Semantic IoT Solutions – A Developer Perspective

Research · October 2019

DOI: 10.13140/RG.2.2.16339.53286

19 authors, including:

Martin Bauer
NEC Laboratories Europe GmbH
76 PUBLICATIONS **1,315** CITATIONS
SEE PROFILE

Hamza Baqa
Institut Polytechnique Paris
8 PUBLICATIONS **21** CITATIONS
SEE PROFILE

Sonia Bilbao
Tecnalia Corporación Tecnológica
18 PUBLICATIONS **40** CITATIONS
SEE PROFILE

Aitor Corchero
Eurecat
3 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Semantic IoT Solutions - A Developer Perspective[1]

Semantic Interoperability White Paper

Editor

- Martin Bauer, NEC Laboratories Europe

Contributors

- Hamza Baqa, Easy Global Market
- Martin Bauer, NEC Laboratories Europe
- Sonia Bilbao, Tecnalia Corporación Tecnológica
- Aitor Corchero, Eurecat - Technology Centre of Catalonia
- Laura Daniele, TNO
- Iker Esnaola, IK4-TEKNIKER
- Izaskun Fernández, IK4-TEKNIKER
- Östen Frånberg,1A Konsult
- Raúl García-Castro, Universidad Politécnica de Madrid
- Marc Girod-Genet, Institut Mines-Télécom
- Patrick Guillemin, ETSI
- Amélie Gyrard, Kno.e.sis, Wright State University
- Charbel El Kaed, Google
- Antonio Kung, TRIALOG
- Jaeho Lee, University of Seoul
- Maxime Lefrançois, École des Mines de Saint-Étienne
- Wenbin Li, Orange
- Dave Raggett, W3C
- Michelle Wetterwald, NETELLANY / FBConsulting

Table of Contents

Background

This paper is co-authored by an informal group of experts from a broad range of backgrounds, all of whom are active in standards groups, consortia, alliances and/or research projects in the Internet of Things (IoT) space.

The idea is to show how IoT systems can be built using semantic technologies, enabling semantic interoperability and thus allowing applications to reuse information originally provided for a specific application or IoT domain. The primary target audience is IoT developers that do not have a previous background in semantic technologies. The paper describes the different tasks and activities required when building semantic systems. The goal is to enable developers to build systems utilizing semantic technologies. It can be seen as one building block to achieve semantic interoperability in IoT and thus create the basis for a true Internet of Things.

1 Introduction

Semantic technologies have recently gained significant support in a number of communities, in particular the IoT community. An important problem to be solved is that, on the one hand, it is clear that the value of IoT increases significantly with the availability of information from a wide variety of domains. On the other hand, existing solutions target specific applications or application domains and there is no easy way of sharing information between the resulting silos. Thus, a solution is needed to enable interoperability across information silos. As there is a huge heterogeneity regarding IoT technologies on the lower levels, the semantic level is seen as a promising approach for achieving interoperability (i.e. semantic interoperability) to unify IoT device description, data, bring common interaction, data exploration, etc.

Semantic technologies have reached a good level of maturity and a number of standards and de-facto standards are available to implement semantic-based solutions. However, currently the widespread use is hindered by the fact that developers and system architects are not familiar with semantic technologies. The respective knowledge is still primarily limited to a group of experts. Thus, the purpose of this white paper is to spread this knowledge further and to show the developer community how semantic solutions can be implemented and how semantic interoperability can be achieved. The goal is to demonstrate the practical feasibility of the approach. The approach followed in the paper is supported with an example to go through different aspects and activities that are needed when developing semantic solutions. For each of the steps, useful tools with short descriptions

and relevant links are provided. Depending on the respective requirements, we guide developers to choose the appropriate tool according to their needs.

2 Problem Description

In this section, we describe the problem space in which semantics can be applied, and we explain why it is needed to provide platform, system or domain interoperability.

Several studies [1], as well as alliances like AIOTI [2], have demonstrated the fragmentation of the IoT ecosystem in terms of standardization, architectures and available technologies and IoT service platforms.

Accordingly, measurements and data available in a certain IoT system or implementation are often not accessible by different digital systems. Furthermore, these digital systems and the data they handle are often still strongly dependent on the vertical domain (e.g. water, energy, agriculture, etc.) in which they are implemented.

Therefore, there is a need for interoperability to address the current fragmentation in the IoT ecosystem and foster cross-domain exchange of measurements and data. There are several levels of interoperability identified by the existing literature (e.g., the GridWise Context-setting Framework v1.0 [3], an earlier AIOTI report [4] and ETSI [5], as follows (also see Figure 1):

- **Technical Interoperability** (connectivity, network) is usually associated with hardware/software components that enable communication. It presupposes an agreement how the information is transported across multiple communication networks and the protocols needed.
- **Syntactic Interoperability** is usually associated with data formats. Messages transferred by communication protocols and their payload need to have a well-defined, agreed syntax and encoding.
- **Semantic Interoperability** is associated with the meaning of the content that is exchanged. This requires agreement on common concepts and their relationships.
- **Organizational Interoperability** is the ability of organizations to effectively communicate and transfer meaningful information among a variety of different information systems and infrastructures. Organizational interoperability depends on successful technical, syntactic and semantic interoperability.

For communication across different IoT systems, the semantic level is essential to achieve interoperability. To that end, both sides of the information exchange (i.e., the different IoT systems under consideration) must refer to a commonly agreed reference model. Ontologies

can be used to represent such common reference model. Ontologies provide a formal specification of a shared conceptualization [6], by formally defining relevant concepts, their attributes and the relationships between these concepts. For example, ontologies can be used to explicitly define the meaning of the data shared by an IoT device with other entities (such as devices, servers, processes, applications, users) that need to correctly interpret the information and commands contained in the transferred data in order to correctly act or react. Note that some people use the term “knowledge graph” as an equivalent term for “ontology”.

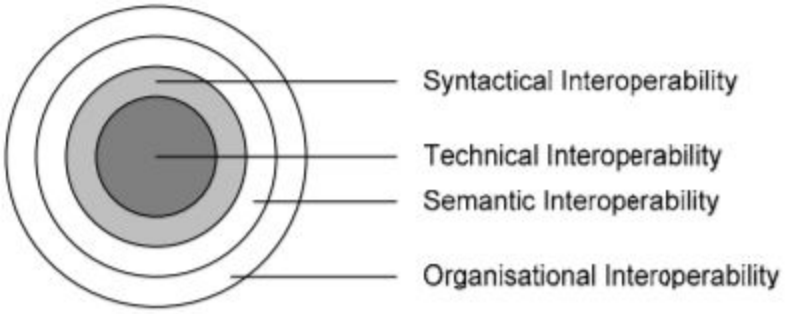

Figure 1. Different levels of interoperability [5]

Not only semantic interoperability enables interoperability at data level between different platforms and IoT systems, but also between various vertical domains. When an ontology is defined for one device from a vertical domain, e.g. agriculture, a generic interworking is enabled, i.e. the data can be understood by entities and devices operating in other domains (e.g., smart mobility or smart city). This enables IoT applications to interpret the containing information exchanged and support smarter decision-making because they collect, understand the meaning, and process data from all sorts of devices.

Within the next sections, a specific use case illustrates how semantic interoperability can be implemented and deployed.

3 Example Use Cases

Describe an example use case that instantiates the problem space, is as simple as possible, but shows the advantages of semantics and can be used in the following subsections.

3.1 Use case 1: Smart Home and Smart Grid

In our smart home use case, smart devices are connected to the smart grid (see Figure 2). The smart home resident wants to optimize the energy consumption of the house, but still be in control of key functionalities (e.g., the washing has to be done at a specific time, when the batteries of the electronic vehicle are recharged, and that the temperature in the house is kept within a certain range). The smart grid company offers the smart home resident a special tariff with significant discounts during times when a surplus of energy is available in exchange for energy consumption savings. Thus, the smart grid company balances the overall energy consumption and the happier smart home resident reduces the energy bill. As defined in a recent report commissioned by the European Commission on interoperable smart homes and grids [7], the ability of the home residents to adapt their electricity consumption in response to market signals is called “Demand Side Flexibility (DSF)”. To enable DSF, the home residents may adjust their demand by postponing some tasks that require large amount of electric power, or decide to pay a higher price for their electricity. To offer DSF, the home resident requires smart appliances that are able to offer flexibility to the smart grid company, such as a washing machine that can shift its power demand, or other appliances such as heat pumps, electric vehicles charging stations, etc. that are able to connect to the home network and act smart.

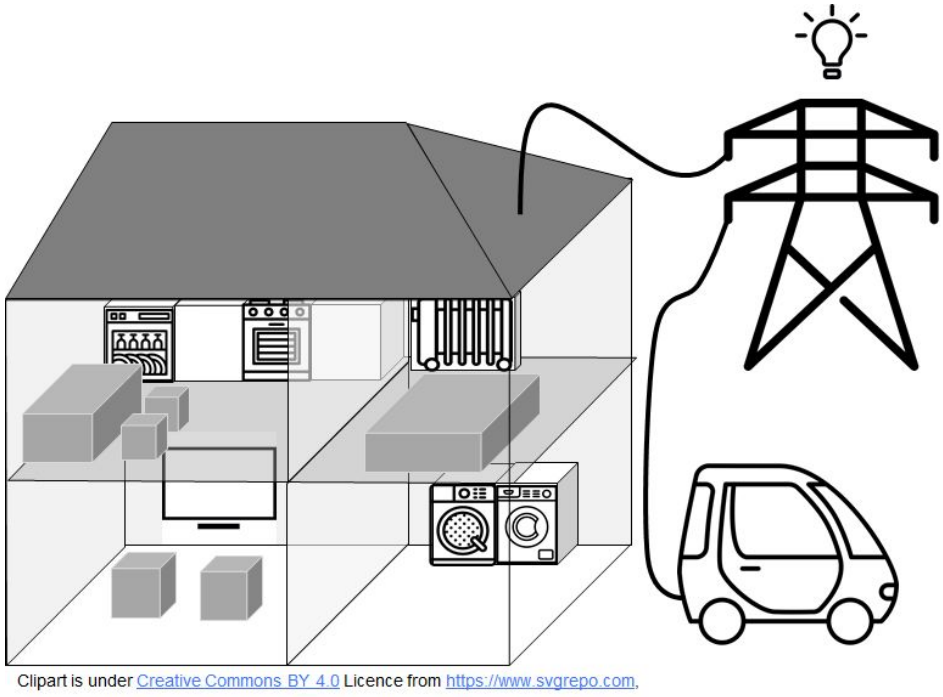

Figure 2: Smart home and smart grid use case

In order to implement the scenario, different systems have to be integrated allowing the following:

- Connecting controllable user devices in the smart home.
- Connecting the smart grid with the smart home.
- Providing the smart home resident device operation policies.
- Providing the smart grid operator time-dependent energy cost definition and request an energy-consumption profile.
- Optimizing energy consumption based on the time-dependent energy costs and energy consumption profiles in line with the operation policies and a possible consumption limit.

To achieve interoperability between the different systems in the smart home and the smart grid, agreement on interfaces and modelling of information is necessary. Thus, we show how the relevant information can be modelled on a semantic level to achieve semantic interoperability.

Examples of information that needs to be modelled include:

- Device (Status, Control, Monitoring, Energy consumption profile, Operation policy).
- Estimated energy cost timeline.
- Energy consumption limit.

Example use cases that require interoperability and involve devices in the smart home and the smart grid are the following:

- Configuration of devices that want to connect to each other in the home network, for example, to register a new dishwasher to the list of smart home devices.
- (Re-)scheduling of appliances in certain modes and preferred times using power profiles to optimize energy efficiency and accommodate the customer's preferences.
- Monitoring and control status of the appliances.
- Reaction to special requests from the Smart Grid, e.g. incentives to consume more or less depending on current energy availability, or emergency situations that require temporary reduction of power consumption.

These use cases are associated with the following user stories described in IEC TR 62746-2 [8]:

- User sets up his/her devices.
- User is notified when the washing machine has finished working.
- User wants to schedule washing at 5:00 p.m. to benefit from the lowest electricity cost.
- User wants to limit his/her own maximum energy consumption.
- User offers flexibility and gives permission to optimize energy consumption (e.g., the freezer in a defined range for a specific time), if the grid faces (severe) stability issues.
- User is notified by the grid in case of emergency situations (e.g. blackout prevention).

3.2 Use Case 2: Ambient Assisted Living (AAL)

Ambient Assisted Living (AAL) is an additional use case (see Figure 3) that we will use to illustrate what needs to be done to achieve semantic interoperability in the context of the smart home. The smart home resident in this case is an elderly person that needs special support and requires continuous monitoring. Depending on their personal health, the monitoring may include vital parameters such as heartbeat, oxygen, temperature, urinal leakage, posture and fall detection. This information has to be continuously communicated to medical and caregiving personal.

Apart from the core health parameters, it is important to understand whether the person is following the daily routine, i.e. what activities are performed in what order, for example whether the interaction with smart appliances like the fridge and the electric kettle indicates that breakfast is being prepared, or whether the use of the washing machine and the subsequent use of the dryer shows that clothes have been washed. Such information is important for caregivers and the social environment (e.g. family and neighbours) to understand what elderly persons can still manage by themselves and where more help is required. The support of such a scenario is especially relevant in the context of an ageing society where limited resources need to be used efficiently and the quality of life for elderly people can be improved by allowing them to stay in their familiar environment for longer than currently possible [9].

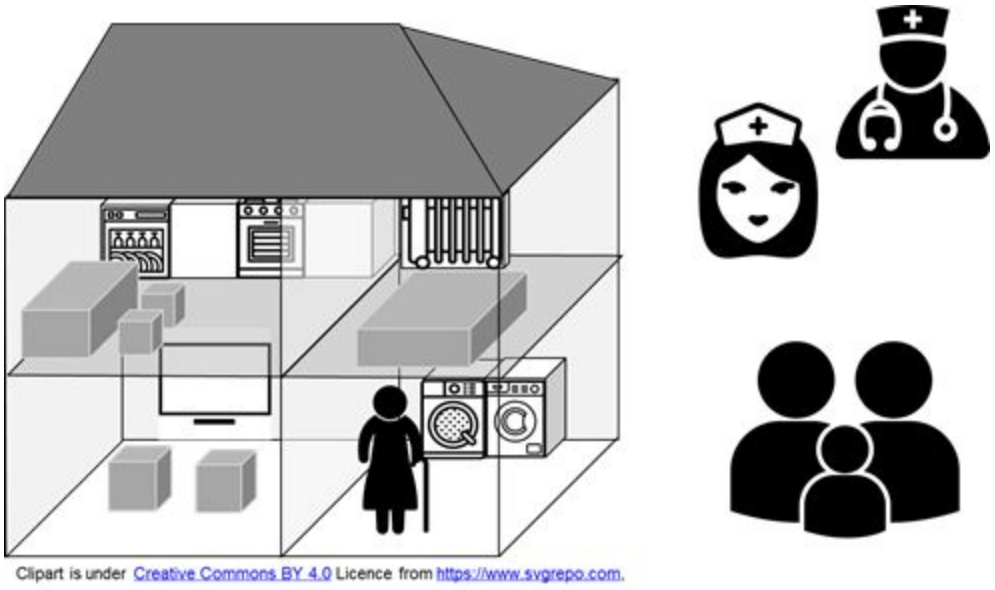

Figure 3: Ambient assisted living use case

To implement the scenario, different systems have to be integrated allowing the following:

- Connect body-worn sensors (heart rate sensor, temperature sensor, Oximeter, …) with each other (Smart BAN) and via a gateway to the network to enable access to caregivers and social environment.
- Enable access to status and energy consumption characteristics of devices (Smart Home) to identify user activity.
- Optimize comfort in the home based on patient information, i.e. control temperature, humidity, lighting condition – which needs to be taken into account by energy management in the smart home.

In this scenario, it is necessary to agree on interfaces and the semantic modelling of information to achieve interoperability between the different systems in the smart home, smart body-worn devices of the user and the applications of caregiving and medical staff, as well as family and social environment. Examples of information that needs to be modelled include:

- Status and energy usage profile of devices and (changes) in actual energy usage as basis for detecting user activities.
- Comfort profile and related parameters to be controlled (e.g. temperature, humidity, lighting conditions).

Example use cases that require interoperability and involve devices in the smart home, smart body-worn devices and applications are the following:

- To detect user activities, (changes) in actual energy usage have to be accessed and mapped to detailed energy usage profiles of devices. Additional user related information (from body sensors, location) can be integrated to improve activity detection accuracy.
- To create a comfortable atmosphere in the smart home, related parameters (e.g. temperature, humidity, lighting conditions) have to be controlled taking into account user profile and body sensor information.

The presented use-cases represent two vertical domains (energy and smart living) that need to interoperate in order to have common benefits in the interaction with the smart home devices: (i) energy efficiency and (ii) elderly people's wellbeing. Thus interoperability between both scenarios is required, i.e. the information needs to be shared and understood in the same way for both use cases and this means semantic interoperability has to be achieved.

4 Ontology Selection / Creation

This section aims to help developers willing to discover, select, reuse, integrate and, if necessary, develop ontologies. We recommend identifying the requirements of the ontology by defining a set of competency questions. Then, we strongly encourage to reuse existing ontologies by providing advice on how to discover and select the appropriate existing ontologies fitting developers' needs. In case the reuse is not enough, some guidelines for ontology development are provided. To give an example, we take aspects of our Smart Home/Smart Grid use case, where we need ontologies describing energy.

An ontology can represent a certain phenomenon, topic, or subject area through the description of classes, properties and instances (also known as individuals). Classes are abstract groups, sets, or collections of individuals and represent ontology concepts. Furthermore, these classes can have a hierarchical relation and can be arranged in taxonomies of superclasses and subclasses. Properties represent features or characteristics of individuals as well as the relationship between them. Finally, instances represent individuals of the classes described in the ontology.

Ontologies can be constructed based on different ontology languages such as the Web Ontology Language (OWL) [10,11]. OWL itself is based on the Resource Description Framework (RDF) [12] and RDF Schema (RDFS) [13], thus the vocabulary used for defining ontologies is a combination of concepts defined in RDF, RDFS and OWL. Certainly, an ontology language provides the expressive capability to encode knowledge about a specific domain and is often complemented with inference rules or validation rules that support the processing of such knowledge.

Figure 4 shows the different steps needed for finding, reusing, extending and, if necessary, creating the ontologies needed as the basis for building a semantic system.

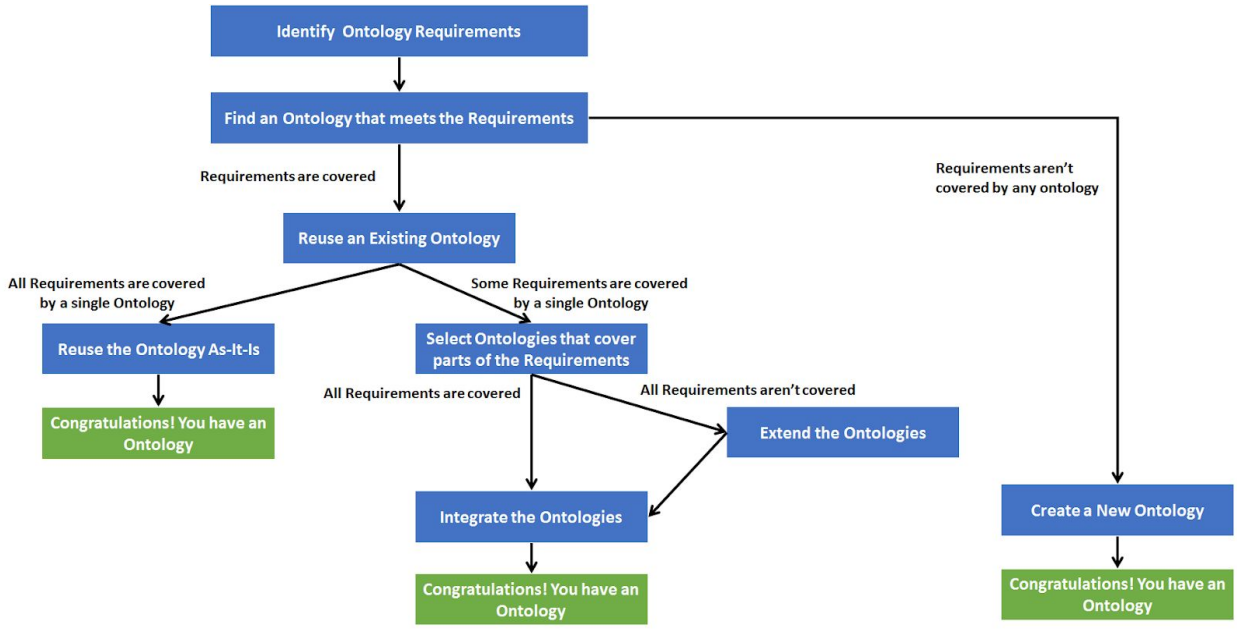

Figure 4: Ontology selection / creation diagram

4.1 Identify ontology requirements

First of all, it is necessary to define the purpose of the ontology. For that purpose, filling an ORSD (Ontology Requirements Specification Document) [14] may be helpful. It facilitates identifying the intended uses of the ontology, the end users, and the requirements the ontology should fulfill. These requirements will guide the developer during the creation of the ontology and can be used (iteratively during the ontology development or later, once ontology is developed) to validate if the ontology fulfills its intended, original purpose.

Use case ontology requirements

An excerpt of the ORSD is shown Figure 5 that can be used for the smart home/smart grid use case considered in this paper:

Ontology Requirements Specification Document
1. Purpose
The purpose of this ontology is to provide a knowledge model for the Smart Home and Smart Grid, and Ambient Assisted Living use cases.
5. Intended Uses
Use 1. Represent the information monitored by different devices (e.g. meters, thermostats, ...). (...)
6. Ontology Requirements
b. Functional Requirements
CQ01: Which is the electric consumption of a given house? CQ02: Which is the manufacturer of a given device? CQ03: Which is the location of a given washing machine inside a given house? CQ04: Which is the electric consumption of a given heat pump? (...)
7. Pre-Glossary Terms
a. Terms from Competency Questions
Electric Consumption House Washing Machine Heat Pump Device Manufacturer Location

Figure 5: Ontology Requirement Specification Document example

Other examples of ORSD that were used to specify the requirements that guided the creation of the Smart Applications REFerence ontology (SAREF) [15] suite of ontologies can be found in a number of ETSI Technical Reports [16–19].

4.2 Reuse existing ontologies

Once the purpose and the level of detail of the ontology are clear, it is necessary to define the concepts, properties and relationships that suit this purpose. Instead of creating an ontology from scratch, it is a best practice to reuse existing ontologies when possible due to the following reasons [20]:

- The sharing and reuse of ontologies increases the quality of the applications using them, as these applications become interoperable and are provided with a deeper, machine-processable and commonly agreed-upon understanding of the underlying domain of interest.

- It reduces the costs related to ontology development because it avoids the reimplementation of ontological components, which are already available on the Web and can be directly – or after some additional customization – integrated into a target ontology.

- It potentially improves the quality of the reused ontologies, as these are continuously revised and evaluated by various parties through reuse.

According to [20], ontology reuse can be understood as a three-step process: (i) ontology discovery, (ii) ontology selection, and (iii) ontology integration.

(i) Ontology discovery

It consists of finding appropriate ontologies that meet our requirements. By default, we encourage to look at ontologies supported by standardization such as ETSI SAREF [15], W3C & OGC SOSA/SSN [21,22] , W3C WoT TD [23], oneM2M Base Ontology [24], ETSI SmartBAN MyOntoSens, etc.), we refer the readers to look at the white paper *Towards Semantic Interoperability Standards based on Ontologies, Section 4.1* [25]. The task of finding appropriate ontologies can nowadays be facilitated due to the numerous ontology catalogs to find existing ontologies. Some of them are listed below, an extensive analysis of ontology catalogs for smart cities can be found in [26]:

- **Linked Open Vocabularies (LOV)** [27] designed by the Semantic Web community. This catalog is highly maintained and references ontology fitting their best practices criteria (e.g., ontology metadata).

- **Linked Open Vocabularies for Internet of Things (LOV4IoT)** [28] references more than 440 ontology-based projects relevant for IoT. It covers more than 20 domains: IoT, Wireless Sensor Networks (WSNs), Web of Things (WoT), smart home, smart energy, healthcare, smart city, robotics, etc. LOV4IoT is highly maintained with the inclusion of new references, ontology-based projects and domains.

- **READY4SmartCities** [29], an ontology catalog for smart cities.

Use case ontology discovery

Since the use case presented in this report is more oriented to smart homes rather than smart cities, we have decided to focus on the LOV4IoT catalogue for the following discussion.

In Figure 6, a screenshot of LOV4IoT is shown. With regards to the "Smart Home, Smart office, Building Automation, Activities of Daily Living Catalog" to which the presented use case belongs to, there are different ontologies that can be leveraged. Among them, SAREF [30] is one of the top recommended ontologies because it is shared online, it is referenced by the LOV community since it follows a set of best practices requested by the community, and it is highly maintained.

Smart Home, Smart office, Building Automation, Activities of Daily Living Catalog

Now the ontologies shared online are on the top, they are not classified by year anymore.

Authors	Year	Paper	Url onto	Technologies	Sensors	Rules	LOV status
Smart Appliances REFerence (SAREF), SAREF4EE, Daniele et al.	2014-2017	See below.	SAREF Ontology URL, SAREF doc, Project ETSI TS 103 264 V1.1.1 (2015-11) SmartM2M Smart Appliances; Reference Ontology and oneM2M Mapping	TopBraid, Alignement with SSN and OneM2M			Referenced by LOV: 15 June 2016
Paper: Towards IoT platform's integration : semantic translations between W3C SSN and ETSI SAREF [Moreira, Daniele et al. SEMANTICS 2017] Paper: Interoperability for Smart Appliances in the IoT World [Daniele et al. ISWC 2016] Paper: Created in close interaction with the industry: the smart appliances reference (saref) ontology [Daniele et al. 2015] White Paper: Study on Semantic Assets for Smart Appliances Interoperability [Daniele et al. 2015]							
Balaji et al.	2016	Paper: Brick: Towards a Unified Metadata Schema For Buildings [Balaji et al. 2016]	Ontology URL Documentation Ontology URL Brick project GitHub Brick project Concepts: Fire Safety System, HVAC, Fan, Pump, Valve, Lighting System, Water System, Building, Floor, Room, Alarm, Sensor, Command				Added to LOV4IoT: August 2017

Figure 6: LOV4IoT Catalog search

As for the LOV, Figure 7 shows ontologies related to the term "electric consumption" which is relevant for the use case at hand as it has been shown in the ORSD. The first results are

terms defined in the m3-lite taxonomy [31], whose main purpose is to extend the representation of concepts that are not covered by the SOSA/SSN [21] [22] ontology (e.g. different types of sensors or actuators) in a rather detailed way.

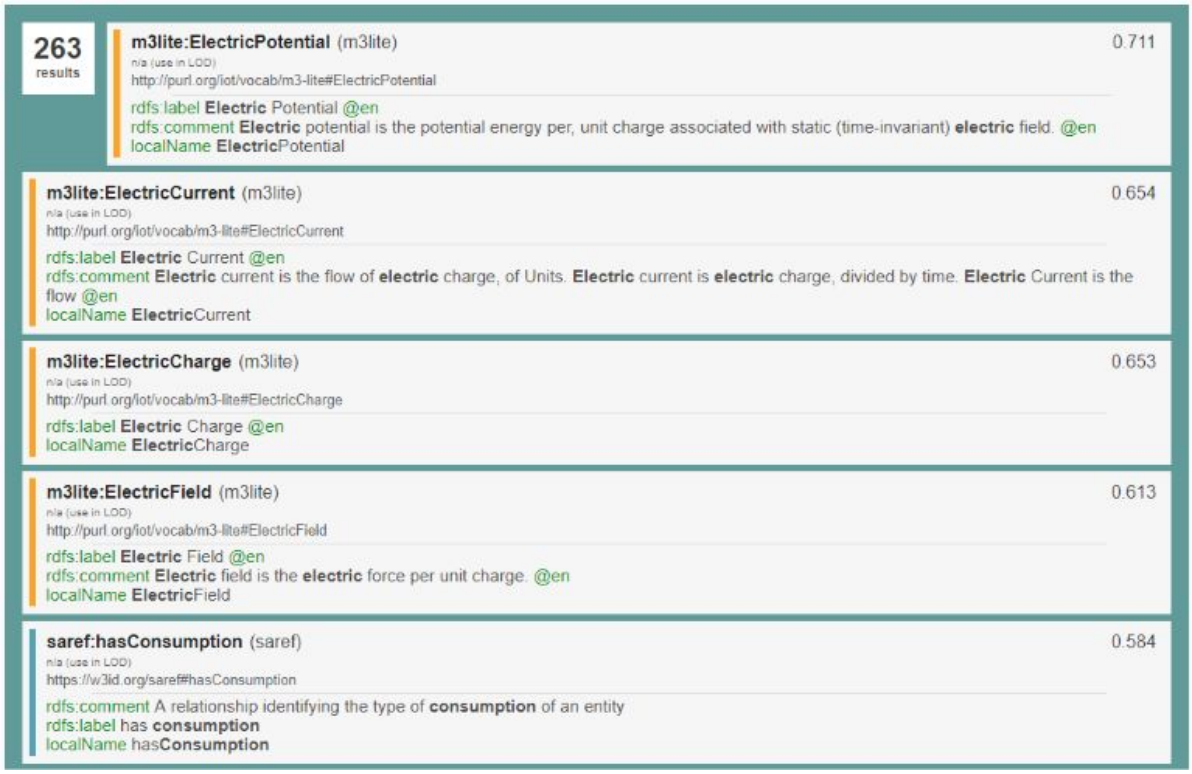

Figure 7: LOV ontologies related to electric consumption

(ii) Selection of suitable (parts of) ontologies

This task deals with assessing the usability of an ontology with respect to the use case requirements. This may result in an arduous task due to the different criteria that make ontologies suitable for a certain use case [32]. These criteria encompass the content of the ontology and the organization of their contents, the language in which it is implemented, the methodology that has been followed to develop it, the software tools used to build and edit the ontology, and the costs that the ontology will require in a certain project. Furthermore, the scarce documentation of ontologies can make this process even more difficult.

As already indicated above, we recommend to first look at ontologies supported by standardization activities (e.g., ETSI SAREF [15], W3C & OGC SOSA/SSN [21,22] , W3C WoT TD [23], oneM2M Base Ontology [24], ETSI SmartBAN MyOntoSens, etc.).

In case the developer needs to reuse only a subset of classes and properties of the ontology, instead of the whole ontology, an extractor tool can be used.

Limitations: Further effort is needed to improve ontology ranking algorithms to support developers to find suitable ontologies that match their needs.

In the context of the smart home/smart grid use case, let us consider the simple example of a temperature sensor in a room that can help to optimize energy efficiency in combination with other smart devices in the home. In this context, an existing ontology that meets the use case description is SAREF [15]. As shown in the UML diagram in Figure 8, by reusing parts of SAREF, our temperature sensor can be described as a Device (`saref:Device`) of type Sensor (`saref:Sensor`) that is provided with some static attributes, such as a Description (String), a Manufacturer (String) and a model (String). In order to provide some measurements, the temperature sensor needs to be in an ON state (`saref:OnState`). As a temperature sensor, this device performs a sensing function (`saref:SensingFunction`, which is a subclass of `saref:Function`) which has

- a range, a sensing time and the sensor type (`saref:Temperature`)
- an associated command (in our example we defined a `ex:GetTemperature` command as a subclass of the more general `saref:GetCommand`).

The temperature sensor is used to make a measurement that relates to Temperature (which is a `saref:Property`), has a unit of measure of type `saref:UnitOfMeasure` (°C in our example), and has a Value (22.0 in our example).

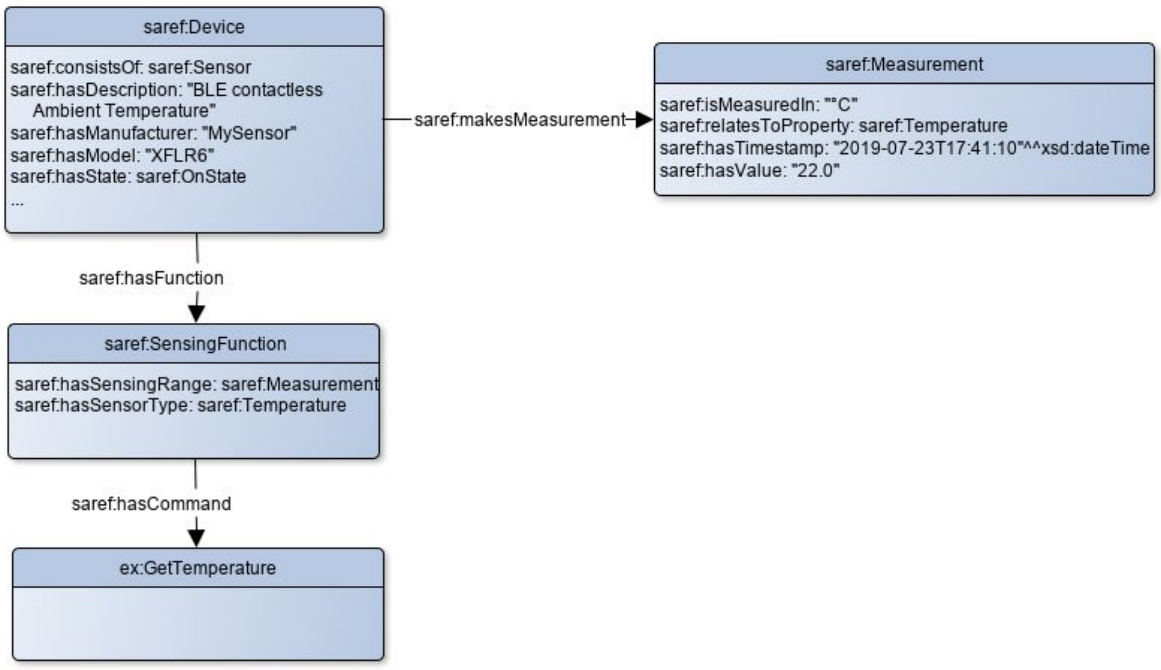

Figure 8: UML diagram representing a temperature sensor according to SAREF [30]

Use case ontology selection

After having identified existing ontologies that are suitable for the use case at hand, it has to be decided if these ontologies can be reused as they are. For our use case, the choice of reusing SAREF is based on its support by a standardization body (ETSI) and the extended community of users. However, SAREF does not cover all the use case requirements, so it is necessary to look further at other ontologies.

If multiple ontologies are imported at the same time, they may overlap to a certain extent in some of their parts and it is desirable to avoid redundancy of similar concepts to enhance interoperability. In our use case example, we find out that the m3-lite ontology [33] can be reused, as it contains terms related to "properties" that are not captured in SAREF. Therefore, SAREF can be integrated with classes from the m3-lite ontology. However, of the several classes of the m3-lite that are shown in Figure 9, we are actually interested only in the subclasses of `qu:QuantityKind`, "`qu:`" being the namespace for the NASA QUDT ontology for Units of Measure, Quantity Kinds, Dimensions and Types [34]. Thus we would like to reuse a subset of the m3-lite ontology and want to extract this subset. To that end, a Module Extractor Tool, e.g. the Locality Module Extractor Tool [35], can be used to reuse only the part of the ontology that is relevant to our use case.

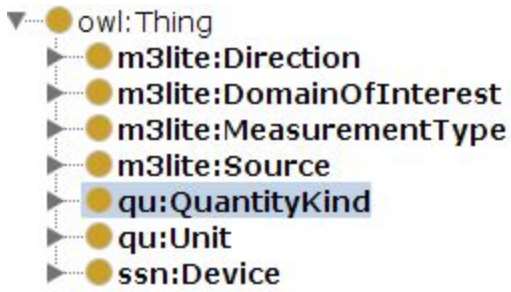

Figure 9: Classes of the m3-lite ontology

After running the Module Extractor Tool, we get an ontology module named "m3-lite_QuantityKindModule" that contains the `qu:QuantityKind` subclasses and their related axioms (see Figure 10). Comparing this module with the original m3-lite ontology, we can see how the size has been reduced, including only the terms that are relevant to our use case (i.e., "Properties"). The number of axioms of the complete m3-lite ontology have been reduced in the "m3-lite_QuantityKindModule" from 2035 to 360, and the number of classes from 451 to 178.

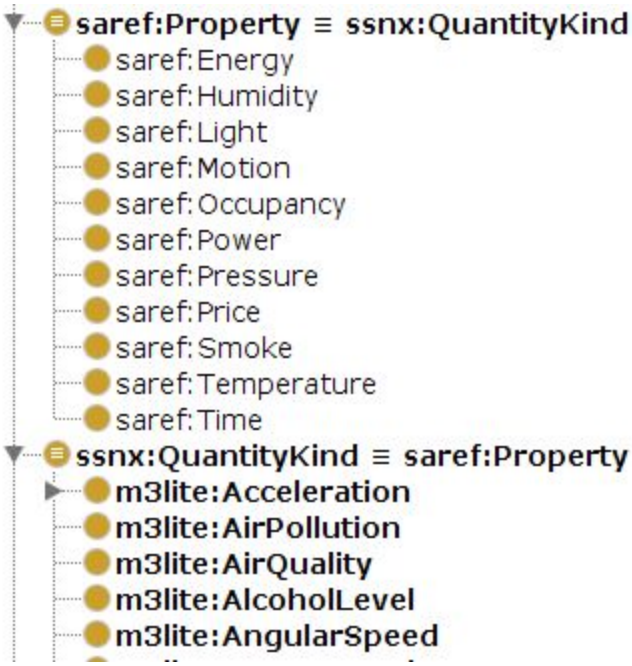

Figure 10: Classes extracted from m3-lite Ontology

(iii) Ontology integration

Finally, the selected ontology or combination of ontologies may need to be customized in order to further accommodate the use case's requirements. This customization may involve additional modification and integration operations such as extraction of ontology parts or even content and structural modification or extension.

When more than one ontology (or parts) are integrated, an ontology matching tool can be used to return a potential alignment between two ontologies. Some basic ontology matching tasks consist in setting relationships such as:

- Equivalences between concepts (with the `owl:equivalentClass` property) and between properties (with the `owl:equivalentProperty`)
- Subsumptions (with the `rdfs:subClassOf` or `rdfs:subPropertyOf` properties)
- Disjointness between concepts (with the `owl:disjointWith` property)
- Labels and comments to deduce similarities (with `rdfs:label` and `rdfs:comment` properties)

Use case ontology integration

The process to be followed to integrate the SAREF ontology and the "m3-lite_QuantityKindModule" module depends on the ontology design tool used (e.g., it can be done by importing the ontology within the Protege [36] ontology editor). Once integrated,

it needs to make explicit that the class `saref:Property` and `qu:QuantityKind` have the same adjacent semantics. That is, the equivalence between the two concepts needs to be set. This equivalence can be set with the following axiom:

```
saref:Property owl:equivalentClass qu:QuantityKind
```

Likewise, the equivalence can be set in the ontology design tool.

4.3 Create new ontology / Extend existing ontologies

If the existing ontologies do not meet all the requirements captured in the ORSD, then they need to be extended. Ontologies must be carefully designed and implemented, as these tasks have a direct impact on their final quality. Therefore, the use of well-founded ontology development methodologies such as the ontology development 101 [37], NeOn Methodology [38], On-To-Knowledge, DILIGENT are advised. The following ontology selection/creation process is inspired by the NeOn Methodology. In case no other existing ontologies match our specific requirements captured in the ORSD, it can be considered to develop a new ontology from scratch. Concerning ontology editing tools, Protégé [36] is one of the most popular software to learn how to create ontologies. Protégé provides a Graphical User Interface (GUI) to design and develop ontologies. One can either set up Protégé on their own computer or use the web collaborative Protégé tool. There are a set of excellent tutorials to develop your first ontology with Protégé [39] [37].

When creating or extending an ontology, some good practices should be followed for associating metadata to the ontology and to the terms it defines, and for extending or reusing existing ontologies. Section 8.3 in ETSI Technical Report 103 608 [40] is dedicated to these issues.

It is also advisable to follow the modularisation principle by separating the required knowledge in well-decoupled ontology modules. The main benefits of this principle are: 1) scalability for querying data and reasoning on ontologies, 2) scalability for evolution and maintenance, 3) complexity management, 4) understandability, 5) context-awareness and personalization, and 5) reuse [15]. For example, when some of the ontology modules are updated, thanks to the modularization, the impact of these changes in other modules and the global ontology is minimized. IoT-O [41] and FIESTA-IoT [33] ontologies are good examples of ontology modules.

Note that the extension and maintenance of ontologies require proper understanding on the resulting business impact. For instance a smart appliance using an extended ontology might no longer be interoperable with another smart appliance using the initial version. A maintenance strategy might therefore have to be defined prior to the implementation of an extension.

It is also recommendable to use Ontology Design Patterns (ODP) as building blocks to create new ontology modules. An ODP is a modeling solution to solve a recurrent ontology design problem. The ODP repository collects and makes ODPs available on the Web. It may contain a solution created by somebody else who already faced the same modeling challenge. For example, ETSI Technical Report TR 103 549 [42] lists Ontology Patterns that may be used for the IoT domain. Also, ETSI Technical Specification TS 103 548 [43] describes a standard ontology pattern to describe connected systems, along with guidelines on how to instantiate this pattern for different verticals.

Once the ontology is created, it is advisable to align it with related ontologies to make the ontology applicable to similar problems in different domains and scenarios.

5 Ontology Development & Instantiation

This section explains the creation of the semantic information based on the ontology concepts previously selected and adapted. The ontology provides the vocabulary describing a smart home to support the use case mentioned earlier.

Once the ontology, e.g. for assisted living or smart grid, has been refined and reaches a minimal viable state, it can be instantiated and be part of the use case or solution. The main objective is to instantiate an ontology which is populated by data from the sensors and actuators deployed in the context. For example, an ontology, similar to the one depicted in the UML diagram of Figure 8, exposes a device which makes measurements and has functions. This ontology needs to be populated with actual devices deployed in the real environment, as shown in Figure 11.

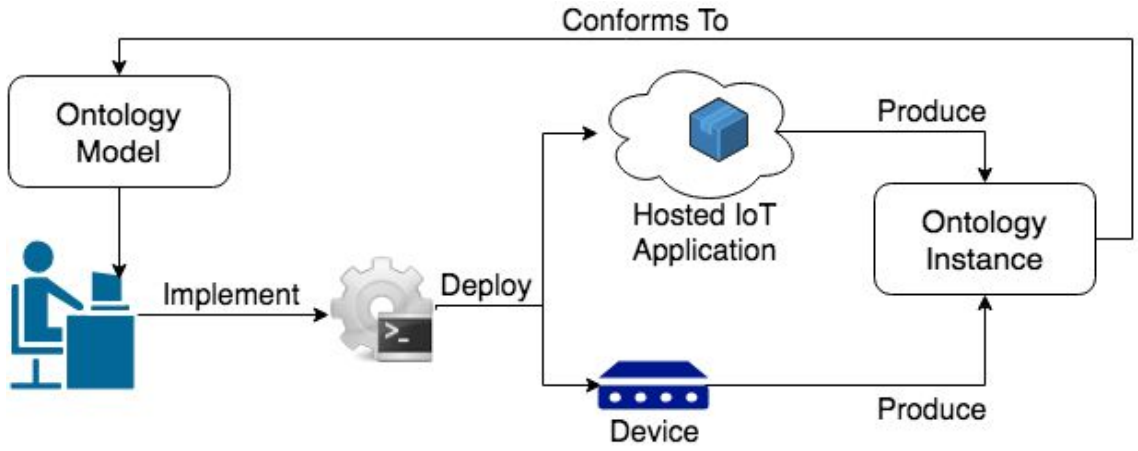

Figure 11: Creation of the dataset annotated with the ontology

The developers create the necessary software which can be deployed on a device firmware such as in [44] to produce data already conforming to the ontology. The software can also be deployed on a more complex system such as a building management system as in [45]. In this example, the data is collected locally from various sensors and gateways, then transformed to an ontology instance conforming to the ontology and then pushed to the cloud. In some cases, where it is not possible to easily update existing devices and systems, the software can be deployed on the cloud as in [46], in order to semantically annotate data to be compliant with the ontology and use cloud resources to save device power consumption.

The following example, depicted in Figure 12, specifies how to instantiate a light switch using the SAREF ontology, as described in [19]. This instantiation is referred to using the *saref-ls* prefix. Note that this prefix is different from the *saref* prefix, which indicates the SAREF ontology on which the *saref-ls* instantiation is built upon. The instantiation of an ontology is also called dataset.

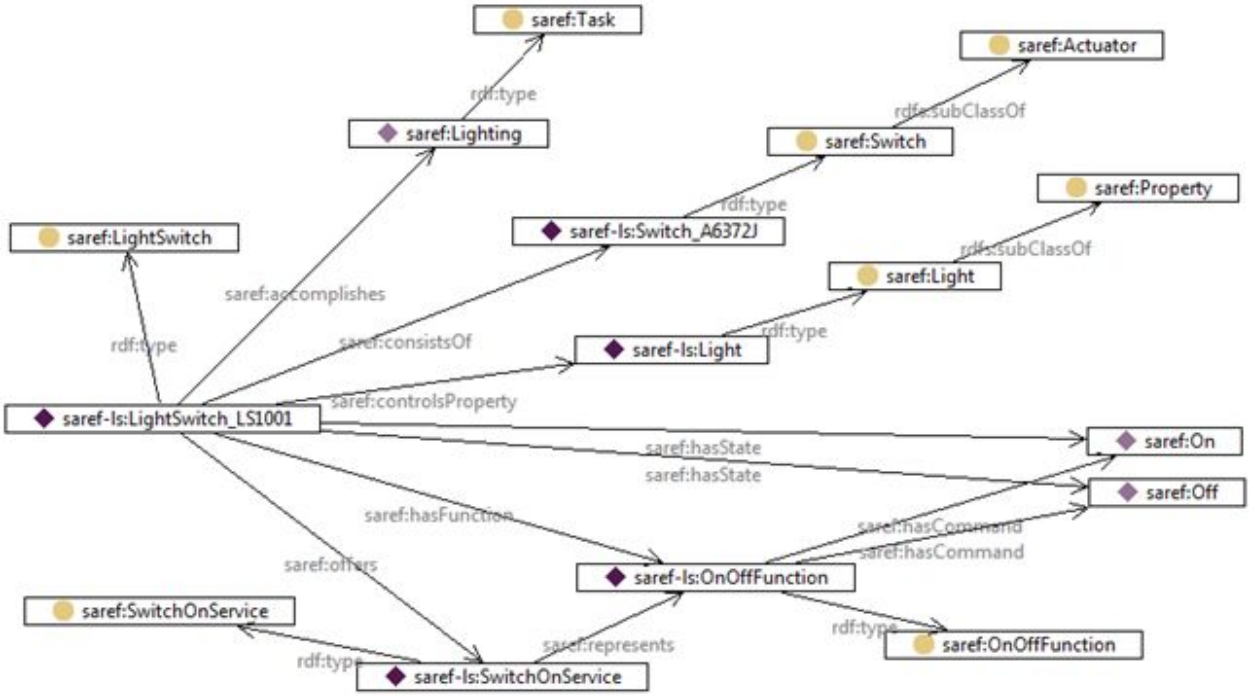

Figure 12: Light Switch example using SAREF (from ETSI TR 103 411 [19])

The light switch instance in Figure 12 is called `saref-ls:LightSwitch_LS1001` and represents a device of type `saref:LightSwitch`, which is a subclass of `saref:Actuator`. The light switch instance also has a human readable label `"Light switch LS1001"` and some properties that uniquely characterize it, namely its model, manufacturer and a human-readable description as follows:

- is designed to accomplish the task of `saref:Lighting`, which is of type `saref:Task`;
- consists of a `saref-ls:Switch_A6372J`, which is of type `saref:Switch` and is used for the purpose of controlling a property of type `saref:Light`;
- can be found in the states `saref:On` or `saref:Off`;
- performs a `saref:OnOffFunction`, which has the commands `saref:On` and `saref:Off` (note that the `saref:On` command acts upon a `saref:Off` state, while vice-versa the `saref:Off` command acts upon a `saref:On` state);
- offers a `saref-ls:SwitchOnService`, which in turn is of type `saref:SwitchOnService`. The `saref:SwitchOnService` is a representation of the `saref-ls:OnOffFunction` to allow the remote switch on of the lights through mobile phone devices that are connected to the local network.

Ontologies and their instantiations are modelled as triples following the RDF (Resource Description Framework) [12] mode. The triples have the form `<subject, predicate, object>`. As the object of one triple can be the subject of other triples, the overall structure becomes a graph, e.g. as the one shown in Figure 12. Subject and object are instances of ontology concepts/classes and predicates are (object) properties that are defined in the ontology as relating instances of a concept/class (domain) to other instances of a concept/class (range).

The RDF triples can be represented in different serialization formats. In particular, the following serialization formats are frequently used: Turtle [47], N-Triples [48], N-Quads [49], JSON-LD [50], N3 [51], RDF/XML [52] and RDF/JSON [53]. The serialization formats are to a large degree equivalent and the choice depends on the tasks and the available tools. In this paper, we primarily use a Turtle representation as it is compact and human-friendly with respect to readability. The Turtle code corresponding to Figure 12 is available at https://saref.etsi.org/saref/v2.1.1/example/lightswitch.ttl. The Turtle code of an additional example of how to instantiate a smart meter with an associated measurement using SAREF is available at https://saref.etsi.org/saref/v2.1.1/example/energymeter.ttl.

There are several libraries and tools which enables developers to accelerate the ontology instantiation as detailed in Section 10 on Software Implementation.

6 Semantic Information and Semantic Annotation

This section describes how the information instances created in the previous section are used – either having uniform semantic information, i.e there is only the semantic information, or using semantic information as annotation of existing non-semantic information, i.e. there is the original information in whatever form and semantic annotation that further describes this original information. Different representations can be used for the semantic information. Aspects of the smart home are described semantically using different representations.

To fully use semantic technologies, systems and platforms are expected to serve information with ontologies so that one can look up data content and get information from ontology definitions including the relationships between the terms in the ontology. Semantic information is regarded as any form of information containing explicit semantic descriptions and using ontologies to drive the information lifecycle. Comparing to classical syntax data, **semantic information is human and machine understandable and unambiguous** to support advanced data functions such as complex query, intelligent human-machine interaction, contextual data analytics and data interoperability.

In order to have semantic information on hand, we have typically two ways, i.e., Semantic Information Creation and Semantic Annotation, as detailed as follows. Both of the processes bridge the gap between syntax and semantics world with different application cases.

Semantic Information Creation produces semantic information using ontologies from scratch. The used ontologies specify the concepts and relations used in the information.

This is the most convenient way to create new semantic data based on semantic technologies, if no existing constraints apply. The semantic information built from scratch fully inherits the semantic benefits, while the required efforts are similar to the efforts of data creation following predefined schemas.

Semantic Annotation is the process of linking existing information in whatever format with specific ontologies to provide both machine understandable and human readable descriptions. This means that the original information is kept as it is and semantic information further describes this original information, i.e. can be seen as meta information. For example, the original information can be structured documents, services, functions, images and videos, etc. Ontologies provide semantics to existing data and furthermore link different information together via predefined relations.

This process is more suitable in cases where data already exists based on other specifications or the data sources can only provide data following specific formats without

semantics. Thus, the objective is to evolve the existing data with semantic technologies while keeping as much as possible backward compatibility with existing specifications.

In order to better illustrate the two processes, we present an example following our smart home scenario, in which a *room* with a URI `sd:Room1001` and an *energy limit profile* is equipped with a *temperature sensor* providing *temperature measurement* and a *washing machine* providing *washing machine states* and *remote washing services* to turn on/off and switch mode. We respectively introduce how we can build semantic information of the example following semantic information creation and semantic annotation; throughout the process, the main ontologies we use for semantic annotation are SAREF, which has been introduced in previous chapters, and the SAREF extension SAREF for Energy (S4ENER) that targets the energy domain.

1. Semantic Information Creation The semantic information creation builds the corresponding information from scratch. The general semantic information creation can be briefly summarized into the two following steps:

1. Identification or definition of ontologies to be used;
2. creation of semantic information by use of ontology concepts;

In our example, we describe the *Room1001* resource of type *Room* defined in our own ontology, (since the *Room* type is not defined in SAREF or S4ENER), and the *Room1001* has an energy profile which points to another resource "/Limit". We use the standard N-Triples as the serialization format and the output of the above descriptions are three triples as shown in Table 1.

Table 1 Semantic modelling and its representation in N-Triples

By doing so, we indicate that the *Room1001* is an instance of the `own:Room` class. The relation between the *Room1001* and the resource *"/Limit"* is further detailed in the `s4ener:hasEnergy` property, and the resource *"/Limit"* is actually an instance of the `s4ener:energyMax` class which specifies the maximum energy profile.
 	
`sd` is the prefix for the SAREF dataset.
	`sd:Room1001 rdf:type own:Room,`
	`sd:Room1001 s4ener:hasEnergy sd:Limit,`
	`sd:Limit rdf:type s4ener:EnergyMax`

Room1001 has two devices with ids *Ts001* and *Wm002*, which are respectively instances of `saref:TemperatureSensor` and	`sd:Room1001 s4ener:hasDevice sd:Ts001,`
	`sd:Room1001 s4ener:hasDevice sd:Wm002,`
	`sd:Ts001 rdf:type saref:TemperatureSensor,`
	`sd:Wm002 rdf:type saref:WashingMachine`

`saref:WashingMachine` classes. This is expressed by the relation `s4ener:hasDevice`.	

As the last step, we further add the descriptions of the two devices we just added.
 	
This example describes that the temperature sensor *Ts001* has a sensed value *25*; the washing machine *Wm002* has a state defined in *Wm002/state* (an instance of `saref:State` class) and offers a switch service defined in *wm002/switch* (an instance of `saref:Service` class).
	`sd:Ts001 saref:hasValue "25",`
	`sd:Wm002 saref:hasState sd:Wm002/state`
	`sd:Wm002/state rdf:type saref:State`
	`sd:Wm002 saref:offers sd:Wm002/switch`
	`sd:Wm002/switch rdf:type saref:Service`

By combining all the triples above, we get a complete description of our example following semantic information creation process. Through the whole process, we also link different information together by use of the properties defined in different ontologies, which further facilitates the data search and analytics.

Moreover, although we use N-triples as the serialization format in our example, the information we created can be easily transformed to other semantic serialization formats such as JSON-LD and RDF/XML.

2. Semantic Annotation Existing non-semantic information can be enriched with semantics and transformed to semantic information via semantic annotation. The general semantic process can be briefly summarized into the following three steps:

1. Preparation of source information to be annotated;
2. Identification or definition of ontologies to be used;
3. Manual or automatic link between source information and ontologies;

In Table 2 we present an example, where a non-semantic JSON representation can be annotated and transformed into a JSON-LD representation.

Table 2. Non-semantic JSON information and annotation transforming it into JSON-LD

The description of the rooms is serialized in JSON and thus in non-semantic form.	<pre>{ "id": "Room1001",
	"type": "Room",
	"energyProfile": "/Limit",
	"devices": [{"id": "Ts001",
	"type": "TemperatureSensor",
	"value": "25"},
	{ "id": "Wm002",
	"type": "WashingMachine",
	"state": "/state",
	"service": /switch" }]
	}</pre>

As a first step for mapping to JSON-LD, the JSON description of the Room1001 as the source information has to be annotated.
 	
For JSON-LD we need @id and @type for each element so that all ids in the JSON descriptions are defined as an object node with a URI as identifier, while all types are identified as @type, whose value will be an ontology class.
As corresponding information, i.e. id and type, already exist in JSON, a simple mapping is sufficient. In other cases the information may have to be added.
	<pre>"id": "@id",
	"type": "@type"</pre>

All type values in the JSON are mapped to different SAREF or SAREF for Energy (S4ENER) classes and properties, except the Room which is not defined in SAREF or S4ENER and thus needs to be defined in a different ontology.	<pre>"saref": "https://w3id.org/saref",
	
	"s4ener": "https://w3id.org/saref4ener",
	
	"own" "https://myOntology"
	
	"TemperatureSensor": "saref:TemperatureSensor",
	"WashingMachine": "saref:WashingMachine",
	"Room": "own:Room"
	
	"energyProfile":"s4ener:hasEnergy",
	"/Limit":"s4ener:energyMax",
	"value":"saref:hasValue",
	"service": "saref:offers",</pre>

	"devices":"s4ener:hasDevice", "Wm002/switch": "saref:Service", "state ": "saref:hasState", "Wm002/state": "saref:State"

This defined mapping is put into an "@context" element and added to the original JSON. As the result of this annotation, all terms used in JSON are linked to semantic concepts, for example, we now know that the resource "Wm002/switch" is a device service defined by SAREF, and the "/Limit" is a resource describing the maximum energy consumption as specified in S4ENER.	{
	"@context": {
	"id": "@id",
	"Wm002/state": "saref:State",
	"Room": "own:Room"
	… …
	},
	"id": "Room1001",
	"type": "Room",
	… …
	}

However, most documents cannot be straightforwardly transformed to RDF with this method. For these other cases, many tools are available off-the-shelf. The W3C wiki contains a list of tools [54] to generate RDF from a set of predefined data formats, or generic solutions to define transformations from a variety of data formats. On top of these, paper [55] proposes an approach to make web services and things on the Web of things reach semantic interoperability, while letting them the freedom to use their preferred formats.

7 Storing Semantic Information

Once the instance dataset is created , we make it accessible to applications. The traditional way is to store the information in a suitable and efficient way.

The instance dataset created needs to be stored and made accessible to applications and other components to later process it. For a fast prototyping, the semantic dataset could be stored in a file. The ideal solution is storing semantic datasets in specialized databases called **triple stores**. As RDF is based on triples, triple stores are optimized for storing and accessing these RDF triples. Other specialized data stores like graph databases can also be used.

When choosing the right storage for your semantic information, a number of different criteria have to be taken into account. An important aspect is the **scalability** of the database, what kind of request you typically execute (detailed in Section 8 Retrieving/Querying Semantic

Information), whether you want to be able to automatically infer information and what kind of **inference** is supported (detailed in the Section 9 Analytics and Reasoning), but also the **programming language supported**, tool integration and under what kind of **license** it is available / how much it **costs**. To help with the selection, benchmarks are used.

Several benchmarks for triple stores are being collected and published by W3C [34]. The W3C benchmark takes into account the type of inferencing you need in the project (RDF, RDFS, OWL), the license (commercial or open source) and the initial information capacity expected for your application. The performance capacity (how the semantic store performs the inferences and where the information is stored) is another key aspect to have in mind when selecting a semantic store. A big drawback of the semantic stores is that they need huge resources to perform the corresponding inferencing, process and load the information. This main drawback is derived mainly from the mix of storing the information in different files and in-memory. An overview of current triple store benchmarks can be found in [56,57].

Semantic repositories typically correspond to a server with a frontend. So, they usually provide common commands and front-end to load and query the information. Moreover, they also provide an API to connect the semantic store to our programs even directly using libraries (Jena, RDF4J, RDFLib, etc) or through REST services (HTTP Requests and responses, e.g. using the SPARQL 1.1 set of W3C recommendations). A common recommendation is to use the commands to load and update the information when large data-sets are present. We recommend using the user-interface when static data are only present or for testing purposes.

8 Retrieving/Querying Semantic Information

Once you have created instances of semantic information or annotated information, you want to make this information available to applications in a suitable and efficient way. For accessing semantic information, query languages and APIs have been defined. In our smart home energy use case, relevant information is about devices, their state, measurements and energy profiles.

As shown in the previous sections, semantic information is typically encoded as RDF triples (subject, predicate, object) in different representations. Objects in one triple can be subjects in other triples, so taking all triples together, we get a graph. An example is shown in Figure 13.

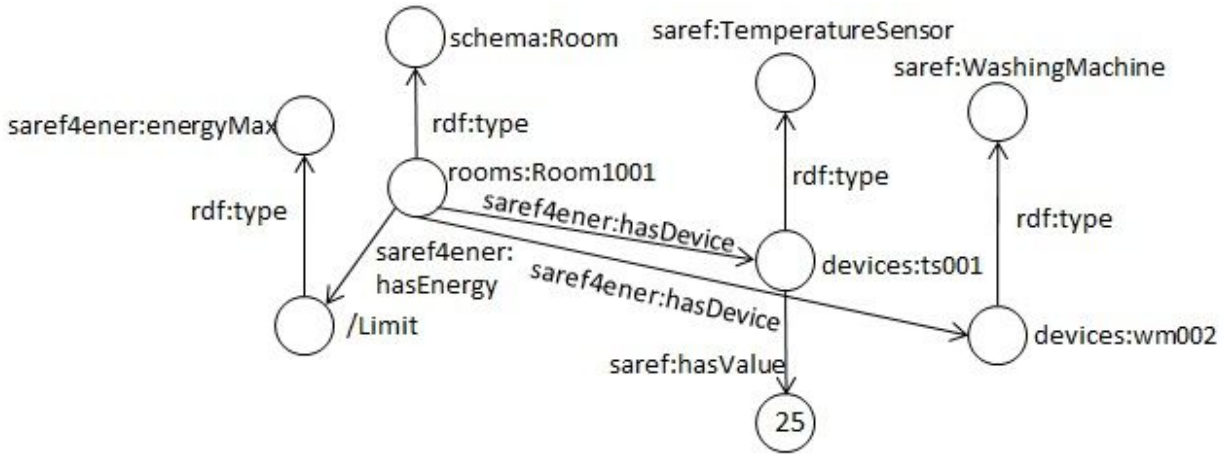

Figure 13: Example of RDF triples as graph

SPARQL (SPARQL Protocol and RDF Query Language) [58] is the most commonly used query language to query RDF graphs. SPARQL provides a set of query functionalities (similar to the SQL language), i.e. join, sort and aggregate, together with graph traversal syntax, e.g. as shown below:

Table 3. Set of questions in natural language and the corresponding SPARQL query and result.

Question	SPARQL query	Result

What devices are associated with Room1001?	`PREFIX s4ener: <https://saref.etsi.org/saref4ener#> PREFIX rooms: <https://myrooms.org> SELECT ?device WHERE { rooms:Room1001 s4ener:hasDevice ?device }`	The result is a set of matching assignments, i.e.
		`device`
		`<https://mydevices.org/ts001>`
		`<https://mydevices.org/wm002>`

What is the temperature in Room1001?	`PREFIX saref: <https://w3id.org/saref> PREFIX s4ener: <https://saref.etsi.org/saref4ener#> PREFIX rooms: <https://myrooms.org> SELECT ?temperature WHERE { rooms:Room1001 s4ener:hasDevice ?device`	The result is the following match:
		`temperature`
		`25`

	. ?device rdf:type saref:TemperatureSensor . ?device saref:hasValue ?temperature }	

As shown in the second example (Table 3), SPARQL enables joins across triples. This works well in centralized architectures – i.e. where all information is available locally – and can be extended to distributed architectures, in which distribution is limited or it is known where to find what triples. However, such expressiveness is problematic in highly distributed settings, where relevant triples could be found anywhere.

An API that targets semantic context information is NGSI-LD [59]. The underlying information model is based on entities, which have a semantic type. Entities have properties used to describe aspects of the respective entity and relationships to other entities. Thus the resulting model represents a graph (see Figure 14). Properties and relationships can again be further described with another level of properties and relationships. Overall, the NGSI-LD information model is less general, but provides a higher abstraction level.

It is based on JSON-LD, which is a representation of RDF – see example representation for *Room1001* (Table 4).

Table 4 NGSI-LD graph: visualization and code

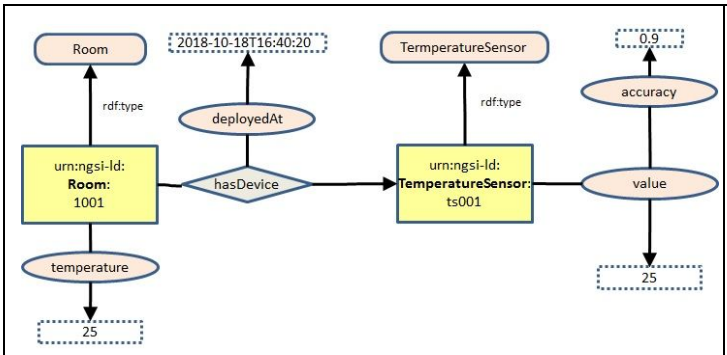

Figure 14: Example of NGSI-LD graph

```
{
        "id": "urn:ngsi-ld:Room:1001",
        "type": "Room",
        "temperature": {
                "type": "Property",
                "value": "25"
        },
        "hasDevice": {
                "type": "Relationship",
                "object":
"urn:ngsi-ld:TemperatureSensor:ts001",
                "deployedAt": {
                "type": "Property",
                "value":
"2018-10-18T16:40:20"
        }
        },
        "@context": [

"http://uri.etsi.org/ngsi-ld/coreConte
```

	xt.jsonld", "http://example.org/ngsi-ld/commonTerms.jsonld", "http://example.org/ngsi-ld/rooms.jsonld", "http://example.org/ngsi-ld/devices.jsonld"] }

The NGSI-LD API enables synchronous queries for entities, as well as asynchronous subscribe-notify interactions relating to changes in the information. The requested entities, properties and relationships can be specified and filtering of results can be based on property values and relationship objects. With requests based only on the entity type or existing properties/relationships, new entities can be discovered, e.g. the following query for the temperature of all Rooms where the temperature is larger than 20.

```
GET
/ngsi-ld/entities/?type=Room&attrs=temperature&q=temperature>20
Accept: application/ld+json
Link:           <http://example.org/cim/aggregatedContext.jsonld>;
rel="http://www.w3.org/ns/json-ld#context";
type="application/ld+json"
```

As location is highly relevant in real-world related use cases, NGSI-LD enables the geographic scoping of request – which may also be necessary to make entity type based discovery practical, e.g. request all entities within 2000m of a geographic coordinate:

```
GET /ngsi ld/entities/?type=Vehicle&georel=near;maxDistance==2000
&geometry=Point&coordinates=[8.684783577919006,
49.406131991436396]
Accept: application/ld+json
Link:           <http://example.org/cim/aggregatedContext.jsonld>;
rel="http://www.w3.org/ns/json-ld#context";
type="application/ld+json"
```

9 Analytics and Reasoning using Semantic Information

In this section we show how additional semantic information can be derived based on explicitly available information together with encoded domain understanding. In this way we get insights and create value. We give an overview of different analytics and reasoning approaches that can be used for this purpose and illustrate them with use case examples.

Not all information is explicitly provided by information sources like sensors. Some information, in particular higher-level information, has to be derived from base information, using knowledge about the domain that is encoded in some way.

For example, we have explicit information that `sd:Wm002` is a washing machine. With knowledge about the domain we know that `sd:Wm002` is also a device, as all washing machines are devices and that it consumes energy, as all devices consume energy. Due to the latter, `sd:Wm002` needs to be included in the home energy management.

In general, there are different formalism and mechanisms for deriving semantic information. In the following part of the document, we focus on two common types of reasoning in the area of semantics: ontology-based reasoning and rule-based reasoning. For ontology-based reasoning the inference rules used for deriving information are fixed by the ontology language. In the case of OWL, there are different profiles with different expressiveness, which define what logic aspects can be expressed and thus what can be derived through reasoning. The expressiveness has an influence on important properties like completeness, decidability and computational complexity. For example OWL DL corresponds to Description Logics, which is a particular decidable fragment of first order logic [10]. Another approach to reasoning is based on rules, e.g. in the form of antecedent ⇒ consequent, where both antecedent and consequent are conjunctions of atoms written as $a_1 \wedge ... \wedge a_n$. In the following, the different analytics and reasoning approaches are described in more detail. Some rules language to encode rules are Semantic Web Rule Language (SWRL) [58,60], SPIN rules [61], Notation3 [51] and SPARQL [58] construct queries.

9.1 Ontology-based Reasoning

Ontology languages define properties and their underlying semantics so that they can be used for reasoning. For example, RDFS defines the semantics of `rdfs:subClassOf` and `rdfs:subPropertyOf` (among others):

- `rdfs:subClassOf` - if A is of type B and B is a subClass of C then A is also of type C

- `rdfs:subPropertyOf` - if A and B are related by property C and C is a sub-property of D then A and B are also related by D

These properties are used when defining ontologies. For example in SAREF a washing machine is defined as a subClass of device:

```
saref:WashingMachine rdfs:subClassof saref:Device
```

When instantiating the ontology for a smart home, a user may define a specific washing machine Wm002 is of type WashingMachine:

```
sd:Wm002        rdf:type   saref:WashingMachine
```

Based on these two statements, a reasoner can infer that the specific washing machine is also a Device:

```
sd:Wm002        rdf:type   saref:Device
```

A similar examples is the following:

An ontology defines measuresTemperature as a sub-property of measures.

```
ont:measuresTemperature    rdfs:subPropertyOf    ont:measures
```

When instantiating the ontology a user defines a triple indicating that a temperature sensor makes a temperature measurement:

```
sd:Ts001        ont:measuresTemperature    sd:Measurement423
```

Based on these two statements, a reasoner can infer that also the property measures holds between Ts001 and Measurement423:

```
sd:Ts001        ont:measures          sd:Measurement423
```

OWL provides some further vocabulary that can be used to construct classes and properties, and to define logical axioms. For example `owl:inverseOf` and `owl:TransitiveProperty`:

- `owl:inverseOf` - if A is related to B by property C and C is inverse of D, then B is related to A by property D
- `owl:TransitiveProperty` - if A is related to B by property D and B is related to C by property D and D is a transitive property, then A is related to C by property D

Table 5 OWL inverse and transitive property examples

owl:inverseOf example	saref:isAccomplishedBy owl:inverseOf saref:accomplishes Wm002 saref:accomplishes WashingTask#table_ngsi-ld_graph_visualization_and_code → WashingTask saref:isAccomplishedBy Wm002
owl:TransitiveProperty example	isPartOf rdf:type owl:TransitiveProperty Room001 isPartOf Appartment005 Appartment005 isPartOf Building125 → Room001 isPartOf Building125

As indicated above, a reasoner is the software that is able to infer information based on user-provided instances (facts) and properties defined in the ontology (axioms). Reasoners differ with respect to the expressiveness they support (which relates to the underlying ontology language), whether they support incremental additions and removals of information, but also the interfaces and tool integrations that are available. A detailed comparison of reasoners can be found in [62].

9.2 Rule-based Reasoning

A rule-based reasoning provides simple IF THEN logical rules. It will enable deducing meaningful information from semantic sensor data (e..g, IF the room temperature is below 15 Degree Celsius, THEN the temperature in the room is considered as cold). For instance, Apache Jena is an open-source Java RDF library. The Jena framework [63] provides an inference engine (rule-based reasoning) to deduce meaningful knowledge from semantic datasets. AndroJena [64], a light version of the Jena framework, compatible with Android devices, also provides the query engine and the inference engine for constrained devices. The Jena inference engine is used to infer high-level abstractions by executing a set of "common sense" rules (e.g., following guidelines).

The example below shows a rule compliant with the Jena framework and the SOSA/SSN ontology to do analytics and reasoning using semantic information

Table 6 OWL inverse and transitive property examples

```
[BelowRoomTemperature:
              (?measurement rdf:type sensorTaxonomy:RoomTemperature)
              (?measurement sosa:hasSimpleResult ?v)
              greaterThan(?v,10)
              lessThan(?v,20)
              ->
              (?measurement rdf:type sensorTaxonomy:BelowRoomTemperature) ]
```

According to Wikipedia Air Quality Index [65] (AQI) guidelines, we can define a set of rules for air quality.

For instance, IF AirQualityIndex greaterThan 101 and LowerThan 150 THEN UnhealthyOutdoorAirQualityIndexUS. The Jena rule is implemented this way:

Table 7 Example of Jena rule regarding air quality index

```
[UnhealthyOutdoorAirQualityIndexUS:
(?measurement rdf:type sensorTaxonomy:OutdoorAirQualityIndex)
(?measurement
sosa:hasSimpleResult#table_owl_inverse_and_transitive_property_examples ?v)
greaterThan(?v,101)
lessThan(?v,150)
->
(?measurement rdf:type sensorTaxonomy:UnhealthyOutdoorAirQualityIndexUS) ]
```

Various rules (compliant with the Jena framework) can be provided by the Sensor-based Linked Open Rules (S-LOR) tool [66–68] which classifies rules per domain. Figure 15 shows a drop-down list with a set of IoT sub-domains such as smart home that we are interested in. Once, the domain is selected, the list of sensors relevant for this domain (e.g., presence detector, temperature, light sensor) are depicted. The developer clicks on the button “Get Project” to retrieve existing projects already using such sensors, or the “Get rule” button to find existing rules relevant for this sensor to deduce meaningful information from sensor data. For instance, for a temperature sensor within a smart home, the smart home application integrating a rule-base reasoner understands that when the temperature is cold or too hot, it can automatically switch on the heater or air-conditioning.

9.3 Other Reasoning

The Knowledge Acquisition Toolkit (KAT) tool [69] focuses on sensor data pre-processing. It is a machine-learning approach dealing with real-time data. KAT infers high-level abstractions from sensor data provided by gateways in order to reduce the traffic in network communications. KAT comprises three components: 1) An extension of Symbolic Aggregate Approximation (SAX) algorithm, called SensorSAX, 2) Abductive reasoning based on the Parsimonious Covering Theory (PCT), and 3) Temporal and spatial reasoning. It uses machine learning techniques (i.e. k-means clustering and Markov model methods) and rule-based systems to add labels to abstractions. KAT employs the abductive model rather than inductive or deductive approaches to solve the incompleteness limitation due to missing observation information. The tool is tested on real sensor data (i.e. temperature, light, sound, presence and power consumption).

10 Software Implementation

To ease developers' life, we introduce various kinds of frameworks, libraries and tools to develop semantics-based systems.

Several open source or proprietary frameworks and libraries allow developers to implement software components to instantiate ontologies. We present in the following an overview of each category.

10.1 RDF Management Libraries

Most programming languages have libraries to serialize, parse, store and manipulate RDF, and potentially interact with RDF triple stores, or reason.

Such libraries usually provide low level classes and functions to manipulate concepts directly mapped to the RDF language without any higher level abstractions. However, developers need to be aware of the technical aspects and theory of the RDF concepts and principles in order to implement an ontology-based solution.

Examples include Cowl [70], Redland RDF [71] or AutoRDF [72] for C/C++, Jena [63] or RDF4J [73] for Java, Ruby RDF [74] for Ruby, dotNetRDF [75] for .Net, RDFLib for Python [76], SWI-Prolog Semantic Web Library 3.0 [77] for Prolog, RDFjs [78] in JavaScript. A comparison of RDF libraries for JavaScript [79] can be found online.

We have provided a code example on Github [80] showing the use of Apache Jena Fuseki [81] with Java.

10.2 Object Relational Mappers (ORMs) and Ontology Library Generators

ORMs are built on top of RDF management libraries and provide an object oriented abstraction layer allowing developers to manipulate objects instead of RDF concepts. Several ORMs are available in various programming languages, such as: 1) KOMMA [82], Empire [83] and AliBaba [84] in Java, 2) RomanticWeb [85] and TrinityRDF [86] in .Net, and 3) RDFAlchemy [87] in Python. ORMs rely on the code decoration where a developer annotates the code with tags referencing IRIs from the ontology. Most of the Java ORM rely on the Java Persistence API (JPA) while the .Net ORMs rely on the Entity Framework. During the code implementation of an application, the developer requests a factory to instantiate the ontology and can generate SPARQL queries with SPARQL query builders or adapters such as the LINQ to SPARQL for the .Net technology. We discuss in the following the ORMs providing some code generation features. Tools such as the OWLBeans, AutoRDF, OLGA [45] (Ontology Library GenerAtor) can be used to generate libraries from an ontology. Such tools accelerate the adoption of Standard W3C Semantic technology among developers, by:

1. Reducing friction barrier for developers when working with an ontology;
2. Accelerating development of ontology based systems;
3. Eliminating complexity by providing Object Oriented libraries for developers.

These tools are based on a model driven approach taking as input an ontology file already defined by an ontology expert and a domain expert (as depicted in Figure 16). From this file, they generate a library based on the ontology model. The generated library can be imported and used when developing to:

- Generate an ontology instance conform to the ontology model.
- Query the generated ontology instance by relying on Object Oriented Model instead of SPARQL.

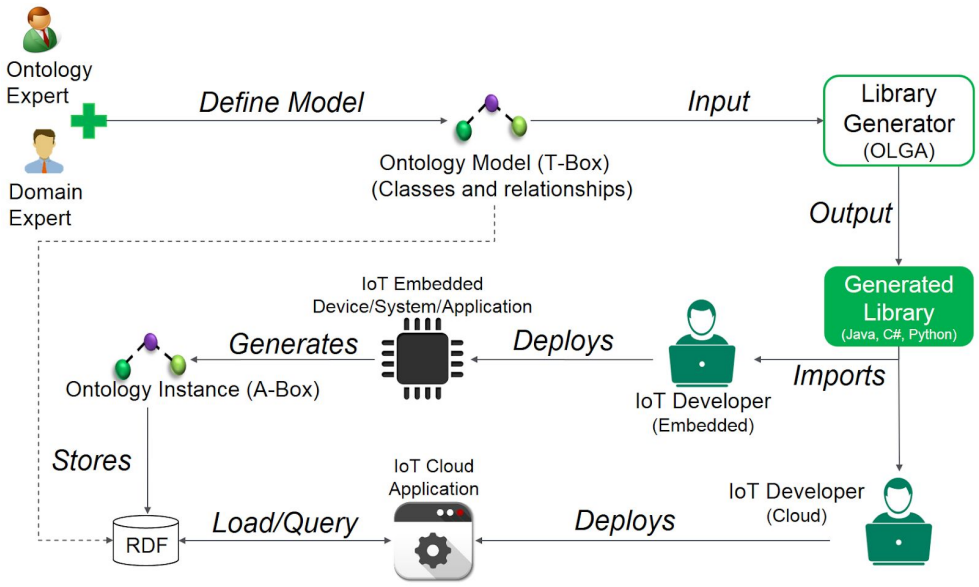

Figure 16: Ontology Library Generator (image from [45])

11 Semantic Interoperability Across Systems

Building semantic systems aims to achieve semantic interoperability across different systems. We discuss how semantic interoperability can be achieved based on the two example use cases smart home/smart grid and ambient assisted living, where we want to make the systems interoperable.

Semantic interoperability ensures that two systems exchange information with the same understanding regarding the meaning of the information. In traditional tightly-coupled systems the semantics is implicitly encoded in the system(s) by the programmers. IoT systems require an explicit understanding of the semantics to share and reuse information across systems that were not explicitly developed together, but integrated later. For a broader introduction to Semantic Interoperability, see the Whitepaper on Semantic Interoperability for the Web of Things [88].

Two use cases have been introduced in this white paper: 1) Smart Home and Smart Grid, and 2) Ambient Assisted Living. In the following, we discuss what is relevant for achieving semantic interoperability taking aspects of the two use cases for illustration purposes.

To achieve semantic interoperability, agreement on the semantic concepts ensures a common understanding on the information exchanged. As discussed in Section 4 Ontology

Selection / Creation, the respective concepts and related properties and possible values are defined in ontologies. The relevant concepts for the information exchange have to be identified. It depends on the status of the systems that should interwork. If none of the systems exists, they can be built together, selecting appropriate ontologies with a common subset for what is required for interoperability and domain-specific parts for both use cases. This can be done based on the description in Section 4.2 Reuse existing ontologies. In case one of the systems already exists as a semantic system, the relevant ontology aspects can be taken into account when designing the other system, either using them directly or defining a suitable mapping. If both systems are already semantic-enabled systems, a suitable mapping between the underlying ontologies has to be defined. The mapping may require changes to the existing system.

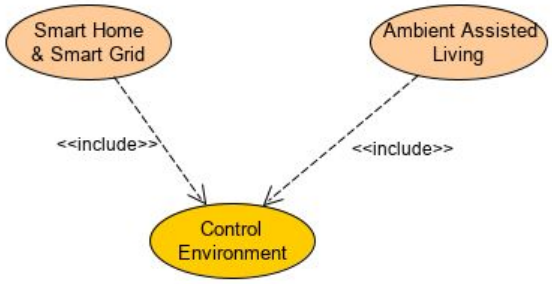

Figure 17: Control environment use case

Taking the example use case of controlling the environment, e.g. temperature and light conditions, in a home as shown in Figure 17, it can be seen that this is a joint use case that is part of the overall smart home & smart grid use case, because controlling the environment is related to energy consumption, but it is also part of the ambient assisted living use case, because controlling the environment is related to the wellbeing of the person living in the home. For the integration, both aspects have to be taken into account and thus relevant concepts like the target temperature range have to be mapped.

For example, if both systems use the SAREF ontology, no mapping is needed. Otherwise, the mapping is more complex, e.g. if one system uses the SOSA/SSN ontology and the other one SAREF. In this case, the relevant concepts and properties have to be mapped. OWL and RDFS provide the relevant vocabulary for such a mapping, i.e `owl:equivalentClass` and `rdfs:subClassOf` for mapping classes and `owl:equivalentProperty` and `rdfs:subPropertyOf` for mapping properties.

For example, SAREF defines `saref:Temperature` as `rdfs:subClassOf saref:Property`, whereas SSN only introduces `ssn:Property` as a generic concept for which temperature can be defined as a specific subclass. One possible way of mapping would be to make `saref:Temperature` also a subclass of `ssn:Property` - assuming that `saref:Property` and `ssn:Property` are not equivalent. If they were, `ssn:Property owl:equivalentClass saref:Property` could be defined instead,

making `saref:Temperature` also a subclass of `ssn:Property` by being a subclass of `saref:Property` and `saref:Property` and `ssn:Property` being equivalent.

When instantiating the semantic information based on the selected ontologies, different systems may choose different syntactic representations, e.g. as introduced in Section 5 Ontology Development & Instantiation. In this case a syntactic translation has to be performed when exchanging information between the systems. However, this is generally possible, as opposed to the case where there is a mismatch between semantic concepts, which cannot generally be resolved.

The translation can be realized directly between the ontologies used by the two systems or by translating each used ontology to a generic ontology that serves as a common reference. The mapping should occur at the interface between both systems to minimize the development effort.

In all these cases, there is a need to test the interoperability between the systems before putting them into service. The more different the semantics used by each component of the system are, the more complex the testing will have to be to ensure the service proper operation. The next section introduces the different steps needed to appropriately prepare this testing.

12 Testing Semantic Interoperability

Testing and validating the semantic IoT solution is the final step. In this section, we describe how developers plan and test their IoT system, and its interaction with other systems using the same, or a different ontology.

Developers wishing to test the semantic interoperability of their implementation should follow the steps below [89]:

1. identifying the features to be tested and when relevant, the set of standards against which the interoperability test will be run. The features to be tested can be basically divided into two categories: ontology management (i.e. acquisition, storage, update of the ontology as well as instantiation of the ontology mapped to the data structure of the tested implementation) and data management, which tests the usage of the ontology by the implementation i.e. its capability to generate a request for a specific data, to understand unambiguously the reply and to reply to a data request from another entity.

2. determining the testing configurations to be run according to the objectives of the previous step. Possible testing configurations range from a very simple case with a sensor and an IoT platform, up to a more complex scenario with two platforms, an

IoT device on one side and an IoT application on the other side, with even more complex configurations where the two IoT platforms use different ontologies and a mapping between these ontologies is required in one of the platforms. Figure 18 shows a basic example where an IoT device registered at one platform reports a measurement (in our example the room temperature) to an application registered at a different IoT platform. In this example, both platforms are using the same ontology.

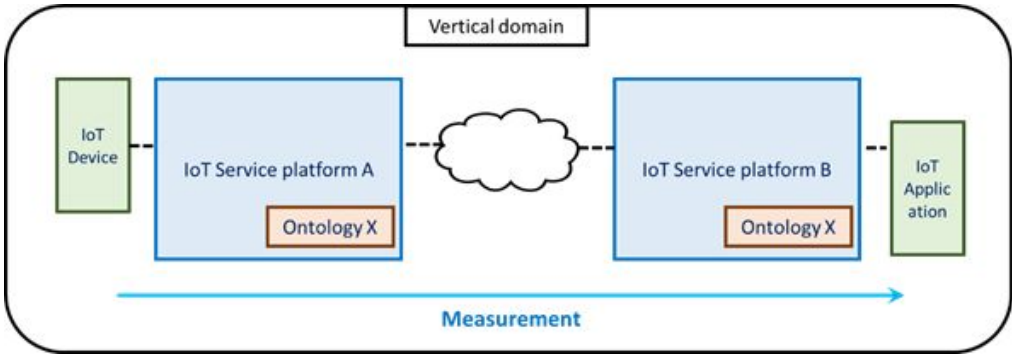

Figure 18: Example of testing configuration

Based on the selected configurations, scenarios and testing sequences need to be defined and documented before the test is executed, to ensure that the test will cover all the implementation details to be validated. A typical scenario defines the entities involved in the test (for example, IoT device, IoT platform, IoT application) and the scenario sequence:

- the starting point conditions,
- acquisition and storage of the tested ontology by the IoT platforms,
- generation of a request from one of the entities,
- reception by the second entity,
- verification of the correct understanding,
- generation of the reply,
- the failure cases associated to this type of sequence.

The last preparation step before running the test is to organise the testing environment, by defining the IT and infrastructure needed for its execution and preparing the testing report. The test report logs the issues and inconsistencies found in each testing sequence and serves as a reference when fixing those issues.

13 Overall Conclusion

This white paper aims to ease the development of semantic systems to achieve semantic interoperability and bridge information silos. We target the software developer and system architect audience who are getting familiar with semantic technologies. We give a step-by-step introduction to the different tasks required for the development of semantic systems, supported by a use case. There are appendices that complement the different tasks by providing links to additional tools to select when developing semantics-based projects, and giving guidance to select the appropriate tool fitting developers' needs.

As semantic technologies have gained a good level of maturity and there is increasing support in a number of communities, this paper lowers the hurdle of developing semantic systems. It represents one step towards a broader adoption of semantic technologies; a promising direction for overcoming the information silos that still exists in many domains and create value across domains, which is of particular importance in the Internet of Things.

References

1. ETSI. SmartM2M; IoT LSP use cases and standards gaps. 2016 Oct. Report No.: TR 103 376 V1.1.1. Available: https://www.etsi.org/deliver/etsi_tr/103300_103399/103376/01.01.01_60/tr_103376v010101p.pdf

2. AIOTI. WG3 IoT Standardisation Landscape. 2017. Available: https://aioti.eu/wp-content/uploads/2017/03/tr_103375v010101p-Standards-landscape-and-future-evolutions.pdf

3. GridWise Architecture Council. GridWise Interoperability Context-Setting Framework. 2008. Available: https://www.gridwiseac.org/pdfs/interopframework_v1_1.pdf

4. AIOTI. WG03 Semantic Interoperability, Release 2.0. 2015. Available: https://www.iab.org/wp-content/IAB-uploads/2016/03/AIOTIWG03Report2015-SemanticInteroperability.pdf

5. van der Veer H, Wiles A. ETSI White Paper No. 3 Achieving Technical Interoperability - the ETSI Approach. 2018. Available: https://www.etsi.org/images/files/ETSIWhitePapers/IOP%20whitepaper%20Edition%203%20final.pdf.

6. Gruber TR. Towards principles for the design of ontologies used for knowledge sharing. International Journal of Human-Computer Studies - Special issue: the role of formal ontology in the information technology. 1993;43: 625–640.

7. Daniele L, Strabbing W, Roelofsen B, Aalberts A, Stapersma P. Study on ensuring interoperability for enabling Demand Side Flexibility. 2019. Available: https://ec.europa.eu/digital-single-market/en/news/digitalising-energy-sector-ensuring-interoperability-enabling-demand-side-flexibility

8. IEC. Systems interface between customer energy management system and the power management system - Part 2: Use cases and requirements. 2015. Report No.: TR 62746-2.

Available: https://webstore.iec.ch/publication/22279

9. European Commission. The 2018 Ageing Report Economic & Budgetary Projections for the 28 EU Member States (2016-2070). 2018. Available: https://ec.europa.eu/info/sites/info/files/economy-finance/ip079_en.pdf

10. Recommendation W. OWL Web Ontology Language Guide. 2004 Feb. Available: https://www.w3.org/TR/owl-guide/

11. W3C. Web Ontology Language (OWL). 2012 Dec. Available: https://www.w3.org/OWL/

12. W3C. Resource Description Framework (RDF). 2004 Feb. Available: https://www.w3.org/RDF/

13. W3C. RDF Schema 1.1. 2014 Feb. Available: https://www.w3.org/TR/rdf-schema/

14. ORSD guide: How to Write and Use the Ontology Requirements Specification Document. [cited 19 Oct 2019]. Available: http://oa.upm.es/5474/1/INVE_MEM_2009_64393.pdf

15. Smart Applications REFerence Ontology (SAREF). [cited 19 Oct 2019]. Available: https://saref.etsi.org/saref#

16. ETSI. SmartM2M; SAREF extension investigation; Requirements for Industry and Manufacturing domains. 2018 Sep. Report No.: ETSI TR 103 507 V1.1.1. Available: https://www.etsi.org/deliver/etsi_tr/103500_103599/103507/01.01.01_60/tr_103507v010101p.pdf

17. ETSI. SmartM2M; SAREF extension investigation; Requirements for AgriFood domain. 2018 Sep. Report No.: TR 103 511 V1.1.1. Available: https://www.etsi.org/deliver/etsi_tr/103400_103499/103411/01.01.01_60/tr_103411v010101p.pdf

18. ETSI. SmartM2M; SAREF extension investigation; Requirements for Smart Cities. 2018 Sep. Report No.: TR 103 506 V1.1.1. Available: https://www.etsi.org/deliver/etsi_tr/103500_103599/103506/01.01.01_60/tr_103506v010101p.pdf

19. ETSI. SmartM2M; Smart Appliances; SAREF extension investigation. 2018 Feb. Report No.: TR 103 411 V1.1.1. Available: https://www.etsi.org/deliver/etsi_tr/103400_103499/103411/01.01.01_60/tr_103411v010101p.pdf

20. Simperl E, Cristina S, Ungrangsi R, Bürger T. Reusing ontologies on the Semantic Web: A feasibility study. Data & Knowledge Engineering. 2009. pp. 905–925. doi:10.1016/j.datak.2009.02.002

21. Sensor Observation, Sample and Actuator (SOSA) Ontology. [cited 19 Oct 2019]. Available: http://www.w3.org/ns/sosa/

22. Semantic Sensor Network (SSN) Ontology. [cited 19 Oct 2019]. Available: http://www.w3.org/ns/ssn/

23. Web of Things (WoT) Thing Description Ontology, W3C Editor's Draft. 18 Oct 2019 [cited 19 Oct 2019]. Available: https://www.w3.org/2019/wot/td

24. oneM2M Base Ontology. [cited 19 Oct 2019]. Available: https://git.onem2m.org/MAS/BaseOntology

25. Bauer M, Baqa H, Bilbao S, Corchero A, Daniele L, Esnaola I, et al. Towards Semantic Interoperability Standards based on Ontologies. 2019. doi:10.13140/RG.2.2.26825.29282

26. Gyrard A, Zimmermann A, Sheth A. Building IoT-Based Applications for Smart Cities: How Can

Ontology Catalogs Help? IEEE Internet of Things Journal. 2018. pp. 3978–3990. doi:10.1109/jiot.2018.2854278

27. Linked Open Vocabularies (LOV). [cited 19 Oct 2019]. Available: https://lov.linkeddata.es/dataset/lov/

28. Linked Open Vocabularies for Internet of Things (LOV4IoT). [cited 19 Oct 2019]. Available: http://lov4iot.appspot.com/

29. READY4SmartCities, an ontology catalog for smart cities. [cited 19 Oct 2019]. Available: http://smartcity.linkeddata.es/

30. ETSI. SmartM2M Smart Appliances; Reference Ontology and oneM2M Mapping. 2017 Mar. Report No.: TS 103 264 V2.1.1. Available: https://www.etsi.org/deliver/etsi_ts/103200_103299/103264/02.01.01_60/ts_103264v020101p.pdf

31. M3 Lite Taxonomy. 7 Sep 2017 [cited 19 Oct 2019]. Available: http://purl.org/iot/vocab/m3-lite#

32. Lozano-Tello A, Gómez-Pérez A. Ontometric: A method to choose the appropriate ontology. Journal of database management. 2004;2: 1–18.

33. Agarwal R, Fernandez DG, Elsaleh T, Gyrard A, Lanza J, Sanchez L, et al. Unified IoT ontology to enable interoperability and federation of testbeds. 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT). 2016. doi:10.1109/wf-iot.2016.7845470

34. Quantities, Units, Dimensions, and Types (QUDT) Ontology. 8 Oct 2019 [cited 19 Oct 2019]. Available: https://bioportal.bioontology.org/ontologies/QUDT

35. Locality Module Extractor Tool. In: Tools [Internet]. [cited 18 Oct 2019]. Available: https://www.cs.ox.ac.uk/isg/tools/ModuleExtractor/

36. Stanford University. Protégé tool. In: Stanford University [Internet]. [cited 20 Oct 2019]. Available: https://protege.stanford.edu/

37. Noy NF, McGuinness DL. Ontology development 101: A guide to creating your first ontology. 2001. Report No.: KSL-01-05. Available: https://protege.stanford.edu/publications/ontology_development/ontology101.pdf

38. Suárez-Figueroa MC, Gómez-Pérez A, Fernández-López M. The NeOn Methodology for Ontology Engineering. Ontology Engineering in a Networked World. 2012. pp. 9–34. doi:10.1007/978-3-642-24794-1_2

39. Horridge M. A Practical Guide To Building OWL Ontologies. 2011. Available: http://mowl-power.cs.man.ac.uk/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf

40. ETSI. SmartM2M; SAREF publication framework reinforcing the engagement of its community of users. 2019 Jul. Report No.: TR 103 608 V1.1.1. Available: https://www.etsi.org/deliver/etsi_tr/103600_103699/103608/01.01.01_60/tr_103608v010101p.pdf

41. Seydoux N, Drira K, Hernandez N, Monteil T. IoT-O, a Core-Domain IoT Ontology to Represent Connected Devices Networks. Lecture Notes in Computer Science. 2016. pp. 561–576. doi:10.1007/978-3-319-49004-5_36

42. ETSI. SmartM2M; Guidelines for consolidating SAREF with new reference ontology patterns, based on the experience from the ITEA SEAS project. 2019 Jul. Report No.: TR 103 549 V1.1.1.

Available:
https://www.etsi.org/deliver/etsi_tr/103500_103599/103549/01.01.01_60/tr_103549v010101p.pdf

43. ETSI. SmartM2M; SAREF consolidation with new reference ontology patterns, based on the experience from the SEAS project. 2019 Jul. Report No.: TS 103 548 V1.1.1. Available: https://www.etsi.org/deliver/etsi_ts/103500_103599/103548/01.01.01_60/ts_103548v010101p.pdf

44. Kaed CE, El Kaed C, Danilchenko V, Delpech F, Brodeur J, Radisson A. Linking an Asset and a Domain Specific Ontology for a Simple Asset TimeSeries Application. 2018 IEEE International Conference on Big Data (Big Data). 2018. doi:10.1109/bigdata.2018.8621972

45. El Kaed C, Ponnouradjane A. A Model Driven Approach Accelerating Ontology-based IoT Applications Development. Proceedings of the SIS-IoT workshop. Available: https://www.researchgate.net/publication/319650390_A_Model_Driven_Approach_Accelerating_Ontology-based_IoT_Applications_Development

46. Kaed CE, El Kaed C, Ponnouradjane A, Shah D. A Semantic Based Multi-Platform IoT Integration Approach from Sensors to Chatbots. 2018 Global Internet of Things Summit (GIoTS). 2018. doi:10.1109/giots.2018.8534520

47. W3C. RDF 1.1 Turtle - Terse RDF Triple Language. 2014 Feb. Available: https://www.w3.org/TR/turtle/

48. W3C. RDF 1.1 N-Triples - A line-based syntax for an RDF graph. 2014 Feb. Available: https://www.w3.org/TR/n-triples/

49. W3C. RDF 1.1 N-Quads - A line-based syntax for RDF datasets. 2014 Feb. Available: https://www.w3.org/TR/n-quads/

50. W3C. JSON-LD 1.1 - A JSON-based Serialization for Linked Data, Working Draft. 2019 Oct. Available: https://www.w3.org/TR/json-ld11/

51. Notation3 (N3): A readable RDF syntax. In: W3C [Internet]. [cited 19 Oct 2019]. Available: https://www.w3.org/TeamSubmission/n3/

52. W3C. RDF 1.1 XML Syntax. 2014 Feb. Available: https://www.w3.org/TR/rdf-syntax-grammar/

53. W3C. RDF 1.1 JSON Alternate Serialization (RDF/JSON). 2013 Nov. Available: https://www.w3.org/TR/rdf-json/

54. W3C Wiki: Converter to RDF. [cited 19 Oct 2019]. Available: https://www.w3.org/wiki/ConverterToRdf

55. Lefrançois M. RDF presentation and correct content conveyance for legacy services and the web of things. Proceedings of the 8th International Conference on the Internet of Things. ACM; 2018. p. 43.

56. W3C. RDF Store Benchmarking. Available: https://www.w3.org/wiki/RdfStoreBenchmarking

57. Saleem M, Khan Y, Hasnain A, Ermilov I, Ngomo A-CN. A fine-grained evaluation of SPARQL endpoint federation systems. Semantic Web. 2016. pp. 493–518. doi:10.3233/sw-150186

58. Recommendation W. SPARQL 1.1 Overview. 2013 Mar. Available: http://www.w3.org/TR/sparql11-overview/

59. ETSI Industry Specification Group on Context Information Management. Context Information Management (CIM); NGSI-LD. 2019 Jan. Report No.: ETSI GS CIM 009 V1.1.1 . Available: https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.01.01_60/gs_CIM009v010101p.pdf

60. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. 21 May 2004 [cited 19 Oct 2019]. Available: https://www.w3.org/Submission/SWRL/

61. SPIN - SPARQL Inferencing Notation. [cited 19 Oct 2019]. Available: https://spinrdf.org/

62. Abburu S. A Survey on Ontology Reasoners and Comparison. Int J Comput Appl Technol. 2012;57. Available: https://pdfs.semanticscholar.org/9091/e269a2cf7a44b46681b3de3ca489a36ad243.pdf

63. Apache Jena Framework. [cited 19 Oct 2019]. Available: https://jena.apache.org/

64. Andro Jena - Apache Jena Framework for Android. [cited 19 Oct 2019]. Available: https://github.com/sbrunk/jena-android

65. Air Quality Index. In: Wikipedia [Internet]. [cited 19 Oct 2019]. Available: https://en.wikipedia.org/wiki/Air_quality_index

66. Gyrard A, Serrano M, Jares JB, Datta SK, Ali MI. Sensor-based Linked Open Rules (S-LOR). Proceedings of the 26th International Conference on World Wide Web Companion - WWW '17 Companion. 2017. doi:10.1145/3041021.3054716

67. Serrano M, Gyrard A. A Review of Tools for IoT Semantics and Data Streaming Analytics. In: Soldatos J, editor. Building Blocks for IoT Analytics. River Publishers; 2016. pp. 139–163.

68. Gyrard A. Sensor-based Linked Open Rules (S-LOR) & Demo. [cited 20 Oct 2019]. Available: http://linkedopenreasoning.appspot.com/

69. Ganz F, Enshaeifar S, Puschmann D, Ahrabian A, Elsaleh T. Knowledge Acquisition Toolkit (KAT). [cited 19 Oct 2019]. Available: http://info.ee.surrey.ac.uk/CCSR/KAT/

70. COWL RDF. [cited 19 Oct 2019]. Available: https://github.com/scslab/cowl/tree/master/rdf

71. Redland RDF Libraries. [cited 19 Oct 2019]. Available: http://librdf.org/

72. AutoRDF - A framework for C++ proxy class generation from Web Ontology Language. [cited 19 Oct 2019]. Available: https://github.com/ariadnext/AutoRDF

73. Eclipse rdf4j - Scalable RDF for Java. [cited 19 Oct 2019]. Available: https://rdf4j.eclipse.org/

74. Ruby RDF - Public domain libraries for RDF & SPARQL in the Ruby programming language. [cited 19 Oct 2019]. Available: https://github.com/ruby-rdf

75. dotNetRDF - An Open Source .NET Library for RDF. [cited 19 Oct 2019]. Available: https://www.dotnetrdf.org/

76. RDFLib for Python. [cited 19 Oct 2019]. Available: https://github.com/RDFLib/rdflib

77. SWI-Prolog Semantic Web Library 3.0. [cited 19 Oct 2019]. Available: https://www.swi-prolog.org/pldoc/doc_for?object=section(%27packages/semweb.html%27)

78. RDF JavaScript Libraries. [cited 19 Oct 2019]. Available: https://github.com/rdfjs

79. Comparison of RDFJS libraries. In: W3C Community [Internet]. [cited 19 Oct 2019]. Available:

https://www.w3.org/community/rdfjs/wiki/Comparison_of_RDFJS_libraries

80. Semantic Whitepaper Code Example. [cited 19 Oct 2019]. Available: https://github.com/martin-p-bauer/SemanticWhitepaperCodeExample

81. Apache Jena Fuseki. [cited 19 Oct 2019]. Available: https://jena.apache.org/documentation/fuseki2/

82. KOMMA - RDF for Eclipse. In: Eclipse Marketplace [Internet]. [cited 19 Oct 2019]. Available: https://marketplace.eclipse.org/content/komma-rdf-eclipse

83. Empire - JPA implementation for RDF. [cited 19 Oct 2019]. Available: https://github.com/mhgrove/Empire

84. alibaba - Sesame RDF framework. [cited 19 Oct 2019]. Available: https://bitbucket.org/openrdf/alibaba/src/master/

85. Romantic Web - Object Mapping for RDF in .net. [cited 19 Oct 2019]. Available: http://romanticweb.net/

86. Trinity - A high-level API for RDF written in C#. [cited 19 Oct 2019]. Available: https://bitbucket.org/semiodesk/trinity/src/default/

87. RDFAlchemy - Object RDF Mapper for semantic web users. [cited 19 Oct 2019]. Available: https://rdfalchemy.readthedocs.io/en/latest/

88. Murdock P, Bassbouss L, Kraft A, Bauer M, Logvinov O, Ben Alaya M, et al. “Semantic Interoperability for the Web of Things”, White Paper. 2016. Available: https://www.researchgate.net/publication/307122744_Semantic_Interoperability_for_the_Web_of_Things

89. ETSI. SmartM2M; Plugtests™ preparation on Semantic Interoperability. 2019. Report No.: TR 103 537.

Glossary and Acronym Table

Table 8 Glossary and acronym table

AAL	Ambient Assisted Living
AIOTI	Alliance for IoT Innovation
API	Application Programming Interface
AQI	Air Quality Index
BAN	Body Area Network
DL	Description Logics
DSF	Demand Side Flexibility

ETSI	European Telecommunication Standards Institute
HTTP	HyperText Transfer Protocol
Interoperability	Interoperability is the ability of two systems to exchange information and utilize it.
IoT	Internet of Things
IRI	Internationalized Resource Identifier
IT	Information Technology
JPA	Java Persistence API
JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation for Linked Data
KAT	Knowledge Acquisition Toolkit . Reasoning tool
LINQ	Language-Integrated Query
LOV	Linked Open Vocabularies Ontology catalog
LOV4IoT	Linked Open Vocabularies for Internet of Things Catalog of ontology-based projects relevant for IoT.
NGSI	Next Generation Service Interface
ODP	Ontology Design Patterns
OLGA	Ontology Library GenerAtor
Ontology	Formal specification of concepts and their relationshionships to describe a specific area. This is sometimes referred to as ontology model or T-Box.
Ontology instance	Instantiation of ontology concepts and relationships. This is the actual information using the ontology's concepts and relationships. This is sometimes referred to as semantic dataset or A-Box.
ORM	Object Relational Mappers

ORSD	Ontology Requirement Specification Document It identifies the intended uses of the ontology, the end users, and the requirements the ontology should fulfill.
OWL	Web Ontology Language Formal language for the description of an ontology
RDF	Resource Description Framework. The basic semantic web language to describe triples.
RDF graph	Exchangeable representation (also called serialised representation) of an ontology instance, based on RDF.
RDFS	Resource Description Framework Schema. Set of classes with certain properties (e.g., subclassOf) using RDF for providing basic elements for the description of ontologies.
SAREF	Smart Applications REFerence ontology SAREF for Energy is SAREF extension for the Energy domain
Semantics	Semantics is the study of meaning. In the context of this work, the idea is to uniquely identify and agree on concepts and their relationships.This can be formally specified in an ontology.
S-LOR	Sensor-based Linked Open Rules
SOSA	Sensor, Observation, Sample, and Actuator
SPARQL	Recursive acronym (SPARQL Protocol and RDF Query Language). A query language to query semantic datasets based on RDF graphs.
SSN	Semantic Sensor Network
Triple	Triples have the form of <subject, predicate, object> and are used to describe ontology elements
W3C	World Wide Web Consortium
WoT	Web of Things
WSN	Wireless Sensor Network

UML	Unified Modeling Language
XML	eXtensible Markup Language

Acknowledgements

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreements No.732240 (SynchroniCity) and No. 688467 (VICINITY); from ETSI under Specialist Task Forces 534, 556, 566 and 578.
This work is partially funded by Hazards SEES NSF Award EAR 1520870, and KHealth NIH 1 R01 HD087132-01. The opinions expressed are those of the authors and do not reflect those of the sponsors.