been addressed by associating each one of them to a *PotentialSupplierOrganization*. The latter has instances, which are generic categories, used to identify a possible supplier. For this purpose, we decided to use a standard, general-purpose ontology like schema.org, given its diffusion in the IT community. For example, *http://schema.org/ElectronicsStore* can be used as a *PotentialSupplierOrganization* when a communication appliance like a router is broken while *http://schema.org/Winery* when the wine stocks are low.

Two specific services – *getMaintenanceNeedService* and *getConsumableNeedService* – will be exposed by the Yacht to provide information about the current needs on board.

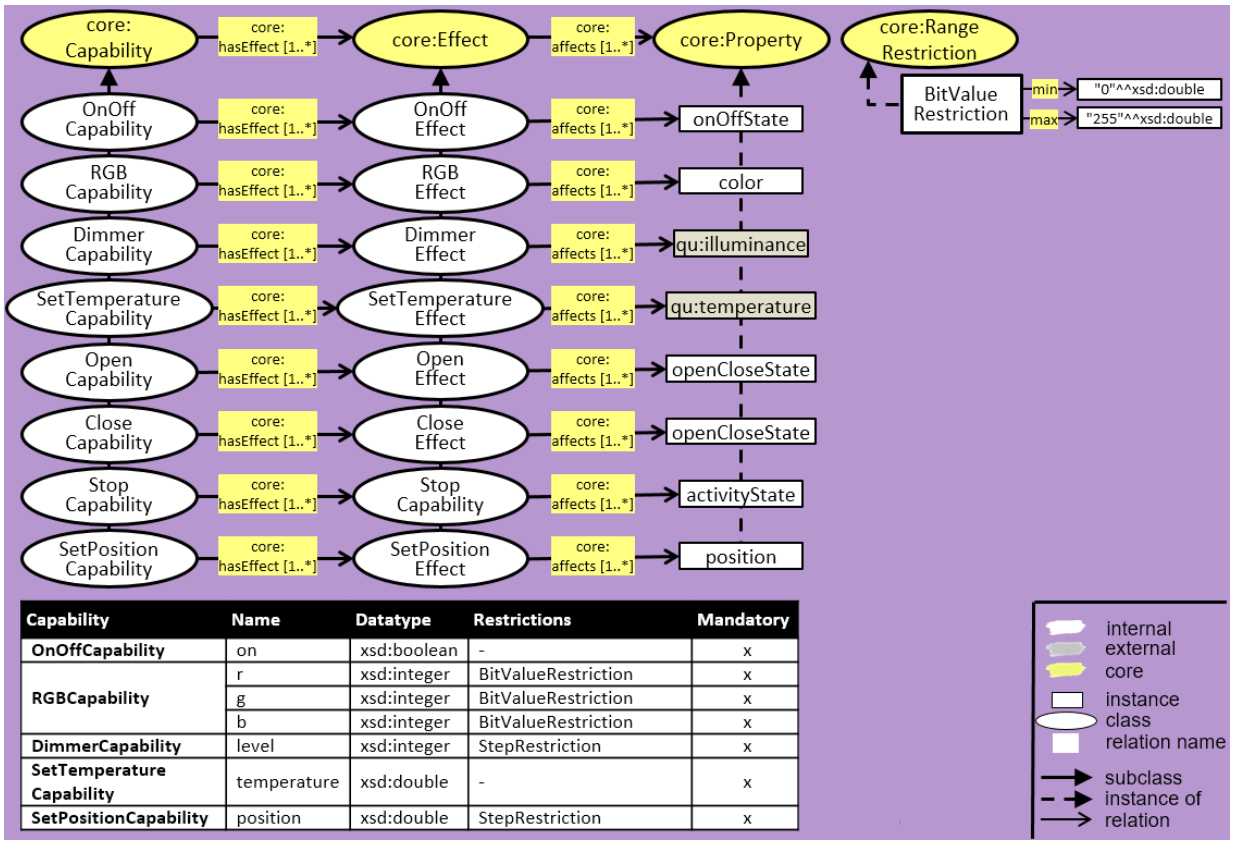
5.5.4 Smart Residence Domain

The Smart Residence use case aims to demonstrate interoperability across different smart home IoT solutions through a generalized abstract model to describe inter-connected objects, providing a dynamic configuration of available services and a natural and homogeneous user experience at home. Moreover, it shows how concepts of collecting health information (e.g. weight, blood pressure) can be embedded into a smart home. Depending on context and usage scenario, dynamically discovered functions will be presented on different devices (e.g. smart phones, TV screens, touch panels, smart objects), instantiated in a local/remote cloud and finally executed by the appropriate physical devices (e.g. light switches, speakers, displays, motor shades).

Residential automation involves a common collection of devices, which are usually installed in a domestic smart environment, i.e. lighting, thermostats and HVAC, motorized shades, curtains and blinds, sprinkler systems, and so forth. The Smart Residence domain model of the BIM aims to reflect the hardware generally provided for an automated home, in order to simplify device modelling to platform owners, which wants to join the symbIoTe environment.

This includes both devices (e.g. lights, fan-coils, curtains, environmental sensors, etc.) and their capabilities, since they affect specific properties of the ambience where they are installed in. Only domotics have been taken into consideration when designing this information model, because it is the main and mostly diffused context, while other areas like for example audio-video or video-surveillance have been excluded, in order to keep the model simpler.

Particular attention has been given to actuators, which are described with a set of capabilities, having effects on a single or a set of properties, as depicted in Figure 21. This allows registering a device that can affect multiple properties on different Feature of Interests, which is shown in detail in the two RDF examples given in Listing 5 and Listing 6.



Capability	Name	Datatype	Restrictions	Mandatory
OnOffCapability	on	xsd:boolean	-	x
RGBCapability	r	xsd:integer	BitValueRestriction	x
	g	xsd:integer	BitValueRestriction	x
	b	xsd:integer	BitValueRestriction	x
DimmerCapability	level	xsd:integer	StepRestriction	x
SetTemperature Capability	temperature	xsd:double	-	x
SetPositionCapability	position	xsd:double	StepRestriction	x

Figure 20 The actuation part of the Smart Residence domain model of the BIM.

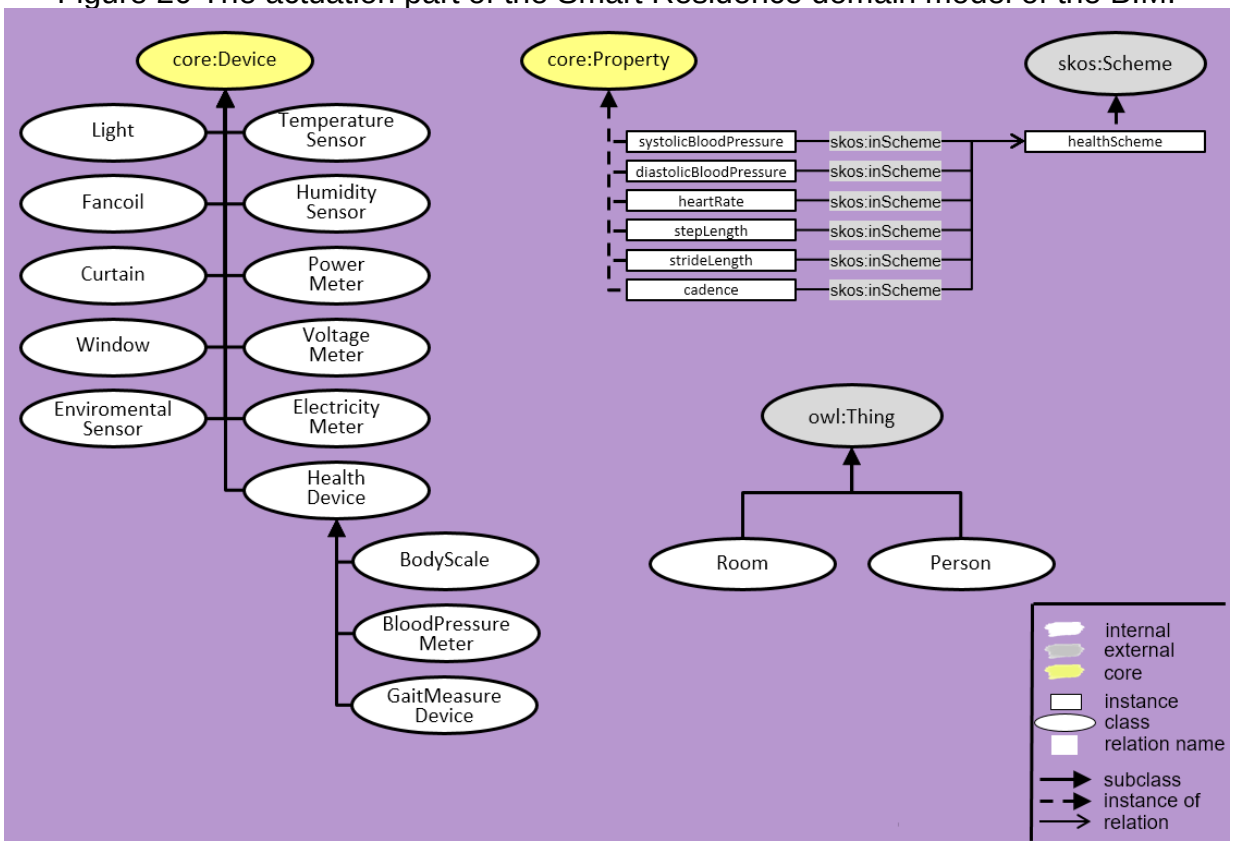


Figure 21 The devices and properties part of the Smart Residence domain model of the BIM.

```
roomX a owl:NamedIndividual ;
    a bim:Room .

lightX a owl:NamedIndividual ;
    a Light ;
    a core:Actuator ;
    a core:FeatureOfInterest ;
    core:hasCapability [ a OnOffCapability ;
        core:hasEffect [ a OnOffEffect ;
            core:actsOn lightX
        ] ;
        core:hasEffect [ a Effect ;
            core:affects illuminance ;
            core:actsOn roomX
    ]] ;
    core:hasCapability [ a DimmerCapability ;
        core:hasEffect [ a DimmerEffect ;
            core:actsOn roomX
    ]] ;
    core:hasCapability [ a RGBCapability ;
        core:hasEffect [ a RGBEffect ;
            core:actsOn lightX
        ]
        core:hasEffect [ a Effect ;
            core:affects illuminance ;
            core:actsOn roomX
    ]] .
```

Listing 5 Example RDF registration payload for a light actuator with on/off, RGB and dimmer functionality.

```
roomX a owl:NamedIndividual ;
    a bim:Room .

lightY a owl:NamedIndividual ;
    a Light ;
    a core:StationarySensor ;
    a core:Actuator ;
    a core:FeatureOfInterest ;
    core:hasCapability [ a OnOffCapability ;
        core:hasEffect [ a OnOffEffect ;
            core:actsOn lightY
        ]
        core:hasEffect [ a Effect ;
            core:affects qu:illuminance ;
            core:actsOn [
                a owl:NamedIndividual ;
                a bim:Room
    ]]] ;
    core:observesProperty qu:temperature ;
    core:hasFeatureOfInterest lightY .
```

Listing 6 Example RDF registration payload for a combined light sensor and actuator with on/off functionality and internal temperature sensor.

Figure 21 presents the device classification used as well as properties used in the health and ambient-assisted-living subdomain. Although it is not visualized as a separate part, tracking of health-related information at a user's home can be embedded in a smart home environment in this use case. Following the concepts of the CIM, it defines a *Person*

providing several health-related properties, namely blood pressure (*systolicBloodPressure* and *diastolicBloodPressure*), the heart rate as well as gait measurements (*stepLength*, *strideLength* and *cadence*). These properties are observed by *HealthDevices* (a *BodyScale*, *BloodPressureMeter* and a *GaitMeasureDevice*).

5.5.5 Smart Stadium Domain

Smart Stadium enhances the user experience of visitors coming to a stadium. In the retail context, it provides communication between visitors and retailers across large distances in the stadium.

Visitors are identified by their smartphones while retailers (both moving carts and physical shops) are identified by their POS (Point of Sale) Terminal and nearby located beacons.

From the visitors' point of view, Smart Stadium brings the opportunity of detecting closest retailers, place orders with them wherever they are and to receive the products they bought directly in their seat.

On the other hand, retailers can broadcast their offers and promotions to all visitors inside the stadium, or moving near specific areas inside the stadium. Retailers can send their promotions to large SmartTVs, named Promowalls, spread throughout the stadium.

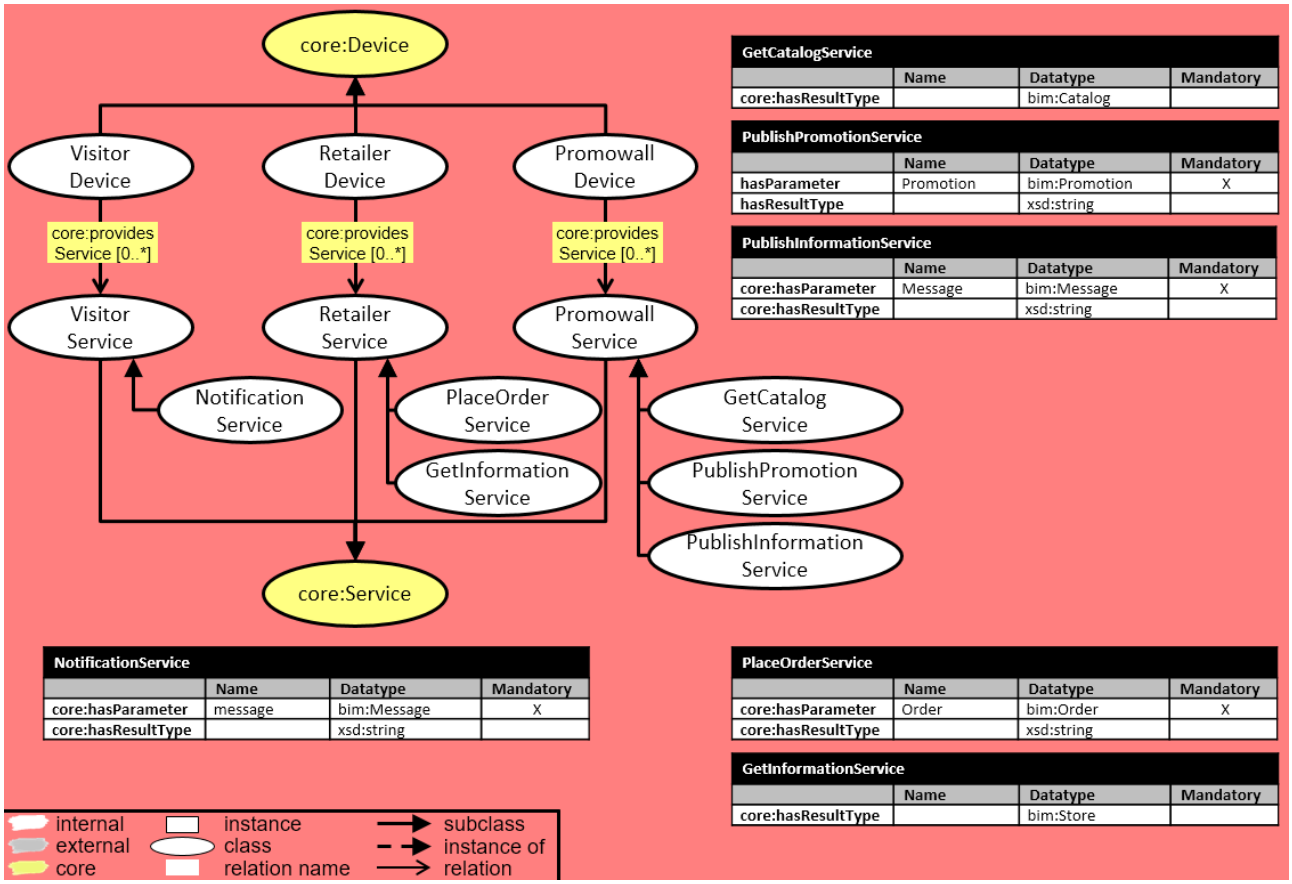


Figure 22 The device and service part of the Smart Stadium domain model of the BIM.

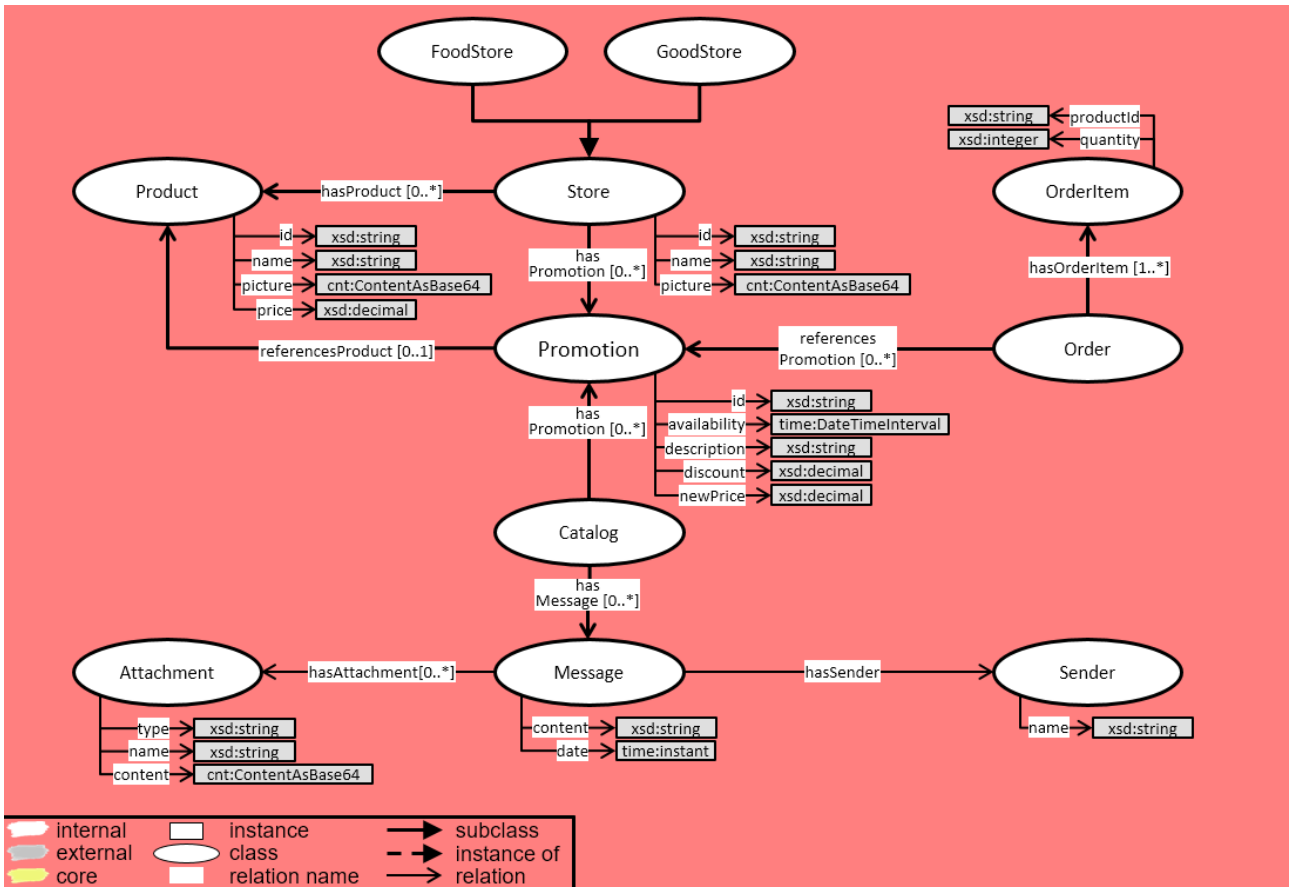


Figure 23 The parameter classes of the Smart Stadium domain model of the BIM.

Once all the participants of the use case were identified, four platforms came up:

- Visitor platform: in charge of taking care of all visitor devices, as well as providing backend information.
- Remote Ordering platform: manages all retailers and their devices and provides access to them.
- Promotion and Information platform: provides access and contents for all Promowalls inside the stadium.
- Beacon platform: provides an extensive catalogue of beacons spread throughout the stadium.

The Smart Stadium BIM uses the generic *Device* class from the CIM to model the IoT devices involved in this use case.

Each intelligent device provides services to interact with them, either to gather information or sending information. All services listed in Figure 22 cover the functionalities described above.

On the other hand, Figure 23 illustrates the data model used in this use case. As this figure depicts, the model is coupled across platforms as all of them are involved in commercial processes.

The model is straightforward to understand:

- The top half of the picture describes a simple model for shops, products, promotions and orders:
  - In Smart Stadium, shops offer a list of products to be sold and can offer promotions associated to them. There are two types of shops: physical shops depending on the products they sell: *FoodStore* and *GoodStore* (assuming sport-related products).
  - Furthermore, the model illustrates the information (named *Catalog*) available on each *Promowall*: promotions associated to shops and products as well as relevant information (messages) of the stadium and sport events.

It is relevant to mention that initially we evaluated using either GoodRelations[23] or schema.org to model our retail-scoped data model.

GoodRelations is the most powerful Web vocabulary for e-Commerce, and appeared to be a fitting solution for us. The same could be said for schema.org that describes multiple entities we need like *Order*, *Message*, *Attachment*, *Store*, etc.

Nevertheless, we decided to create our own model with our own versions of those classes. The reason for this was, in short, that the models from GoodRelations and schema.org are too complex. Generally, complexity is not a bad thing for such a model. However, in this special case, the classes depicted in Figure 23 are used to describe to data structures used as parameters and return values of the smart stadium services. Making them more complex as needed would mean to complicate the usage dramatically. For example, the class *Message* is also defined with schema.org and it also contains a property *sender*. The type of sender within schema.org is however, *(Audience or Organization or Person)* which are all again complex types with many properties that are not needed within this use case. As these classes are used to describe the allowed (JSON) structure of parameters, this makes quite a difference as it would need to change the structure of the JSON element. For this reason, we decided not to re-use the existing classes directly but to re-create custom versions that are strongly based on the original ones, but simpler and adapted to the concrete use case. However, alignment can easily be achieved later on (at least to some degree), e.g., by linking similar concepts with *owl:sameAs*.

Listing 7 and Listing 8 give an example definition of the *GetInformationService* and *PlaceOrderService* definition. Listing 9 shows registration of an example retailer device providing these services.

[23] http://www.heppnetz.de/projects/goodrelations/

```
 GetInformationService a owl:Class ;
      rdfs:subClassOf bim:RetailerService ;
      core:name "getInformation"^^xsd:string ;
      core:hasReturnType [ a core:ComplexDatatype ;
           core:basedOnClass bim:Store ;
           core:hasProperty [ a core:PrimitiveProperty ;
                core:hasDatatype xsd:string ;
                core:basedOnProperty bim:id ;
           ] ,
           core:hasProperty [ a core:PrimitiveProperty ;
                core:hasDatatype xsd:string ;
                core:basedOnProperty bim:name ;
 ]] .
```

Listing 7 Definition of *GetInformationService* class within the BIM.

```
PlaceOrderService a owl:Class ;
     rdfs:subClassOf bim:RetailerService ;
     core:name "placeOrder"^^xsd:string ;
     core:hasParameter [ a core:ComplexDatatype ;
          core:basedOnClass bim:Order ;
          core:hasProperty [ a core:ComplexProperty ;
               core:basedOnProperty bim:hasOrderItem ;
               core:hasDatatype [ a core:ComplexDatatype ;
                    core:basedOnClass bim:OrderItem ;
                    core:hasProperty [ a core:PrimitiveProperty :
                         core:basedOnProperty bim:quantity ;
                         core:hasDatatype xsd:string ;
                    ] ;
                    core:hasProperty [ a core:PrimitiveProperty :
                         core:basedOnProperty bim:productId ;
                         core:hasDatatype xsd:string ;
     ]]]] ;
     core:hasReturnType xsd:string .
```

Listing 8 Definition of *PlaceOrderService* class within the BIM.

```
retailerDeviceX a bim:RetailerDevice ;
     rdfs:label "device of retailer X@en"^^xsd:string ;
     core:locatedAt [ a WGS84Location ;
          geo:long "8.414127"^^xsd:double ;
          geo:lat "49.020464"^^xsd:double ;
     ] :
     core:providesService [ a owl:NamedIndividual ;
          a bim:GetInformationService ] ;
     core:providesService [ a owl:NamedIndividual ;
          a bim:PlaceOrderService ] .
```

Listing 9 Example registration of a retailer device.

5.6 Related Information Models and Ontologies

A vast amount of information models and ontologies related to the IoT domain exist. However, most of them are quite specific, e.g., focused on a specific subdomain, certain use cases or even single applications, and/or are of low quality. Nevertheless, some of them are of high quality and well established in the IoT domain. In the following, we present the related models that influenced the symbIoTe information model the most.

5.6.1 SSNO & SOSA

The Semantic Sensor Network (SSN) Ontology [12] can easily be called the most important ontology in the sensing domain. It was created by the W3C Semantic Sensor Network Incubator Group and motivated by OGC's Sensor Model Language (SensorML) [18] and Observations and Measurements (O&M) [19]. It is a high-level ontology and built around the *Stimulus Sensor Observation* (SSO) pattern [16]. Figure 24 shows the ontology structure of the SSN ontology.

symbIoTe is strongly inspired by the SSN ontology in terms, especially by its skeleton. It boroughs multiple concepts, e.g. *Device*, *Sensor*, *Property*, *FeatureOfInterest*, *Observation* and additionally many relations between them. However, the SSN ontology did not support important things we need in symbIoTe, e.g. querying what feature of interest a sensor observes before it has any observations.

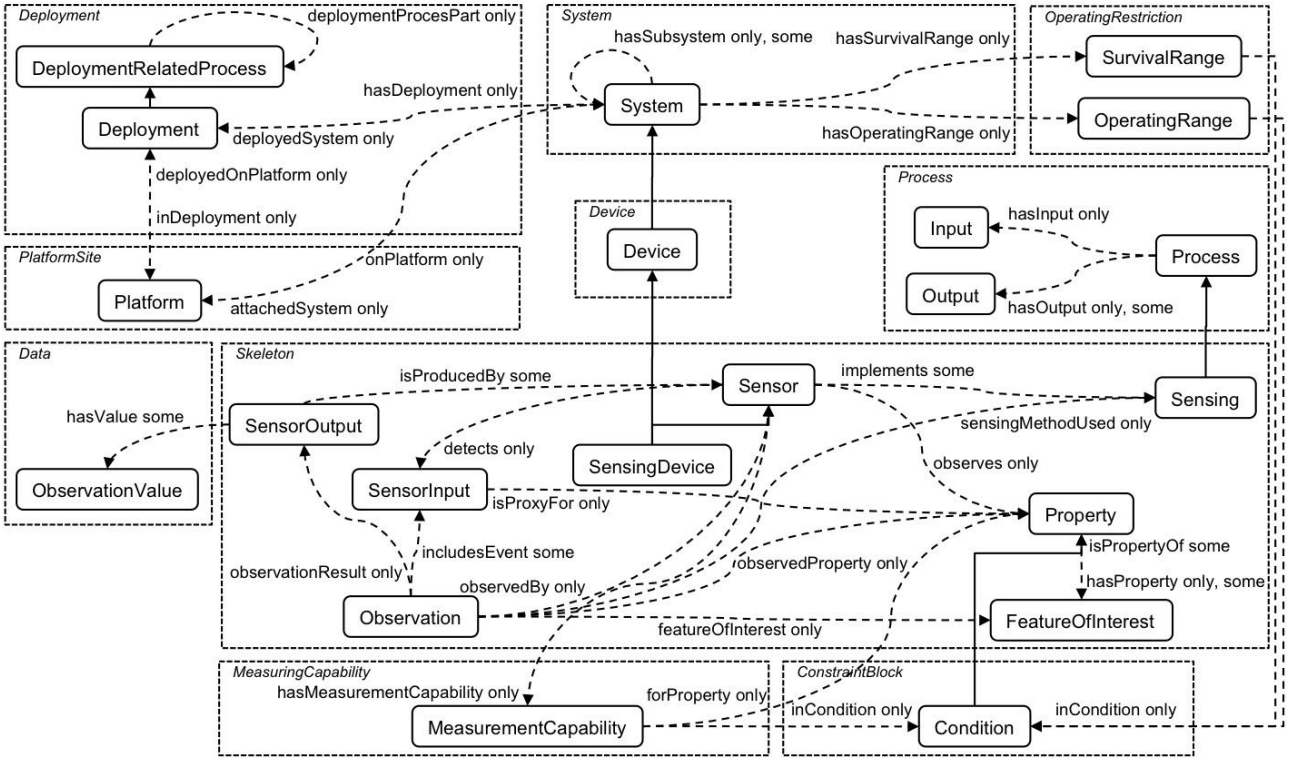


Figure 24 Semantic Sensor Network (SSN) Ontology.

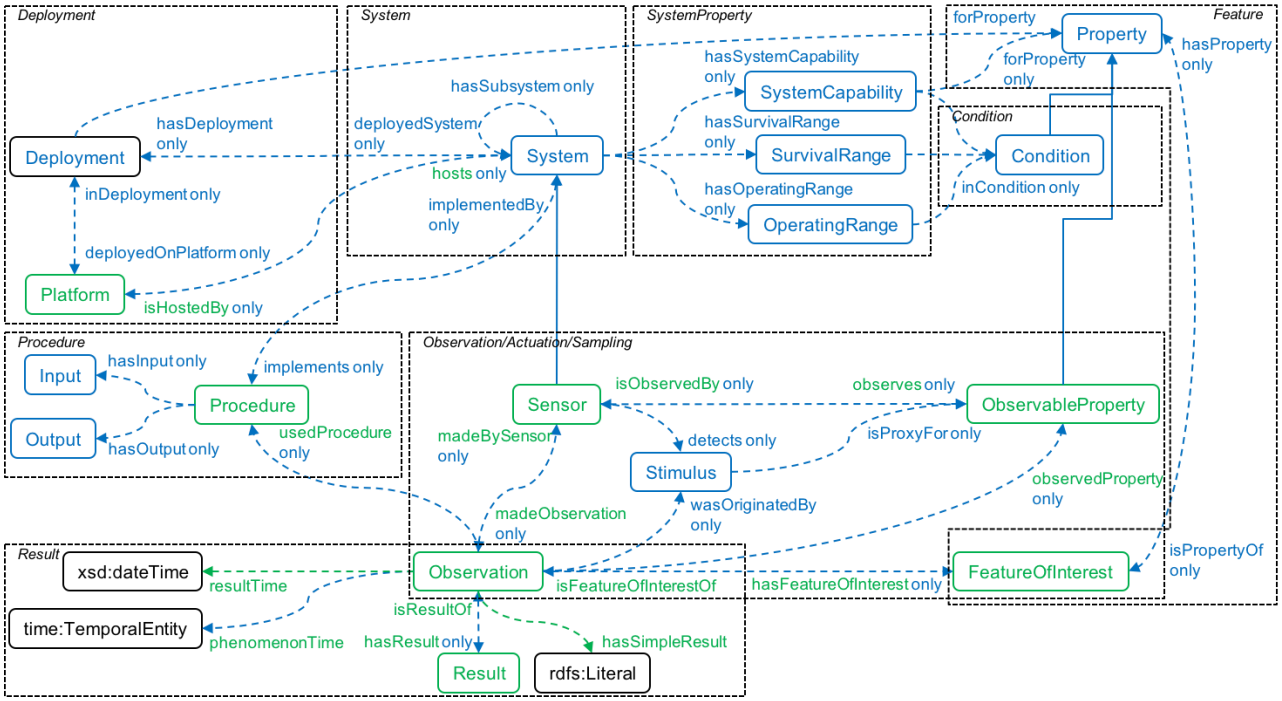


Figure 25 Updated SSN (blue) & SOSA (green) Ontology.

These problems were then partially addressed by the updated version of the SSN ontology that goes hand in hand with the introduction of the Sensor, Observation, Sample, and Actuator (SOSA) ontology. Figure 25 shows the updated SSN ontology (depicted in blue) together with the SOSA ontology (depicted in green). Unfortunately, this updated version together with the SOSA ontology were released too late to be integrated into the symbIoTe Core Information Model. However, multiple things that were created within the symbIoTe CIM were also introduced in the same or a very similar way in the SOSA ontology, which indicates that the CIM is designed well and does address relevant issues.

symbIoTe Information Models are not aligned with the SSN or the SOSA ontology so far. Adding an alignment to SSN would be possible (e.g. by adding *owl:equivalentClass* tags) if needed as the sensing part is strongly motived by the concept of the SSN ontology. Some concepts similar in the CIM and SOSA could be mapped this way.

5.6.2 SensorThings API Information Model

The OGC SensorThings API [20] standard provides an open and unified framework to interconnect IoT devices, data and applications over the Web. Up to now, only the first part of the standard is published which covers the sensing domain. The second part, covering actuation domain, is not yet published and is still under discussion. Sensor Things API has influenced symbIoTe in two ways. First, its information model, depicted in Figure 26, has been used as inspiration and basis for discussion regarding the relation between Sensor and Observation, which is done differently in SensorThings API than in SSN. Second, the REST- and JSON-based API that is using OData has been adopted in symbIoTe.

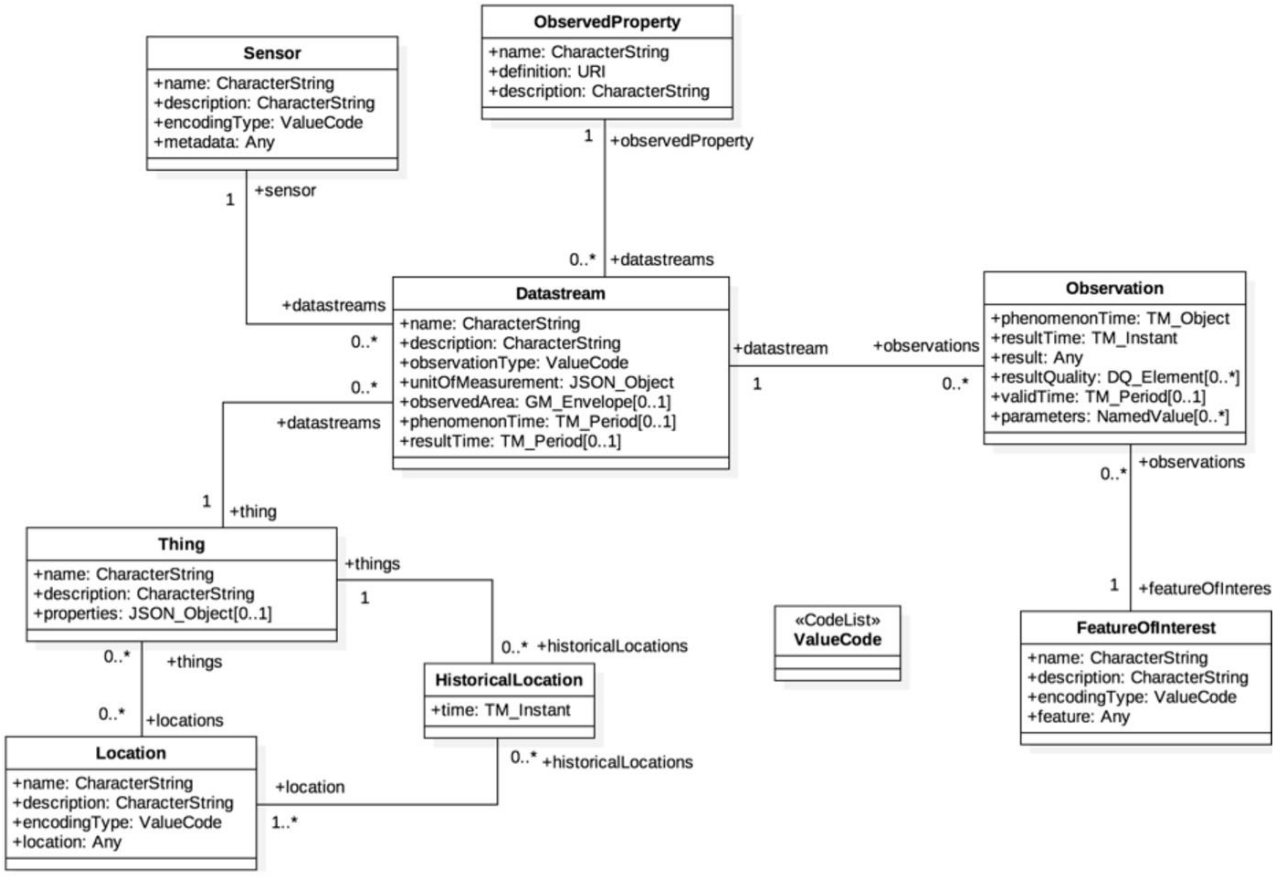


Figure 26 SensorThings API Information Model[24].

5.6.3 Schema.org

Schema.org is an initiative trying to “standardize” ontologies in multiple domains to describe structured/semantic data on the web. It was launched by Bing, Google and Yahoo! in 2011, but the work is community driven and most communication takes place on the W3C public vocabularies mailing list.

It is mentioned here, because it may have a big influence on symbIoTe or at least on the semantic interoperability part of it. As explained previously in the document, the symbIoTe consortium takes a position that using a common, shared vocabulary is always the best option. However, we also agree that this will not be possible in all cases. schema.org now tries exactly to do this and we are looking forward to see how this initiative evolves.

Furthermore, there is an extension of schema.org called iot.schema.org aiming at defining common, shared vocabularies for the IoT domain. Unfortunately, again, this project has started too late to be recognized by symbIoTe and it is progressing very slowly.

For now, schema.org is only used within some parts of the BIM where it is suitable, e.g., the Smart Yachting part of the BIM. We considered using it also in the smart stadium part of the BIM, but the schema.org classes turned out to be too complex for that special kind

[24] http://docs.opengeospatial.org/is/15-078r6/15-078r6.html

of usage scenario. However, we designed our classes a simplified version of the schema.org classes, which makes adding an alignment later on easy and straightforward.

The BIG IoT (Bridging the Interoperability Gap of the Internet of Things)[25] H2020 project is developing an extension to schema.org for the mobility domain called mobility.schema.org [21]. Once this vocabulary is finished, we will check to align the smart mobility part of the symbIoTe BIM to it.

5.6.4 oneM2M Base Ontology

oneM2M is a global standardization body for the machine-to-machine (M2M) communications and IoT which has been established in 2012 following an initiative from the European Telecommunications Standards Institute (ETSI). It is formed as an alliance of standardization organizations with 200 member companies from across the world working together “to develop a single horizontal platform for the exchange and sharing of data among IoT devices and applications” [22]. oneM2M focuses on standardization of platform interfaces and aims to provide an interworking framework across different sectors. Within this interworking framework, semantic interoperability plays a central role. Just like symbIoTe, oneM2M has recognized that agreeing on one (or very few) complete ontologies (“complete” meaning covering the whole IoT domain) is not a suitable approach. Therefore, oneM2M pursues a quite similar approach as symbIoTe by providing a core ontology, called oneM2M Base Ontology, and allowing custom extensions of it. However, there are two major differences regarding the approach to semantic interoperability between oneM2M and symbIoTe.

The first one is the information model shared between platforms. Figure 27 shows the one used in oneM2M called oneM2M Base Ontology. Some classes like *Device* and *Service* appear in the oneM2M Base Ontology as well as the CIM. Furthermore, they both follow the design principle to create a rather abstract high-level ontology that can be extended. Nonetheless, they are quite different: the major difference is that in oneM2M actuation and sensing capabilities are modelled as functions, while in the CIM we use separate classes for them. These are contrary design decisions, but it was made because symbIoTe needs a deeper understanding of things to provide additional services on top of them.

The second major difference is related to semantic mapping. Although oneM2M states to support semantic mapping, it is limiting support only to sub-classing the concepts of the oneM2M Base Ontology and expressing equality by using the concepts provided by RDFS and OWL (rdfs:subProperty, owl:equivalentProperty, rdfs:subClassOf and owl:equivalentClass). Our vision regarding semantic mapping goes beyond that allowing also complex mappings to be defined.

As the first release of the oneM2M Base ontology was in December 2016 [13] it did not have a strong influence on the symbIoTe information models. In fact, it can be seen as an acknowledgement of symbIoTe’s approach to semantic interoperability that oneM2M also supports semantic mapping and highlights its importance.

[25] http://big-iot.eu/

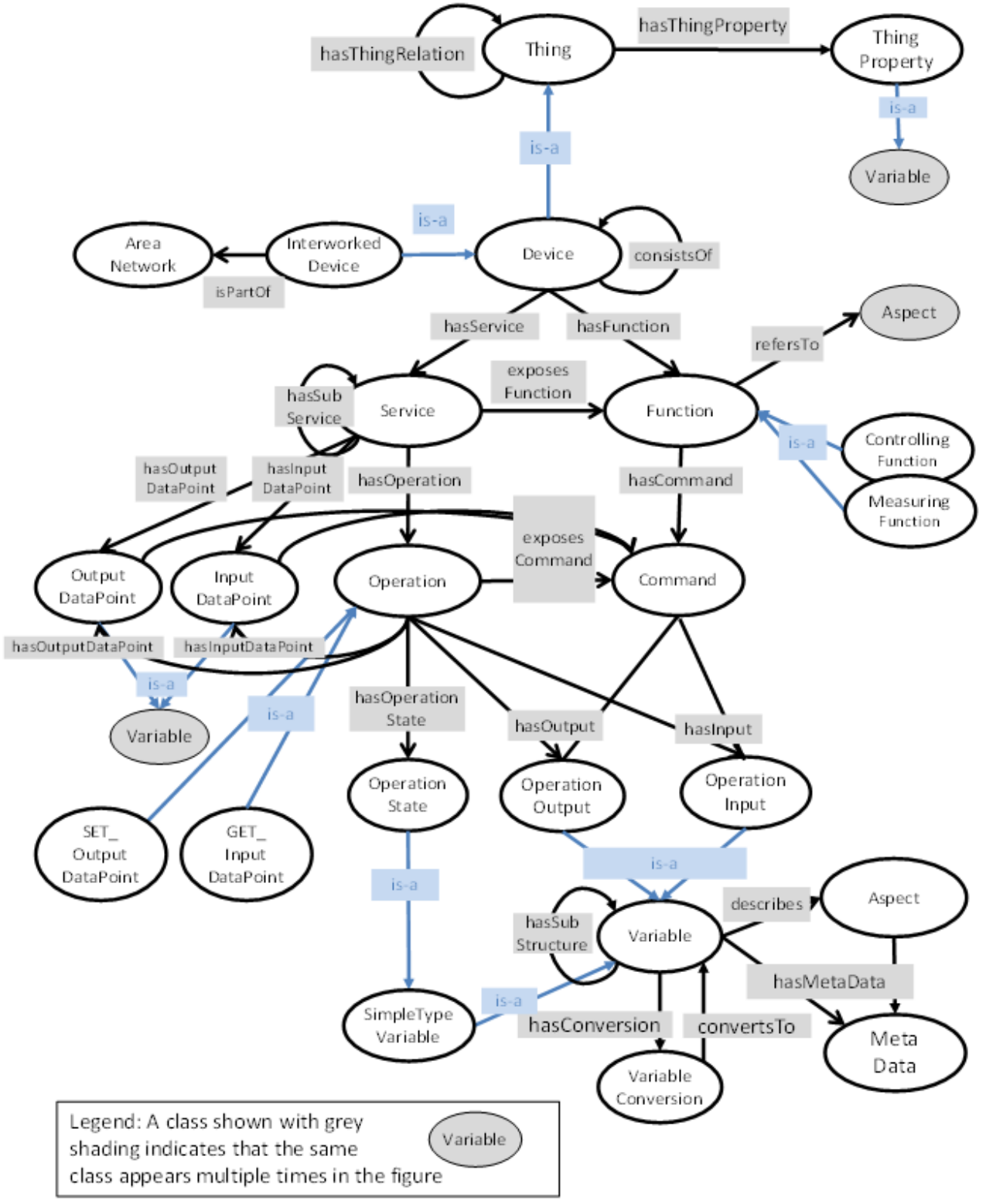


Figure 27 The oneM2M Base Ontology v3.2.0 [23].

5.7 symbIoTe Architecture from the Semantic Interoperability Perspective

In this section, we provide details on how the chosen approach to semantic interoperability influences the symbIoTe architecture, as defined in D1.4 “Final Report on System Requirements and Architecture” [24]. We start from a modified version of the component diagram for symbIoTe Level 1 compliance as presented in D1.4, which highlights all components that are directly influenced by semantics. We then describe how the functionality of these components is adapted to support semantics. Additionally, we provide sequence diagrams showing the internal workflow between components for basic operations like resource registration. Details on how semantic mapping and SPARQL query re-writing influence the architecture can be found in the next subsection. This document provides only a coarse overview of how semantic influence the architecture and implementation. Details on the architecture can be found in D1.4 “Final Report on System Requirements and Architecture” [24] and details on implementation in D2.5 “Final symbIoTe Virtual IoT Environment Implementation” [25].

5.7.1 Component Descriptions

Figure 28 shows the symbIoTe component diagram for Level 1 compliance. All components that are directly affected by semantics are highlighted in green. The most affected parts of the architecture are those that enable interaction between the symbIoTe Core Services and symbIoTe-enabled IoT platform. Besides the components highlighted in green also every other component in the Core is influence by semantics, as they need to be aware of and use the defined information models. However, in this document, we focus on the components that are more affected by semantics than just by adopting the models.

The following tables describe the functionality specific to semantic interoperability for the most influenced components. The component descriptions must be understood as an update/extension to the complete component definitions in D1.4.

Table 4 Changes of the Administration component introduced by semantic interoperability.

Component	Administration
symbIoTe Domain	APP
Description	This component enables registration of a PIM through its interface. The PIM should be provided in the RDF format, which can be verified to check if the PIM is aligned with the CIM. The models are stored in the Registry and in the Semantic Manager for further validation. This component will also provide an interface to register mappings between two PIMs. The used language to define the mapping is EDOAL. The mappings are stored in the Registry and Search Engine and later on used for SPARQL query re-writing.
Provided functionalities	• Provides an interface for registration of a PIM. • Provides an interface for registration of a mapping.
Relation to other components	Registry: stores PIMs and mappings Semantic Manager: stores PIMs for validation Search Engine: stores mappings

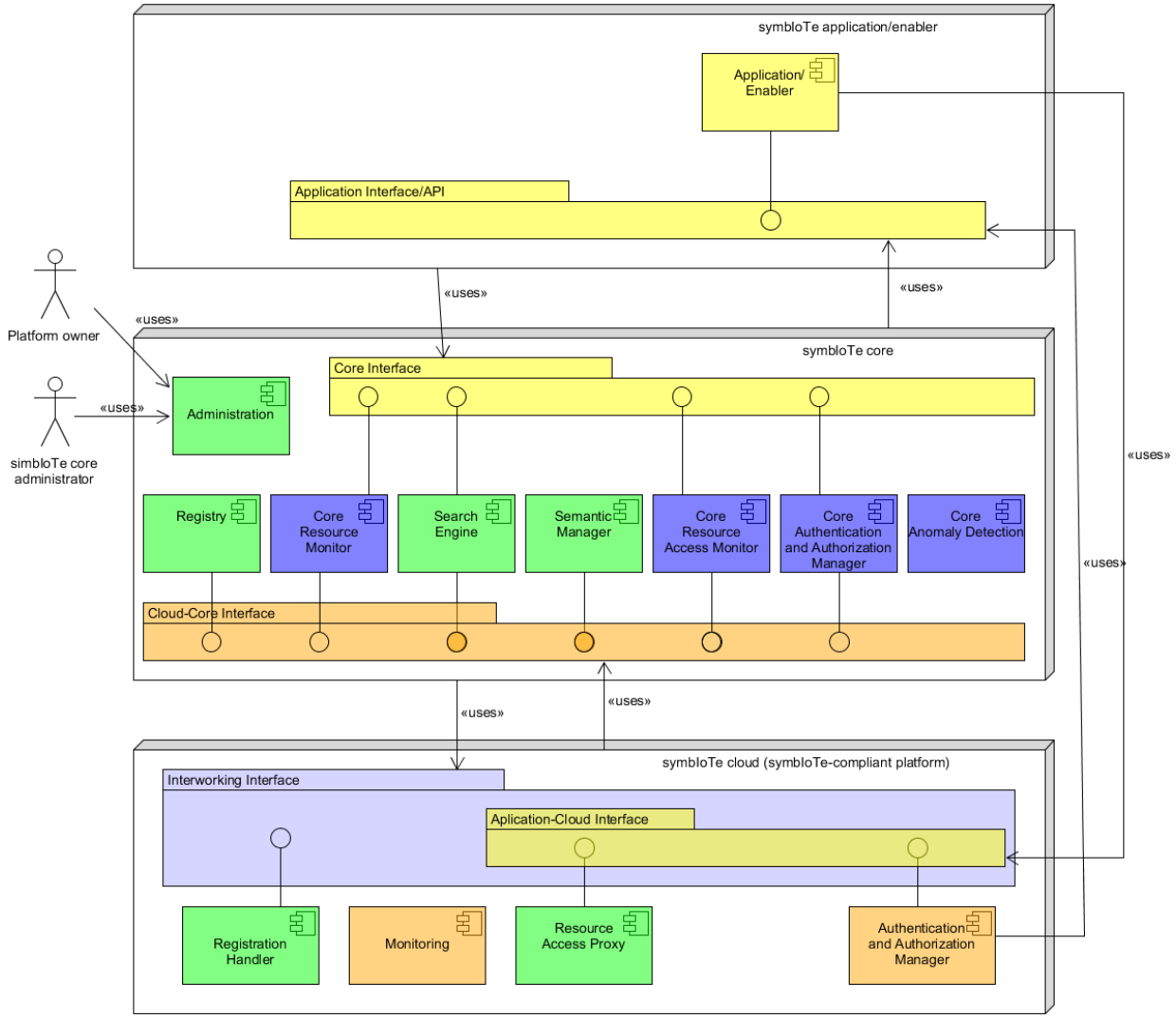


Figure 28 symbIoTe component diagram for Level 1 compliance (from D1.4) with changes regarding semantic interoperability highlighted in green.

Table 5 Changes of the Registry component introduced by semantic interoperability.

Component	Registry
symbIoTe Domain	APP
Description	The registry must allow resources to be registered according to the CIM, BIM or any valid PIM. Furthermore, the Registry stores the definition of PIMs. All interfaces should communicate using the Semantic Web technologies. In case of data manipulation (insertion, deletion, update), resources and resource metadata should be communicated in the RDF format (e.g., as payload of REST message).
Provided functionalities	• Handles requests for resource registration according to CIM, BIM or any valid PIM. • Stores definitions of PIMs.
Relation to other components	Search Engine: uses PIM definition for SPARQL query re-writing

Table 6 Changes of the Search Engine component introduced by semantic interoperability.

Component	Search Engine
symbIoTe Domain	APP
Description	This component allows searching for registered resources across platforms registered to symbIoTe in a unified way. A query must be formulated against an information model previously registered with symbIoTe, which can be either the CIM, any PIM or the BIM, i.e., a special case of PIM that is used by multiple platforms. The goal is to return cross-platform results that satisfy the conditions of the query. When the query is formulated against the CIM than it can be answered by straightforward execution. Otherwise, if the query is formulated against a PIM (or, as a special case, the BIM), the query is translated based on the mappings between the used PIM and other PIMs with the help of the Semantic Manager to provide results across multiple platforms. The component's primary interface accepts SPARQL queries, but to simplify its usage, it also offers multiple pre-defined parameterized search queries (such as: search for a property, location, owner, etc.), which will be subsequently transformed to the SPARQL query using pre-prepared templates before executing it in Search component.
Provided functionalities	• Search for resources with support for SPARQL query re-writing.
Relation to other components	Semantic Manager: provides query re-writing functionality

Table 7 Changes of the Semantic Manager component introduced by semantic interoperability.

Component	Semantic Manager (SM)
symbIoTe Domain	CLD
Description	This component stores the CIM, MIM, BIM and all PIMs. Upon registration of a new PIM, it is validated and checked to be compliant with the CIM. When registering resources to symbIoTe Core by using JSON description, the component translates resource description to RDF. It also validates the resources described using RDF to ensure they conform to the information model they claims to be using (PIM/BIM/CIM).
Provided functionalities	• stores MIM, CIM, BIM, PIMs • validates PIMs • validates if the instances of data (resource descriptions) conform to PIM/BIM/CIM they claim to be using • translates resource descriptions from JSON to RDF format • translates resource descriptions from RDF to JSON format • provides SPARQL re-writing functionality.

Relation to other components	Registry: sends the resource description obtained during registration process for validation. Sends new PIM models being registered by the platforms. Search Engine: Provides SPARQL re-writing functionality needed by Search.

Table 8 Description of the Registration Handler component regarding the symbIoTe Information Model.

Component	**Registration Handler (RH)**
symbIoTe Domain	CLD
Description	Registration using the PIM relies on the Semantic Web technologies. PIM and resource instances description should be provided in the RDF format for the general version of the API of this component but for ease of usability additional interfaces hiding the Semantic Web technologies can be added.
Provided functionalities	• Registers resources to the symbIoTe core, virtual and physical, using the PIM • Updates resource status and unregistered resources
Relation to other components	Registry (within symbIoTe Core Services): stores data about resource according to the PIM, assigns unique symbIoTe IDs and maintains information about current resource status.

Table 9 Description of the Resource Access Proxy component regarding the symbIoTe Information Model.

Component	**Resource Access Proxy (RAP)**
symbIoTe Domain	CLD
Description	This component enables symbIoTe-compliant access to resources within an IoT platform or (enabler acting as a platform). The data generated by IoT Services must be returned in accordance with the used PIM. This means that the returned JSON object must fit the PIM, i.e., it must contain additional properties of an object that are defined within the PIM.
Provided functionalities	• Ensures formatting of data generated by resources in accordance with the PIM
Relation to other components	Application/Enabler: provides requested data

5.7.2 Resource Registration

In this section, we describe in detail how resource registration is affected by introduction of the semantic interoperability approach. Figure 29 shows the sequence diagram for resource registration and is taken from Deliverable D1.4. As the dashed lines represent optional messages (Message 2 to 6), they are neglected in this document.

The workflow is as follows. First, The Resource Handler (RH) notifies the RAP and the Monitoring component updates the changes (Message 1). Then the RH sends a register/unregister/modify request to the Registry (Message 7). The payload of this request is a JSON object according to the available resource types in the CIM. If a PIM is used on platform side, it additionally contains a RDF description of the affected resource. This message is forwarded to the Semantic Manager (SM) (Message 8). In case there is not yet an RDF description, (i.e. when the platform does not use a PIM) the SM converts the JSON object into RDF. If the RDF description already exists, it is checked by the SM to comply with the used PIM. The RDF description is also stored in the Search Engine (Message 8). Finally, the Core Resource Monitor (CRM) and Core Resource Access Monitor (CRAM) are notified about the changes (Message 9) and the result is provided to the RH (Message 10).

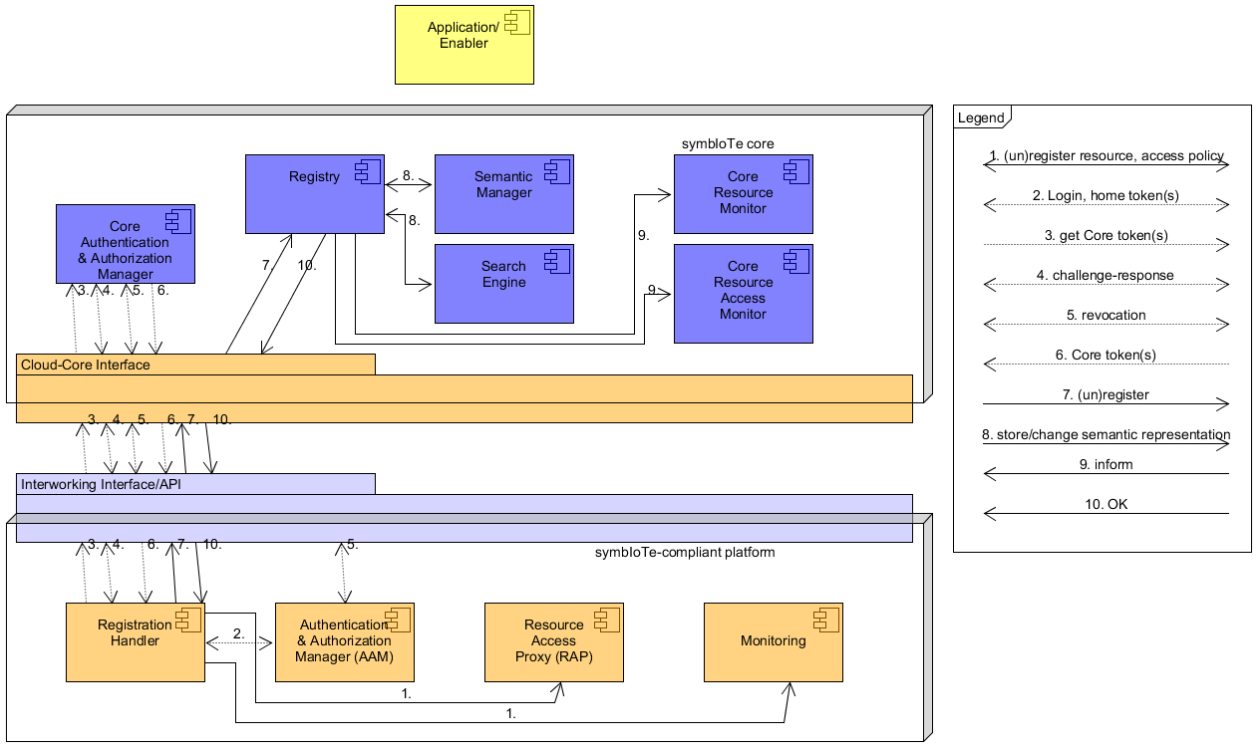


Figure 29 Sequence diagram showing resource registration, unregistration and modification (from D1.4).

5.7.3 Resource Search using Semantic Mapping and SPARQL Query Re-Writing

Ontologies are a way to formally describe the concepts and relations within a domain. However, even if two ontologies cover the same domain, they can describe the domain quite differently, e.g. use a taxonomy with another scope or granularity, use the same terminology but in a different language or even use a different terminology. Such differences between ontologies are called ontology mismatches.

Semantic mapping refers to the idea to resolve ontology mismatches by defining statements and rules how data expressed using one ontology can be translated into the terms of another ontology. As depicted in Figure 30, such a statement or rule is called a *correspondence pattern* and consists of a source ontology, a target ontology and some correspondence/transformation information. All correspondence patterns having the same

source and destination ontology together form an *alignment* between the two ontologies. Such an alignment contains all correspondence patterns necessary to translate instances of the source ontology into instances of the target ontology. Some mismatches are so profound that this translation is not possible without loss of information. In fact, this is quite common as only the data modelled in both ontologies, i.e., a semantic intersecting set of the two ontologies, can be safely translated between them.

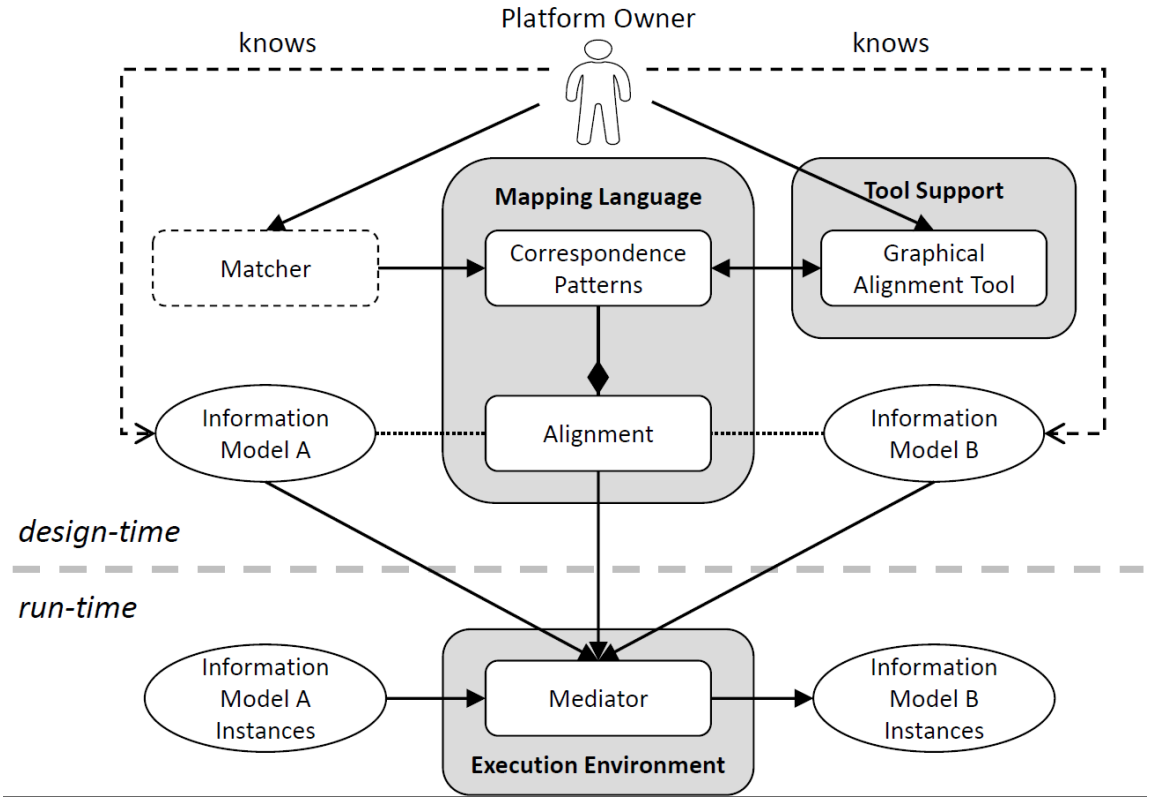


Figure 30 Schematic representation of an example usage of semantic mapping for semantic interoperability.

As depicted in Figure 30, semantic mapping is not only about defining mapping between ontologies but also about using these mappings at runtime to mediate between them. This functionality is provided as an execution framework or a mediator. In symbIoTe, this will be implemented using SPARQL query re-writing techniques [26] [27] [28]. How this works in detail is shown in Figure 31, which is an enhanced version of the sequence diagram describing the symbIoTe Core search functionality taken from D1.4 “Final Report on System Requirements and Architecture”.

The part most relevant to semantic mapping in Figure 31 is highlighted in green. With message 5, the Search Engine receives a SPARQL query which has to be formulated against an information model (in this context, information model and ontology are used synonymously) previously registered with symbIoTe. Optionally, the search can be invoked using a flag to indicate the query re-writing should be used. If done so, the green part of the diagram is executed, otherwise the grey part (normal query execution).

We are aware that not all people are familiar with semantic technologies like SPARQL and that enforcing them to use semantic technologies can be quite an entry barrier. As SPARQL is only a direct interface to the Search Engine, this does not mean it must be the only one. To facilitate access to the Search Engine we are planning to also provide a

simpler interface with a limited subset of functionality, e.g., a REST-based interface supporting multiple pre-defined parameterized SPARQL queries.

If a search requests specifies to use query re-writing, the workflow is as follows. First, the Search Engine looks up all existing mappings $M_{1..n}$ (Message 11a) related to the used information model IM_{Query} in the query Q. The other information models related to IM_{Query} via the mappings $M_{1..n}$ are referred to as $IM_{1..n}$. Then, the Search Engine forwards the query Q and the mappings $M_{1..n}$ to the Semantic Manager with the request to re-write the query based on the mappings (Message 11b). The Semantic Manager returns the re-written queries $Q_{1..n}$ to the Search Engine (Message 11c) which then executes them (Message 11d) receiving a list of query results. These are forwarded to the Semantic Manager for result re-writing (Message 11e) and the re-written results $QR_{1..n}$ are returned back to the Search Engine (Message 11f). As final step, the Search Engine combines the re-written results $QR_{1..n}$. After that, the workflow is the same as for a normal query without query re-writing.

Following this approach, symbIoTe is able to answer queries across all platforms using a different information model than the one used in the query by applying SPARQL query re-writing. Multiple different types of ontology mismatches exist, and only a subset will be supported for SPARQL query re-writing by symbIoTe. This means that PIMs, which extend the CIM in such ways that they have serious types of mismatches will probably not be able to interoperate in practice.

5.7.4 Resource Access

Resource access is influenced by semantic interoperability in two ways. First, the REST-based interface for resource access, called Interworking Interface/API, is based on the concepts that are defined within the CIM, e.g. Sensor, Actuator and Service. These are used to define the URLs/URL patterns that the RAP exposes. Second, the model used on platform side, BIM or PIM, determines the structure of the JSON object returned as response to a call to these URLs. As any PIM must be an extension of the CIM, anyone can at least understand the properties of the JSON object that are defined in the CIM. If the PIM defines additional properties, these can either be ignored if understanding is not achievable or be translated to fit the own PIM using a symbIoTe library.

5.8 Vision

As stated before in this document, the second approach to semantic interoperability, interoperability through mapping, has not been foreseen (in such a detail) in the DoA. Nevertheless, the symbIoTe consortium decided that this approach is of such importance that it should be addressed. As it is quite complex and touches multiple areas of current research, symbIoTe focuses on analysing it and providing a proof-of-concept solution limited in functionality and ease-of-use. Additionally, we provide this section, which puts the proof-of-concept solution in a larger context. This section will therefore present an overall vision how this approach could be realized in detail covering the needed technologies that are currently under research or not yet developed. Furthermore, it explains which components and technologies are needed to make the proof-of-concept solution more convenient to use, especially for non-experts, and lists pitfalls that are already recognized by the consortium.

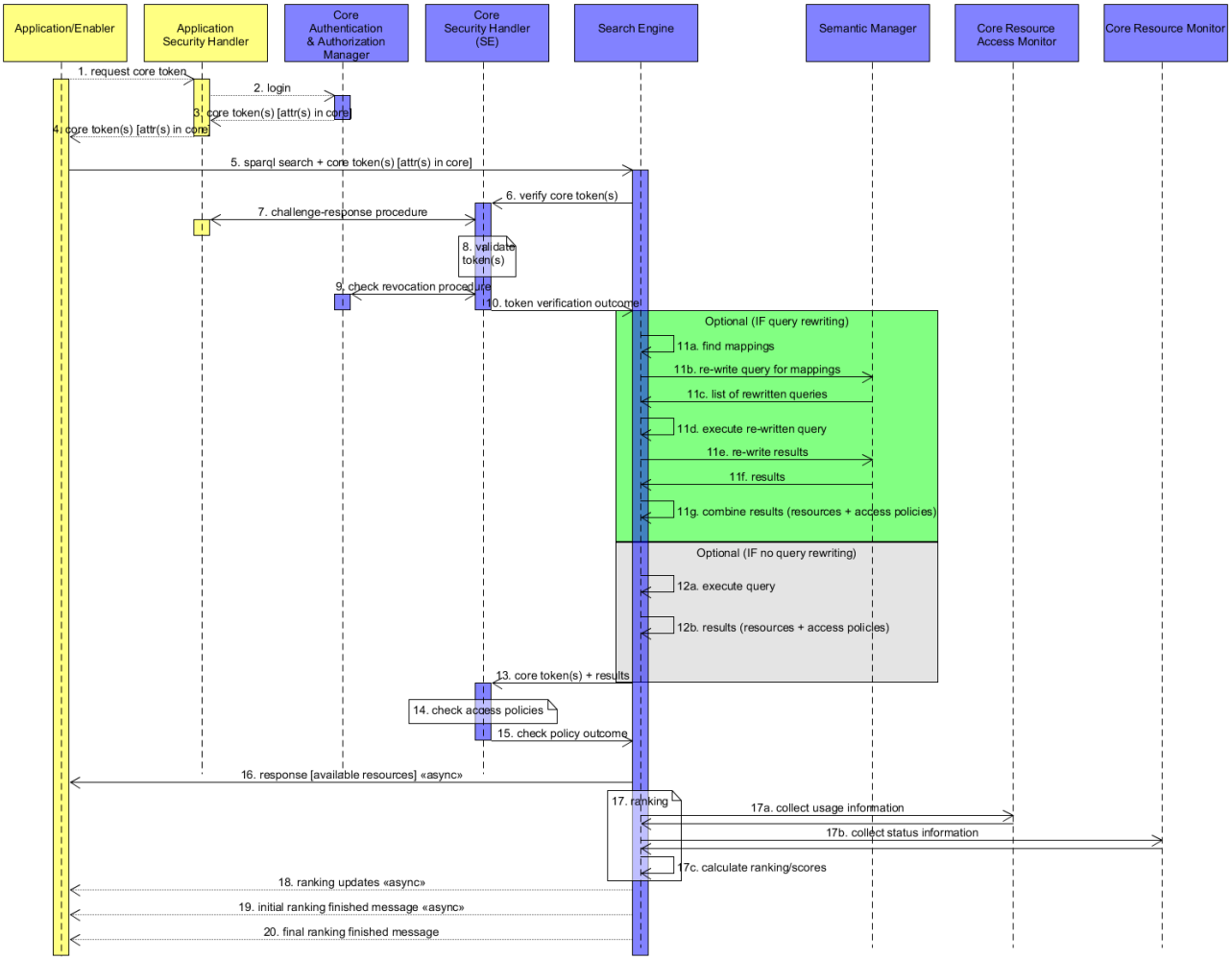


Figure 31 Enhanced sequence diagram describing the search functionality with respect to SPARQL query re-writing.

Figure 30 depicts a schematic representation on how semantic mapping can be used to achieve semantic interoperability and which kind of software and tools are involved. In the centre, we see an alignment as an aggregation of multiple correspondence patterns. To formulate, express and exchange such correspondence patterns, a mapping language is needed. The main criteria for choosing a mapping language is its expressivity identifying what kind of ontology mismatches can be resolved using this language. This first requires a classification of ontology mismatches. This area has been subject to research for around 20 years [29] [30] [31] [32] [33], but up to now there has not been a generally accepted classification. Choosing a mapping language is an even more complex thing to do. There a multiple existing ones but most of them were not designed to be used as a mapping language. For example, the well-known Semantic Web standards from the W3C like SPARQL Construct[26], SPARQL Inferencing Notation (SPIN) [27] or Semantic Web Rule Language (SWRL) [34] could be used in such a way. As they were not created for that purpose, they will most likely turn out to have a limited expressivity in the context of

[26] https://www.w3.org/TR/rdf-sparql-query/#construct
[27] https://www.w3.org/Submission/spin-overview/

semantic mappings. Another candidate would be OWL with its predicates *owl:sameAs*, *owl:equivalentProperty* and *owl:equivalentClass*, but obviously the expressivity is quite limited as the mentioned predicates can only express equality. Furthermore, there are vocabularies and languages specifically designed for defining mapping between ontologies, e.g. the Alignment Format [17] and its extension EDOAL (Expressive and Declarative Ontology Alignment Language) [35], C-OWL [36] or MAFRA (Ontology MApping FRAmework) [37]. symbIoTe proposes to use EDOAL for the proof-of-concept implementation as it seems the most mature and has the highest expressivity. This proof-of-concept implementation will be used be the EduCampus use case.

In the upper centre of Figure 30, we see the user, who in our case is a platform owner of an IoT platform. He has knowledge about the information model used in his/her platform (Information Model A) as well as about the one used in another platform (Information Model B). Based on this knowledge he wants to formulate a mapping/alignment between the two information models using a mapping language. As this is a complex task, he should be supported by tools. One option thereby would be to add a *Matcher* to the workflow. A matcher is a tool that automatically discovers correspondences between two given ontologies, e.g., by applying different similarity measures [38] [11] [39] [40]. This problem is known as *ontology matching* and is a broad research area on its own. An international workshop (called International Workshop on Ontology Matching[28]) is addressing issues in this field every year. Another option to support a user in defining semantic mappings is by providing a visual editor. A wide body of research on visual aids for semantic mapping covers surveys [41] and example editors [42] [43] [44], but up to now, only very few types of ontology mismatches, often even only equality between classes can be expressed since the complexity level of ontologies can increase very quickly. However, providing a suitable graphical user interface that is intuitive gets noticeably harder with rising expressivity of the mapping language.

An additional challenge is that the whole tool chain has to work with the same mapping language. This is a big problem with currently existing prototypes and software artefacts as they are often using some proprietary mapping language because there is no well-established standard so far.

Once an alignment between two information models is defined at design-time, it can be used at run-time by some kind of *mediator* to translate information expressed using information model A to information model B. In symbIoTe, we realize this mediator using SPARQL query re-writing.

Another challenge is the question “How does one even find another platform using a different PIM that is worth the effort to define a mapping to it in the first place”? For symbIoTe, this is considered to be done either offline, e.g., by knowing the owner of a different platform or through search via the SPARQL endpoint where one can browse the models of other platform and thereby can identify platforms that may offer resources that are of interest. However, for this to work in a more user-friendly way, especially in cases of dynamical interaction of platforms, a more sophisticated search would be desirable. Such search feature could be quite primitive, e.g., a simple full-text search on the terms defined in the PIMs, or more sophisticated using concepts like phonetic search, natural language processing or translations to enable finding of relevant terms even if they do not syntactically match a search term or are expressed in another language.

[28] http://om2017.ontologymatching.org

To conclude this short discussion, we also see that this approach paves the way for further research topics. One possible area of research could be the theoretical evolution of the requirements for and the impact of enabling the chaining of mappings. This means that if there is a mapping between platform A and platform B and one between platform B and platform C, then we could theoretically chain these mappings and gain a mapping between platform A and platform C. With this technique, only a few mappings would be needed to make many platforms interoperable. Another very exciting question is, whether such a network of models and mappings between them can be seen as creating some sort of a unified common vocabulary in a hidden manner. Eventually, such a system could even become a transition from custom, separated models to a unified model and could replace or at least boost classical standardization work.

6 Conclusions and Next Steps

This deliverable presents the work done in the symbIoTe project regarding semantics and especially semantic interoperability. It introduces the topics of semantics, semantic technologies and semantic mapping in Section 3.

In Section 4, the problem of semantic interoperability between multiple heterogeneous IoT platforms is described together with a formal analysis of the solution domain to this problem. Additionally, five different possible approaches to achieve semantic interoperability are presented, ranging from a “monolithic” approach where all platforms use a single completely agreed upon information model, to a “distributed” approach where each platform defines their own information model and exchange information via mappings between those information models. The section closes with a comparison of the five approaches identifying their advantages and disadvantages.

As semantics is essential for IoT platform interoperability (as explained in Section 3.1), we decided to build upon well-established semantic technologies (e.g. RDF) to describe information models. In Section 5, we present symbIoTe’s approach to semantic interoperability, which is based on the *Core Information Model with Extensions* approach introduced in Section 4.2.3. In Section 5.2 to Section 5.5, we provide detailed information on the created information models in, namely the Core Information Model (Section 5.2), Meta Information Model (5.3), Platform-Specific Information Models (Section 5.4) and the Best-Practice Information Model (Section 5.5). In Section 5.6, we explain how these models related to other already existing ontologies and models like the SSN and SOSA ontology, the SensorThingsAPI Information Model, Schema.org and the oneM2M Base Ontology. In the following section, we present the influence of the semantic interoperability approach to the symbIoTe architecture and implementation.

As explained in the previous section, our approach to semantic interoperability using semantic mapping is very powerful and has a high potential to influence how semantic interoperability is approached in the future. However, there are several hard research questions to be answered before this technology can be used to its full extent. As next steps, we focus on implementation of the parts of the symbIoTe approach to semantic interoperability that go beyond what we promised in the DOA, namely the semantic mapping.

7 References

[1] Network Centric Operations Industry Consortium, “NCOIC, "SCOPE",” 2008.

[2] M. Jacoby, A. Antonic, K. Kreiner, R. Lapacz and J. Pielorz, “Semantic Interoperability as Key to IoT Platform Federation,” *Interoperability and Open-Source Solutions for the Internet of Things,* Forthcoming 2017.

[3] R. Herzog, M. Jacoby and I. Podnar Zarko, “Semantic interoperability in IoT-based automation infrastructures,” *at-Automatisierungstechnik,* vol. 64, no. 9, pp. 742-749, 2016.

[4] “Merriam-Webster,” 06 04 2016. [Online]. Available: http://www.merriam-webster.com/dictionary/semantics.

[5] T. R. Gruber, “A translation approach to portable ontology specifications.,” in *Knowledge acquisition 5.2*, 1933, pp. 199-200.

[6] A. Tolk and J. A. Muguira, “The levels of conceptual interoperability model,” in *Proceedings of the 2003 Fall Simulation Interoperability Workshop*, Citeseer, 2003, pp. 1-11.

[7] “Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/ Conceptual_interoperability#Levels_of_conceptual_interoperability. [Accessed 06 04 2016].

[8] T. Berners-Lee and L. Ora, “The semantic web,” *Scientific american,* pp. 28-37, 2001.

[9] W3C, “Vocabularies,” [Online]. Available: https://www.w3.org/standards/semanticweb/ontology. [Accessed 06 10 2016].

[10] H. Wache, T. Voegele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann and S. Hübner, “Ontology-based integration of information-a survey of existing approaches,” in *IJCAI-01 workshop: ontologies and information sharing*, 2001.

[11] N. Choi, I.-Y. Song and H. Han, “A survey on ontology mapping,” *ACM Sigmod Record,* 2006.

[12] M. Compton, P. Barnaghi, L. Bermudez, R. García-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson and A. Herzog, “The SSN ontology of the W3C semantic sensor network incubator group,” *Web Semantics: Science, Services and Agents on the World Wide Web,* 2012.

[13] oneM2M Partners Type 1, “oneM2M Base Ontology,” 2016.

[14] C. Bizer, T. Heath and T. Berners-Lee, “Linked data - The story so far,” *Semantic services, interoperability and web applications: emerging concepts,* pp. 205-227, 2009.

[15] S. Cox, “SOSA ontology,” 2016. [Online]. Available: https://www.w3.org/2015/spatial/wiki/SOSA_Ontology. [Accessed 30 06 2017].

[16] K. Janowicz and M. Compton, “The stimulus-sensor-observation ontology design pattern and its integration into the semantic sensor network ontology,” in *Proceedings of the 3rd International Conference on Semantic Sensor Networks-Volume 668*, 2010, pp. 64-78.

[17] J. Euzenat, “An API for ontology alignment,” in *International Semantic Web Conference*, Springer, 2004, pp. 698-712.

[18] O. SysML, *OMG Systems Modeling Language, Version 1.2,* 2013.

[19] S. Cox, *Observations and Measurements (O&M),* Open Geospatial Consortium, 2011.

[20] S. Liang, C.-Y. Huang and K. Tania, *OGC SensorThings API-Part 1: Sensing,* 2016.

[21] B. I. Consortium, “Deliverable 3.2.a Semantic Interoperability Design for Smart Object Platforms and Services,” 2016.

[22] oneM2M, “The interoperability enabler for the entire M2M and IoT ecosystem,” oneM2M whitepaper, 2015.

[23] oneM2M, “Technical Specification: Base Ontology,” TS-0012-V2.3.0, 2017.

[24] symbIoTe consortium, “D1.4 - Final Report on System Requirements and Architecture,” 2017.

[25] symbIoTe consortium, “D2.5 - Final symbIoTe Virtual IoT Environment Implementation,” 2017.

[26] G. Correndo, M. Salvadores, I. Millard, H. Glaser and N. Shadbolt, “SPARQL query rewriting for implementing data integration over linked data,” in *Proceedings of the 2010 EDBT/ICDT Workshops*, ACM, 2010, p. 4.

[27] M. Konstantinos, N. Bikakis, N. Gioldasis and S. Christodoulakis, “SPARQL-RW: transparent query access over mapped RDF data sources,” in *Proceedings of the 15th International Conference on Extending Database Technology*, ACM, 2012, pp. 610-613.

[28] B. Quilitz and U. Leser, “Querying distributed RDF data sources with SPARQL,” in *European Semantic Web Conference*, Springer, 2008, pp. 524-538.

[29] P. Visser, D. M. Jones, T. J. Bench-Capon and M. J. Shave, “Assessing heterogeneity by classifying ontology mismatches,” in *Proceedings of the FOIS*, 1998.

[30] P. Visser, D. M. Jones, T. J. Bench-Capon and M. J. Shave, “An analysis of ontology mismatches; heterogeneity versus interoperability,” in *AAAI Spring Symposium on Ontological Engineering*, Stanford CA., USA, 1997.

[31] M. Klein, “Combining and relating ontologies: an analysis of problems and solutions,” in *IJCAI-2001 Workshop on ontologies and information sharing*, 2001.

[32] F. Scharffe, O. Zamazal and D. Fensel, “Ontology alignment design patterns,” *Knowledge and Information Systems,* 2014.

[33] M. Rebstock, J. Fengel and H. Paulheim, Ontologies-based business integration, Springer Science & Business Media, 2008.

[34] I. Horrocks, P. Patel-Schneider, H. Boley, S. Tabet, B. Grosof and M. Dean, “SWRL: A semantic web rule language combining OWL and RuleML,” *W3C Member submission,* vol. 21, p. 79, 2004.

[35] J. Euzenat, F. Scharffe and A. Zimmermann, “Expressive alignment language and implementation,” 2007.

[36] P. Bouquet, F. Giunchiglia, F. Van Harmelen, L. Serafini and H. Stuckenschmidt, “C-OWL: Contextualizing ontologies,” in *International Semantic Web Conference*, Springer, 2003, pp. 164-179.

[37] A. Maedche, B. Motik, N. Silva and R. Volz, “MAFRA—An Ontology MApping FRAmework in the Context of the Semantic Web,” in *Workshop on Ontology Transformation at ECAI-2002*, 2002.

[38] P. Shvaiko and J. Euzenat, “A survey of schema-based matching approaches,” in *Journal on data semantics IV*, Springer, 2005, pp. 146-171.

[39] E. Rahm and P. A. Bernstein, “A survey of approaches to automatic schema

matching," *the VLDB Journal,* vol. 10, no. 4, pp. 334-350, 2001.

[40] B. T. Le, R. Dieng-Kuntz and F. Gandon, "On ontology matching problems," *ICEIS (4),* pp. 236-243, 2004.

[41] M. Granitzer, V. Sabol, K. W. Onn, D. Lukose and K. Tochtermann, "Ontology alignment—a survey with focus on visually supported semi-automatic techniques," *Future Internet,* vol. 2, no. 3, pp. 238-258, 2010.

[42] Á. Siciliaa, G. Nemirovskib and A. Nolleb, *Map-On: A web-based editor for visual ontology mapping.*

[43] S. Massmann, S. Raunich, D. Aumüller, P. Arnold and E. Rahm, "Evolution of the COMA match system," in *Proceedings of the 6th International Conference on Ontology Matching-Volume 814*, 2011, pp. 49-60.

[44] M. Kerrigan and A. Mocan, "The web service modeling toolkit," in *European Semantic Web Conference*, Springer, 2008, pp. 812-816.

[45] H. Rijgersberg, M. van Assem and J. Top, "Ontology of units of measure and related concepts," *Semantic Web,* pp. 3-13, 2013.

[46] M. Botts and A. Robin, *SensorML: Model and XML Encoding Standard 2.0,* OGC 12-000, 2014.

[47] P. Visser, D. Jones, T. Bench-Capon and M. Shave, "Assessing heterogeneity by classifying ontology mismatches," *Proceedings of the FOIS. vol. 98,* 1998.

[48] R. Studer, V. R. Benjamins and D. Fensel, "Knowledge engineering: principles and methods," *Data & knowledge engineering,* 1998.

8 Acronyms

API	Application Programming Interface
BIM	Best Practice Information Model
C-OWL	Context OWL
CIM	Core Information Model
EDOAL	Expressive and Declarative Ontology Alignment Language
IAM	Identity & Access Management
ID	Identifier
IM	Information Model
IoT	Internet of Things
IoT-EPI	IoT-European Platforms Initiative
JSON	JavaScript Object Notation
JSON-LD	JSON-based Serialization for Linked Data
KES	Knowledge Engineering System
LCIM	Levels of Conceptual Interoperability Model
MAFRA	Ontology MApping FRAmework
MIM	Meta Information Model
N3	Notation3
OData	Open Data Protocol
OWL	Web Ontology Language
PIM	Platform-Specific Information Model
QoS	Quality of Service
QR	Query Results
RDF	Resource Description Framework
RDFa	Resource Description Framework in Attributes
RDFS	RDF Schema
REST	Representational State Transfer
SE	Core Security Handler
SIM	symbIoTe Information Model
SLA	Service Level Agreement
SPARQL	SPARQL Protocol and RDF Query Language
SPIN	SPARQL Inferencing Notation
SQL	Structured Query Language
SSN	Semantic Sensor Network

SWRL	Semantic Web Rule Language
symbIoTe	Symbiosis of Smart Objects across IoT Environments
Turtle	Terse RDF Triple Language
UoM	Units of Measurements
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WGS	World Geodetic System
WKT	Well-Known Text
XML	Extensible Markup Language
XSD	XML Schema Definition