WoO - Web of Objects Project
D.3.2 Domain model

Contract Number:	**ITEA-2 10028**		
Starting date:	**01/01/2012**	Ending date:	**31/12/2014**

Title of the main deliverable:	**Domain Model :** Conceptual model of system entities, their responsibilities and relationships
Title of the main deliverable:	D3.2 Domain Model
Work Package deliverable:	**D3.2**
Task related to the deliverable:	**T3.2**
Type (Internal / Restricted / Public):	

LEADER	CONTRIBUTORS:
UPE LISSI	
REVIEWERS	

HISTORY

NB: a status is associated to each step of the document lifecycle:

- **Draft:** this version is under development by one or several partner(s);
- **Under review:** this version has been sent for review;
- **Issued:** this version of the document has been submitted to EC.

Version	Status	Date	Author	Main Changes
0.1	Draft	14/09/2012	Abdelghani Chibani	First draft
~~0.2~~	~~Draft~~	~~13/11/2012~~	~~Abdelghani Chibani~~	~~update~~
0.3	Draft	01/02/2013	Vicente Hernández y Alexandra Cuerva	update
0.4	Draft	15/03/2013	Abdelghani Chibani,	update
0.5	Draft	23/04/2013	Abdelghani Chibani, Mihaela Brut	Update, review

TABLE OF CONTENTS

1. References

1.1. List of acronyms

1.2. List of Tables

1.3. List of Figures

1.4. Referenced documents

1.

2. Introduction

A semantic domain model is created in order to annotate semantically the key concepts, and the domain-vocabulary of the system being modeled and implemented. The semantic model identifies the relationships among all major entities within the system and describes their purpose and major attributes. The domain model provides a structural view of the system.

An important benefit of a semantic model model is that it describes and constrains the system scope. The domain model can be effectively used to verify and validate the understanding of the problem domain among various stakeholders of the project group. It is especially helpful as a communication tool and a focusing point between people working on different levels of technical abstraction.

2.1. Objective of the deliverable

The purpose of the deliverable is to provide a semantic modelling framework of both devices and services.

Thus, the first task results in a set of models describing essential entities of the Web of Objects as well as their mutual interactions. Moreover, a glossary specifying the essential terms describing the Web of Objects as an infrastructure as well as its devices, services, capabilities, and measurements. Existing standards about semantic modelling of sensors will be taken into account, like the standardization initiative Sensor Web Enablement under the OGC umbrella or OntoSensor. These results provide the basis for the work on the reference architecture as well as for the work in the technical work packages.

In addition, this task will also model the services associated with the intelligent objects, based on the previously defined devices model.

A semantic representation of both functionality and interface of a service allow for application scenarios in the Web of Object by enabling features such as service discovery and validation of interface consistency.

The goal of this task is to create ontologies or extend existing ones and develop a concept to use this semantic representation in a Web of Objects.

Standard binary representation of ontologies, such as RDF(S), may be sufficient to describe static concepts, but in the case of the Web of Objects dynamic aspects also need to be considered.

Therefore, extending the representation model with n-ary ontologies may be required to model dynamic concepts such as self-* mechanisms (reconfiguration and adaptation).

….

2.2. Organization of the deliverable

This deliverable provides is organized as the following

- .

3. Data Structures And Syntactic Description

Introduction …

In this section we present the basic constructs for the low-level description of objects and their capabilities that are provided by the standards for data structures: XML Schema and JSON. We present the use of XML to describe a new specifications of sensors data reading using XMPP and an XML schema for sensor web enablement by the Open Geospatial Consortium (OGC).

4.2.1 JSON based Description of Objects and Data

JavaScript Programming Language (JSON) [1] is a lightweight data-interchange language, which is easy for humans to read and write.

JSON is a text notation that has all the advantages of XML [2], but is much better suited to data-interchange and has the benefits of interoperability and openness. JSON is much simpler than XML because JSON has a smaller grammar and maps more directly onto the data structures used in modern programming languages. However, JSON is not extensible because is not a document markup language, so it is no necessary to define new tags or attributes to represent data in it.

In addition, JSON is easier for human and machines to read and write than XML. Like XML, JSON can be used as an exchange format to enable users to move their data between similar applications. JSON is processed more easily because its structure is simpler

On the one hand, JSON needs much less specialized software than XML. For example, in JavaScript and Python languages the JSON notation is built into the programming language or in Java only is needed a small amount of JSON-specific code to work.

XML and JSON separate the data presentation from the data structure. JSON uses structures based on arrays and records meanwhile XML structures are based on elements, attributes, DTDs and other meta-structures. Therefore, this mapping used by XML can be more complicated than in JSON.

On the other hand, the right option if a data exchange format is needed is JSON. But if you need a document exchange format, a XML option is better.

Furthermore, the machines can parser and generate it easily. For instance the following JSON format can be generated easily by a temperature-sensing object:

```
{
   temperature_sensor_value: "X"
   temperature_sensor_unit: "X"
}
```

The grammar of JSON is formed by two universal data structures:

- A collection of name/value pairs. This is realized as an object. An object begins with "{" and finishes with "}". Each name is followed by ":" and the name/value pairs are separated by ",".

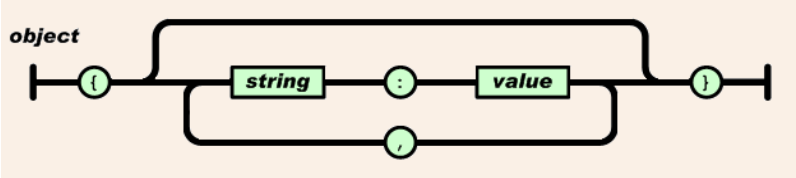


Figure: Object representation in JSON [1].

- An ordered list of values. This is realized as an array. An array is an ordered collection of values. It begins with "[" and ends with "]". Values can be a string, a number, true, false, null, an object or an array and they are separated by ",".

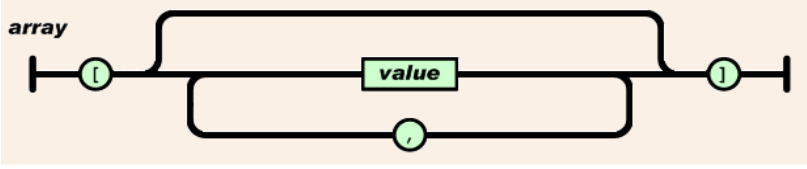


Figure: Array representation in JSON[1].

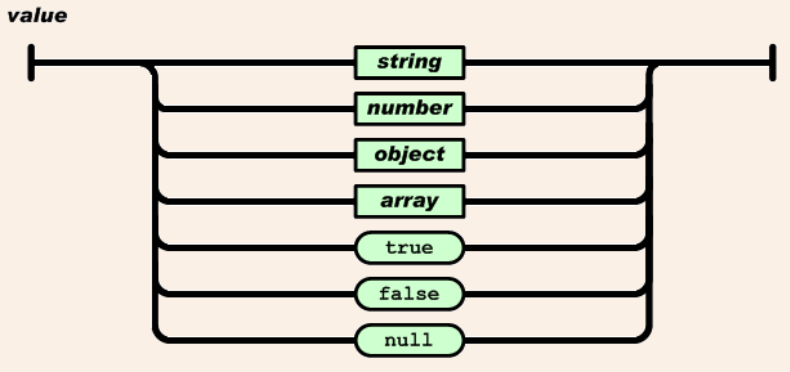


Figure: Value representation in JSON [1].

4.2.2 XMPP XEP-0323 Standard for Sensors Data Access and Representation

In (REF) Waher proposed a new for XMPP (XEP-0323) specification provides the common framework for sensor data interchange over XMPP networks. This Specification provides the underlying architecture, basic operations and data structures for sensor data communication over XMPP networks. It includes a hardware abstraction model, removing any technical detail implemented in underlying technologies.

Note has to be taken, that these XEP's are designed for implementation in sensors, many of which have very limited amount of memory (both RAM and ROM) or resources (processing power). Therefore, simplicity is of utmost importance. Furthermore, sensor networks can become huge, easily with millions of devices in peer-to-peer networks.

Sensor networks contains many different architectures and use cases. For this reason, the sensor network standards have been divided into multiple XEPs according to the following table:

The proposed specification uses the following terminology

- **Actuator**: Device containing at least one configurable property or output that can and should be controlled by some other entity or device.
- **Computed Value:** A value that is computed instead of measured.
- **Concentrator:** Device managing a set of devices which it publishes on the XMPP network.
- **Field:** One item of sensor data. Contains information about: Node, Field Name, Value, Precision, Unit, Value Type, Status, Timestamp, Localization information, etc. Fields should be unique within the triple (Node ID, Field Name, Timestamp).

- **Field Name:** Name of a field of sensor data. Examples: Energy, Volume, Flow, Power, etc.
- **Field Type:** What type of value the field represents. Examples: Momentary Value, Status Value, Identification Value, Calculated Value, Peak Value, Historical Value, etc.
- **Historical Value:** A value stored in memory from a previous timestamp.
- **Identification Value:** A value that can be used for identification. (Serial numbers, meter IDs, locations, names, etc.)
- **Localization information:** Optional information for a field, allowing the sensor to control how the information should be presented to human viewers.
- **Meter:** A device possible containing multiple sensors, used in metering applications. Examples: Electricity meter, Water Meter, Heat Meter, Cooling Meter, etc.
- **Momentary Value:** A momentary value represents a value measured at the time of the read-out.
- **Node:** Graphs contain nodes and edges between nodes. In Sensor Networks, sensors, actuators, meters, devices, gatewats, etc., are often depicted as nodes and links between sensors (friendships) are depicted as edges. In abstract terms, it's easier to talk about a Node, than have to list different types of nodes possible (sensors, actuators, meters, devices, gateways, etc.). Each Node has a Node ID.
- **Node ID:** An ID uniquelly identifying a node within its corresponding context. If a globally unique ID is desired, an architecture should be used using a universally accepted ID scheme.
- **Parameter:** Readable and/or writable property on a node/device. The XEP xep-0000-SN-Concentrators deals with reading and writing parameters on nodes/devices. Fields are not parameters, and parameters are not fields.
- **Peak Value:** A maximum or minimum value during a given period.
- **Precision:** In physics, precision determines the number of digits of precision. In sensor networks however, this definition is not easily applicable. Instead, precision determines, for example, the number of decimals of precision, or power of precision. Example: 123.200 MWh contains 3 decimals of precision. All entities parsing and delivering field information in sensor networks should always retain the number of decimals in a message.
- **Sensor:** Device measuring at least one digital value (0 or 1) or analog value (value with precision and physical unit). Examples: Temperature sensor, pressure sensor, etc. Sensor values are reported as fields during read-out. Each sensor has a unique Node ID.
- **SN:** Sensor Network. A network consisting, but not limited to sensors, where transport and use of sensor data is of primary concern. A sensor network may contain actuators, network applications, monitors, services, etc.

- **Status Value:** A value displaying status information about something.
- **Timestamp:** Timestamp of value, when the value was sampled or recorded.
- **Token:** A client, device or user can get a token from a provisioning server. These tokens can be included in requests to other entities in the network, so these entities can validate access rights with the provisioning server.
- **Unit:** Physical unit of value. Example: MWh, l/s, etc.
- **Value:** A field value.
- **Value Status:** Status of field value. Contains important status information for Quality of Service purposes. Examples: Ok, Error, Warning, Time Shifted, Missing, Signed, etc.
- **Value Type:** Can be numeric, string, boolean, Date & Time, Time Span or Enumeration.
- **WSN:** Wireless Sensor Network, a sensor-network including wireless devices.
- **XMPP Client:** Application connected to an XMPP network, having a JID. Note that sensors, as well as applications requesting sensor data can be XMPP clients.

An example of Sensor Data Representation is given in the following listing.

```
<iq type='get'
       from='master@xmpp-domain.com/amr'
       to='device@xmpp-domain.com'
       id='6'>
      <req xmlns='urn:xmpp:sn' seqnr='6' momentary='true'>
         <node nodeId='Device04'/>
         <field name='Energy'/>
         <field name='Power'/>
      </req>
   </iq>

   <iq type='result'
       from='device@xmpp-domain.com'
       to='master@xmpp-domain.com/amr'
       id='6'>
      <accepted xmlns='urn:xmpp:sn' seqnr='6'/>
   </iq>

   <message from='device@xmpp-domain.com'
            to='master@xmpp-domain.com/amr'>
      <fields xmlns='urn:xmpp:sn' seqnr='6' done='true'>
         <node nodeId='Device04'>
            <timestamp value='2013-03-07T22:03:15'>
                     <numeric  name='Energy'  momentary='true'  automaticReadout='true'
value='12345.67' unit='MWh'/>
                     <numeric  name='Power'  momentary='true'  automaticReadout='true'
value='239.4' unit='W'/>
            </timestamp>
         </node>
      </fields>
   </message>
```

For each object supporting the the XEP specification it must advertise its capabilities by returning a feature of "urn:xmpp:sn" when it receives the message type “services discovery

information requests". An example of request structure and its response given in the following listing.

```
<iq type='get'
    from='master@xmpp-domain.com/amr'
    to='device@xmpp-domain.com'
    id='disco1'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
     <!-- Services discovery information request !-->

<iq type='result'
    from='device@xmpp-domain.com'
    to='master@xmpp-domain.com/amr'
    id='disco1'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    ...
    <feature var='urn:xmpp:sn'/>
    ...
  </query>
</iq>
     <!-- Services discovery information response !-->
```

Service discovery information request and its corresponding response

4.2.3 XML Schema for Sensor Web Enablement

Open Geospatial Consortium (OGC™) has built a framework called Sensor Web Enablement (SWE) that refers to web accessible sensor networks and archived sensor data. It can be accessed using standard protocols and application program interfaces (APIs) [3].

SWE is composed by three languages:

1. Sensor Model Language (SML): Standard models and XML Schema for describing sensors systems and processes; provides information needed for discovery of sensors, location of sensor observations, processing of low-level sensor observations, and listing of task able properties [4].
2. Observation and Measurements (O&M): Standard models and XML Schema for encoding observations and measurements from a sensor, both archived and real-time [5].
3. Transducer Markup Language (TML): Describe any transducer (sensor or transmitter) in terms of a common model, including characterizing not only the data but XML formed metadata describing the system producing that data [6] [7].

Moreover SWE has three services specifications:

1. Sensor Observation Service (SOS): Includes three core operations: GetObservation, DescribeSensor, and GetCapabilities. The GetObservation operation provides an interface to query over observation data and returns an O&M document. The DescribeSensor operation provides an interface to query for the description of a sensor and returns a SML document. The GetCapabilities operation provides an interface to query for the description of a Sensor Observation Service. GetCapabilities allows clients to retrieve service metadata about a specific service instance and returns a GetCapabilites response document [8].
2. Sensor Planning Service (SPS): Offers a standardized interface for the control of sensors and simulations [9].
3. Sensor Alert Service (SAS): Dispatches alerts via XMPP (extensible messaging and presence protocol) [10].

Due to different levels of detail are required for modelling sensors, the features they observe, their capabilities, their environment and their conditions of use, four groups of use cases habe been defined.

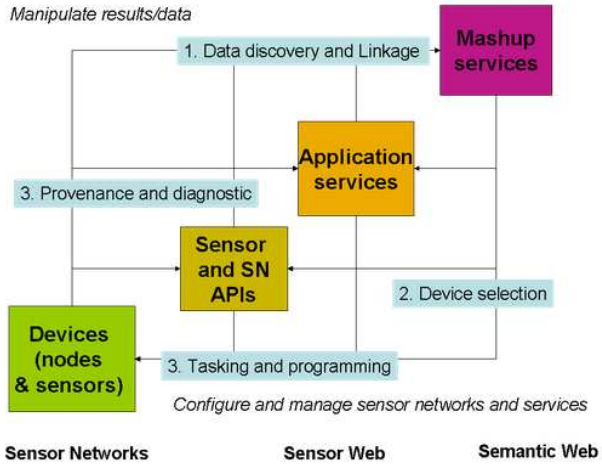


Figure: Technologies and use cases [3].

Data Discovery and Linking: users may need to find observations, obtained from the network, that meet certain criteria, and to link them to external data sources. Sensor and other data assets are described with respect to the SSN ontology. In this context, special care has been

taken to provide information about the related measurements and observations, with less attention paid to the descriptions of the sensor hardware from which these measurements and observations are made.

Device Discovery and Selection: this use case focus on with more emphasis on the structure and capabilities of the sensor or sensor networks than on how it relates to its environment. The ontology should capture information about sensor capabilities, performance, and the conditions in which it can be used. It should include the most common metrological definitions like accuracy, precision, drift, sensitivity, selectivity, measurement range, detection limit, response time, frequency and latency. In addition, other information such as the type of feature or phenomenon to be observed or the procedure to be followed is also needed.

Provenance and Diagnosis: In a sensor deploying is necessary to monitor their assets to detect the possible occurrences of faults and the degradation in the quality of measurement over time (drift), possibly caused by changes in the operating conditions. Provenance data is important for the users of sensor networks wishing to evaluate the fitness of the data for their purpose. The ontology needs to provide information about the sensor that is used to derive measurement data. Firstly, it needs to describe the physical property being observed, and properties of the sensor's measuring capability such as frequency, accuracy, measurement range, and precision. This can be used to check whether the sensor has been properly used to derive some subsequent conclusion, and could also be used to derive error bounds on derived results such as forecasts. Moreover, the ontology should describe the physical structure on which the sensor is deployed and its location, the custodian responsible for the sensor, the maintenance schedule for the device (e.g. how often is the sensor checked) and the environmental conditions under which the sensor is expected to operate according to specification. The custodian information and maintenance information can be used to determine trust in the data. The environmental conditions might be used to diagnose the reasons for an apparent data failure.

Device Operation Tasking and Programming: this use case refers to the assembly and transmission of instructions intended to control the behaviour of the sensing device. The main task is to control of sensing behaviour and actuation that is necessary to obtain the desired sensing data. The ontology should capture sufficient information about the capability of sensor devices and about the current state of the devices to support a palette of options for selection by the user. For example, a user should be able to define the desired observation frequency expressed in terms of the possible frequencies identified for the sensing device.

4. Ontology Based Modeling

Ontologies are expected to be used for the creation and sharing of structured vocabularies that explicate the relationships between different terms, allowing intelligent agents (and humans) to interpret their meaning flexibly yet unambiguously. Ontology languages allow users to write explicit, formal conceptualizations of domains models using a well-defined syntax, a well-defined semantics, an efficient reasoning support and a sufficient expressive power.

Usually Ontology is composed of (some of) the following items: classes of objects, a vocabulary of terms (instances), and various relations between terms and classes. Several projects were undertaken in this decade to make progress in ontology definition and ontology languages in different applications spanning several domains. For that reason, three ontology languages have been standardized RDF, RDFS and OWL in the context of what is called the semantic web stack.

In this section we provide an present the semantic web stack, ontology languages and domain ontologies that are relevant for the application domain and issues addressed in the topic of the web of objects. We conclude this section by a presentation of the Micro-Concept language, a variante of the OWL 2 DL that allows for handling reasoning issues addressed in data base application and business processing management systems.

4.1. Semantic modelling foundations of the semantic web

The Semantic Web introduces semantic technologies, like ontologies and taxonomies, to enrich the computer with the ability to understand the meaning of texts or images. These technologies try to structure the web and the included content to make it machine-understandable. The well-known Semantic Web Stack, originally introduced by Berners-Lee and his colleagues in their seminal paper on “Scientific American” (2001), is reproduced in Figure 1. It consists of a sequence of layers, where each layer exploits and uses the capabilities of the previous layers. It shows how the technologies that have been standardized for the Semantic Web are organized to make the Semantic Web possible. The technologies going from the bottom of the stack up to OWL are currently standardized and they are currently used to develop Semantic Web applications.

The bottom layers concern technologies that are well known from hypertext web and that, without change, can offer support for building up the semantic web.

- Internationalized Resource Identifier (IRI) provides means for uniquely identifying semantic web resources. Semantic Web needs unique identification to allow provable manipulation with resources in the top layers.

- Unicode serves to represent and manipulate text in many languages. Semantic Web should also help to bridge documents in different human languages, so it should be able to represent them.
- XML is a markup language that enables creation of documents composed of structured data. Semantic web gives meaning (semantics) to structured data.
- XML Namespaces provides a way to use markups from more sources. Semantic Web is about connecting data together; therefore, we need to refer more sources in one document.

Middle layers contain technologies standardized by W3C to enable building semantic web applications.

- Resource Description Framework (RDF) is a framework for creating statements in ~~the~~ form of so-called triples. It enables us to represent information about resources in the form of graphs - the semantic web is sometimes called Giant Global Graph.
- RDF Schema (RDFS) provides basic vocabulary for RDF. Using RDFS it is, for example, possible to create hierarchies of classes and properties.
- Web Ontology Language (OWL) extends RDFS by adding more advanced constructs to describe the semantics of RDF statements. It allows us stating additional constraints, such as for example cardinality, restrictions of values, or characteristics of properties such as transitivity. It is based on description logic and so brings some reasoning power to the semantic web.
- Cryptography is important to ensure and verify that semantic web statements are coming from trusted source. This can be achieved by appropriate digital signature of RDF statements.

- Trust to derived statements will be supported by (a) verifying that the premises come from trusted source and by (b) relying on formal logic during deriving new information.

xxxx

Figure 1: Semantic Web Stack

4.2. Semantic Web Ontology Languages

4.2.1. RDF/RDFS

RDF is a W3C (World Wide Web Committee) recommendation. Its motivation is to provide a standard for the meta-data to be used for setting up descriptions about resources on the web. However, the distinction between data and meta-data is a particularly complex one: RDF, considered as a framework for describing data about web-resources, is also capable to represent data. The basic construction in RDF is an object-attribute-value triple: an object O

has an attribute A with value V. Such a triple corresponds to the relation that is commonly written as A (O, V). RDFS is the schema language for RDF. But, first of all, what is a schema language? There are a number of successful schema languages for familiar technologies, but the role that each of these languages plays in the management of information is closely linked to the particular language or system.

There's a lot more into the RDF Schema Language (RDFS) than just the type propagation rule of rdfs:subClassOf. RDFS consists of a small number of inference patterns, each one of which can provide type information for individuals in a variety of circumstances.

The modeling primitives offered by RDF are very basic. Therefore, the RDF Schema specification defines further modeling primitives in RDF. Examples are class/subclass relationship, domain and range restrictions for property, and sub-property. Despite the similarity in their names, RDF Schema fulfills a different role than XML Schema does. XML Schema, and also DTDs, prescribes the order and combination of tags in an XML document. In contrast, RDF Schema only provides information about the interpretation of the statements given in an RDF data model, but it does not constrain the syntactical appearance of an RDF description.

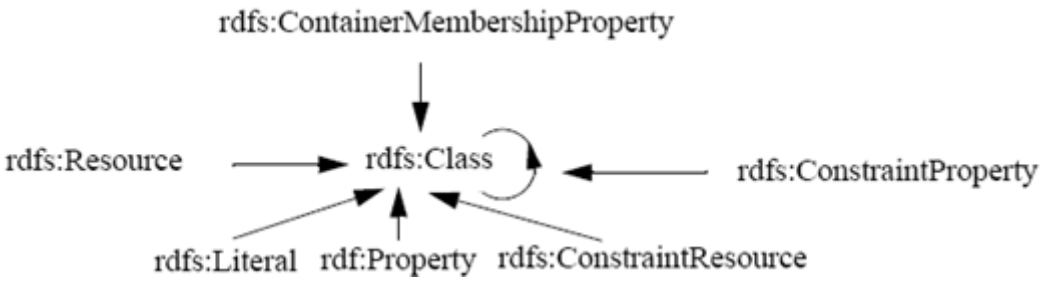


Figure 2: RDFS constructs

However a number of other features are missing:

- Local scope of properties: rdfs:range defines the range of a property, say \eats", for all classes. Thus in RDF Schema we cannot declare range restrictions that apply to some classes only. For example, we cannot say that cows eat only plants, while other animals may eat meat, too.
- Disjointness of classes: Sometimes, we wish to say that classes are disjoint. For example, male and female are disjoint. But in RDF Schema we can only state subclass relationships, e.g. female is a subclass of person.
- Boolean combinations of classes: Sometimes, we wish to build new classes by combining other classes using union, intersection and complement. For example, we

may wish to define the class person to be the disjoint union of the classes male and female. RDF Schema does not allow such definitions.

- Cardinality restrictions: Sometimes, we wish to place restrictions on how many distinct values a property may or must take. For example, we would like to say that a person has exactly two parents, and that a course is taught by at least one lecturer. Again such restrictions are impossible to express in RDF Schema.
- Special characteristics of properties: Sometimes, it is useful to say that a property is transitive (like \greater than"), unique (like \is mother of"), or the inverse of another property (like \eats" and \is eaten by")

Therefore, we need an ontology language that is richer than RDF Schema, a language that offers these features and more. In designing such a language, one should be aware of the tradeoff between expressive power and efficient reasoning support.

Generally speaking, the richer the language is, the more inefficient the reasoning support becomes, often crossing the border of non-computability. Thus we need a compromise, a language that can be supported by reasonably efficient reasoners, while being sufficiently expressive to express large classes of ontologies and knowledge.

To solve the lack of formalization in RDF and RDF-S and augment the expressivity and reasoning power of DAML-OIL, the W3C's Web Ontology Working Group recommended the use OWL a standard towards fulfilling different aspects of ontology engineering and application.

4.2.2. OWL

The Web Ontology Language (OWL) is an XML-based language designed for use by applications that need to process the content of information in a form compatible with the architecture of the World Wide Web, instead of just presenting information to specific user communities. OWL facilitates greater machine interpretability of Web content than that supported by other W3C standards such as XML, RDF, and RDF Schema by providing additional vocabulary along with a formal semantics. OWL is inspired from Description Logics DL and DAML+OIL, OWL does not only enable knowledge representation as web based Meta data, but also (partial) reasoning on this knowledge. Built on top of RDF and DLSH, constructor, OWL is articulated around three main modeling entities: the class (also called concept), the property (also called role) and the individual of a class (also called instance of a class). Using these atomic entities, it is possible to build complex classes.

OWL was based on SHOIN only simple role hierarchy, and unqualified NRs

OWL 2 (like OWL, DAML+OIL & OIL) based on DL SROIQ (i.e., ALC extended with transitive roles, a role box nominals, inverse roles and qualified number restrictions).

• OWL 2 EL based on EL

• OWL 2 QL based on DL-Lite

• OWL 2 EL based on DLP

OWL Constructor	DL Syntax	Example
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	Human ⊓ Male
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	Doctor ⊔ Lawyer
complementOf	$\neg C$	¬Male
oneOf	$\{x_1\} \sqcup \ldots \sqcup \{x_n\}$	{john} ⊔ {mary}
allValuesFrom	$\forall P.C$	∀hasChild.Doctor
someValuesFrom	$\exists P.C$	∃hasChild.Lawyer
maxCardinality	$\leqslant nP$	⩽1hasChild
minCardinality	$\geqslant nP$	⩾2hasChild

The open-world assumption is distinguishing features of Description Logics with respect to the modeling languages developed in the study of databases. The semantics of ABoxes is an "open-world semantics," since normally knowledge representation systems are applied in situations where one cannot assume that the knowledge in the KB is complete (i.e. absence of information in an ABox only indicates lack of knowledge).

OWL DL reasoning is built on top of DL reasoning operations. We can distinguish here three levels of reasoning:

- Classes: It consists of a subsume tests that allows us to build up a full taxonomy of ontology classes.
- Properties: It consists of a relationship inference between instances of various classes making use of OWL properties like transitive or functional property. For example, this type of reasoning can be used to determine through transitivity the existence in an ontology of a relation between instances of two different classes that do not have an explicit relation within this ontology.
- Instances (individuals): It consists of two tests: i) Consistency Check test, i.e. to check if a class has an instance in the knowledge, ii) verifying in the knowledge base the classes that correspond to a full or partial description of a given instance.

OWL constructors and reasoning capabilities can be used to give a full semantic representation and reasoning on context. The reasoning on classes and properties levels is necessary to build a complete taxonomy of context attributes. This taxonomy makes it possible to check the validity of contextual facts and how to extend context ontology. The reasoning on individual level can be used during execution run time of agents to allow them discovering agent services, which are necessary to process contextual knowledge inferences rules.

Figure 2. OWL DL Ontology of objects.

In addition to OWL native reasoning capabilities, it is also possible to carry out FOL inferences on top of properties instances in the same way as predicate based inference. The properties are thus integrated in inference rules following this form: If AND/OR {properties (xi, yi)} then AND/OR {properties (xj, yj)}. For that purpose SWRL (Semantic Web Rule Language) was proposed as W3C standard proposal for (FOL based) OWL ontology rules language It allows then to express Horn like rules by using the syntax of OWL and Rule ML. While Protégé 3 ontology editor integrates a plugin for SWRL rules, inference engines supporting SWRL reasoning like KAON are still in the beta stage. Thus to implement SWRL reasoning, we have to translate rules and ontologies expressed into OWL/ SWRL to first order logic formulas in the style of the CLIPS rules to be processed then with JESS inference engine.

4.2.3. RDF, OWL Comparison

	RDF	OWL 1 & 2
Semantic Expressiveness	++	+++
Reasoning Operations	+	+++
Inference ability	++	+++
Reuse	+++	+++
Possibility of matching	+	+++
Constraints	+	+++
Formal Logics Compliancy	+	+++
Unique Name Assumption	-	-

Table 1: RDF, OWL COMPARISON

+++: very good

++: good

+: medium

- : not supported

The owl language can be considered as an interesting solution to model a security policy and a user profile.

Along with the properties mentioned in the table below, OWL's use has many advantages such as:

- Interoperability: OWL offers properties allowing us to compare properties and classes: equivalence, symmetry, transitivity properties between classes.
- OWL allows also us to describe classes as intersection, union and complement of other classes.
- Conflict resolution: thanks to disjointness property.
- Extensibility: it is always possible to enrich the ontology classes and to create new individuals and new properties. It makes also possible the support of changes given that it is possible to add or to delete classes.
- Constraints management: There are restriction properties in owl that allow constraints description: value restriction and cardinality restriction.
- Abstraction: OWL full is the unique version of owl that allows a class to be an instance of another class (Meta-class). This property offers a high level of abstraction for owl, which make it possible to create ontology that can be common to many organizations.
- Flexibility: There are multiple categories for ontology: RDF, RDFS, OWL light, OWL DL and OWL full. Thus, an ontology designer can opt for the ontology the most suitable to the organization requirements.

4.2.4. Querying Ontology and Knowledge in the Semantic Web

The most popular query languages that are used to extract knowledge from RDF knowledge bases are mainly based on RDF Graphs RDF-Query and SPARQL. RDF-Query is supported by various implementations in Java, PHP, Perl.

SPARQL [12] is the W3C recommended querying language for RDF and OWL databases. It has been designed for querying the RDF based data on the web. SPARQL aims to let people work upon the data without bothering them about its internal organization. Therefore, SPARQL enables people to share, merge, and reuse data globally, fulfilling the goal of Semantic Web. There are different SPARQL variants:

- *nSPARQL:* is a navigational language for RDF that uses nested regular expressions to enhance the expressiveness of SPARQL. nSPARQL defines the semantics of SPARQL over RDFS.
- *SPARQL-ST*: introduces additional syntaxes for spatial type and a temporal type.

- *iSPARQL:* extension of conventional SPARQL to analyse the data during query processing supporting the User Defined Similarity Function (UDSF).
- *SPARQL++*: provides aggregate functions, value generating build-ins and recursive processing of mappings.
- *PSPARQL:* provides queries on path expressions which are made from regular expression patterns.
- *SPARQL-DL*: resolves the compatibility issue between SPARQL queries and OWL-DL databases.

Unlike RDF based query languages, OWL ontologies querying can be done by SWRL reasoning rules or OWL Query Languages. RQL (Racer Query Language) is another query language, with its proper syntax, based on description logics. This language makes it possible to extract information from a knowledge base represented in OWL-DL and enable DL inferences. OWL-QL is another example of query language based on OWL, which provides software agents capabilities to automatically share their knowledge. OWL-QL specifies the semantic relations between a request, the response and the knowledge base containing the elements of the answer.

For developing semantic web applications based on W3C recommendations for RDF and OWL we propose to use Jena, a Java framework that provides extensive Java libraries for helping developers. Jena handles RDF, OWL and SPARQL in line with published W3C recommendations. Furthermore, Jena includes a rule-based inference engine to perform reasoning based on OWL and RDFS ontologies, and a variety of storage strategies to store RDF triples in memory or on disk [13].

4.2.5. Micro-Concept Ontology Language

The Micro-Concept model is designed to describe an environment where heterogeneous objects can evolve over the time and interact with each other, in particular in the context of the internet of objects. The goal is also to provide a model that can support high efficiency reasoning based on production inference rules.

Micro-concepts define groups of objects sharing common characteristics. In order to describe them, we use properties, which are relations between a micro-concept, and either another micro-concept or a literal value (an integer or a string for instance). Those relations are defined independently of micro-concept, and may symbolize a relation between any kinds of micro-concepts. Each micro-concept can define actions that each instance (or concrete object) belonging to this micro-concept will be able to execute. This model can also be used to

describe the instances of micro-concept, describing it by giving concrete properties values for this instance.

Micro-Concept language objects span from imported ontologies, concepts, actions to instances, enumerations and enumeration items. All these objects must be identified with a unique identifier, which is a URI specified according to the [RFC3986] .

In Micro-Concept model, the UNA (Unique Name Assumption) is used: two elements with different identifiers are considered as different. Unlike OWL or RDF-S, this property allow to link a conceptual representation of an entity to the source in the real world such as a device, sensor or web service in case the entity is immaterial.

Properties of Micro-Concepts can have Default values. These values can be modified without limitation. Previous values can be removed; new values can be added as default values. The given default values must be compatible with the properties restrictions defined for the current concept.

Besides the formal representation of the concepts, the ontology defined with this standard must be understandable by business users. For this purpose, we describe a mechanism to annotate the concepts, properties, actions and instances with multi-lingual labels and descriptions. This supposes that every concept described in a single context is completely understandable in any language as long as we stay in this context.

In the domain of semantics and in particular the semantic web, Resource Description Framework [RDF] and RDF Schema [RDF-S] are considered as reference standards. Considering this and the possibility of extension and interoperability of RDF/RDFS, we choose to build the Micro-Concept model on top of these standards. As a result, our model is semantically and syntactically compatible with RDFS, the only exception being that instances and classes are disjoint (an element must not be simultaneously an instance and a concept). Besides this point, an RDFS document must directly be used as a micro-concept model. It may be extended using some attributes specific to the micro-concept model.

There is an important overlap between OWL 2 and Microncept. Indeed, most of the constructs provided in OWL 2 are reused in Microncept. The difference lie in the fact that OWL formalization is based on Description Logic while Micro Concept is closer to conceptual models such as Entity/Relationship model. Current efforts are done to render Micro Concept compliant with Description Logic to allow consistency checking and concepts reasoning operations.

4.2.6. Universal Upper ontologies

We will take advantage of existing upper level ontologies and general purpose domain ontologies to reuse if necessary their concepts to build the foundation of the Web of Object Domain Ontology. These ontologies will have high-level concepts that are linked to those used in the domain ontologies. A top-level ontological framework provides support to make ontological decisions as transparent as possible in the resulting domain ontolog [REF]. There exist several ontologies of this kind; the main difference between them is the criteria used to classify general concepts.

Examples of the upper ontologies that can be of particular interest in the context of this project are:

• OpenCyc (http://www.cyc.com/platform/opencyc), is the open source version of the world's largest and most complete general knowledge and commonsense reasoning engine Cyc [xx] . The release 4 of OpenCyc ontology whose domain is all of human consensus reality, is available for free in the sourceforge web platform under the Opencyc License. It is based on Cyc's microtheories and contains around Cyc 239.000 terms, 2.093.000 triples and 69,000 links to external semantic web data namespaces using owl:sameAs property.

• Suggested Upper Merged Ontology (SUMO) developed by the Working Group entitled "Standard Upper Ontology" funded by IEEE [11]. SUMO provides general concepts that are used by several domain ontologies. It has been created by merging different available ontological content sources like the Sowa upper ontology, ontologies available on the Ontoligua server, CNR's group mereotopology, and so on. Therefore, this ontology contains, for instance, temporal concepts such as time duration, part and wholes relations, and semiotic relations. The knowledge representation language used is SUO-KIF, a variation of KIF (Knowledge Interchange Format);

•YAGO, [xx], is a new upper ontology built from the widely used lexicons web platforms Wikipedia and Wordnet. It is composed of more than 10 million entities (like persons, organizations, cities, etc.) and contains more than 120 million facts about these entities. Yago is intended to be used as the universal taxonomy and multi thematic ontology for any semantic web based application. In ([xx]) the authors reported an attempt of integrating YAGO in SUMO allow for incorporating millions of entities to enable automated processing and advanced reasoning that can be valuable for large scale networking applications such as robots indoor and outdoor semantic navigation or cloud robotics. Yago-SUMO can be considered as

universal upper ontology of high quality, with confirmed accuracy around 95 percent where every relation is annotated with its confidence value and all the assertions are anchored in time and space.

DOLCE (a Descriptive Ontology for Linguistic and Cognitive Engineering). DOLCE aims at capturing the ontological categories underlying natural language and human commonsense. DOLCE is made of concepts that describe endurant objects, called also continuants, and perdurant objects, called also things that occur in time such as events. Endurants and perdurants can be co-located in the same space-time. Compared to OpenCyc, DOLCE is considered as an upper ontology of instances with a taxonomy composed for example of the following conceptual categories : qualities, objects, features, substantials, aggregates, etc. To ensure an efficiently exploitation of DOLCE and render it more conceptually more rigorous, an attempt was aligning with WordNet, in the context of the OntoWordNet Project s. The DOLCE+DnS Ultra lite (DUL) version is a simplification and an improvement of some parts of the DOLCE. DUL provides a set of upper level concepts that can be the basis for easier interoperability among many middle and lower level ontologies. DUL only uses OWL-Lite and disjointness constraints. The upper ontologies that will comprise the foundation of our work, will be DOLCE DUL. . Dolce DUL provides upper concepts for sensing, navigation and spatio temporal respresentation that are mostly compliant with OpenCyc. We will consider also the YAGO and SUMO sine they started to be widely used in the community working on search computing issues. The Yago contains several terms and encompasses very important interconnexions with Wordnet

Matching ontologies or merging concepts from different upper level ontologies enables the interoperation of objects knowledge expressed using concepts belonging to several domain and upper ontologies. In this context, several attempts were reported in the state of the art. COSMO (COmmon Semantic Model) is a lattice of ontologies that integrates concepts from the OpenCyc and SUMO ontologies.

Besides considering upper level ontologies, we will investigate and attach a particular attention to well known domain ontologies, respectively for sensing and actuation, robotics and automation, web services, workflows and business process management domains. In the literature, we can find several examples in the literature, that will be evaluated and considered for incorporation in our project.

4.3 General Purpose Domain Ontologies

High quality domain ontologies are essential for successful employment of semantic Web services. However, their acquisition is difficult and costly, thus hampering the development of this field (REF).

4.3.1 Composite Capability/Preference Profiles (CC/PP)

As the number and variety of devices connected to the Internet grow, there is a corresponding increase in the need to deliver content that is tailored to the capabilities of different devices. Some limited techniques, such as HTTP 'accept' headers and HTML 'alt=' attributes, already exist. As part of a framework for content adaptation and contextualization, a general purpose profile format is required that can describe the capabilities of a user agent and preferences of its user. CC/PP is designed to be such a format.

CC/PP is based on RDF, the Resource Description Framework, which was designed by the W3C as a general purpose metadata description language. RDF provides the framework with the basic tools for both vocabulary extensibility, via XML namespaces [XMLNAMESPACES], and interoperability. RDF is a standard format for data interchange on the Web. There is a specification that describes the complete semantics of RDF [RDFSEMANTICS], as well as of the RDF Schema description language [RDFSCHEMA]. RDF can be serialized into XML using the RDF/XML format as defined in [RDFXML]. This serialization is used in this document, although other serializations of RDF could be permissible (like Turtle [TURTLE]). For an introduction to RDF, see also the RDF Primer [RDFPRIMER]. RDF is a natural choice for the CC/PP framework since user agent profiles are data intended primarily for communication between user agents and resource data providers.

A CC/PP profile contains a number of CC/PP attribute names and associated values that are used by a server to determine the most appropriate form of a resource to deliver to a client. It is structured to allow a client to describe its capabilities by reference to a standard profile, accessible to an origin server or other sender of resource data, and a smaller set of features that are in addition to or different than the standard profile. A set of CC/PP attribute names, permissible values and associated meanings constitute a CC/PP vocabulary.

Some information contained in a profile may be sensitive, and adequate trust and security mechanisms must be deployed to protect users' privacy. As a part of a wider application, CC/PP cannot fully cover such issues, but is intended to be used in conjunction with appropriate mechanisms. This topic is covered in Appendix F (CC/PP applications).

It is anticipated that different applications will use different vocabularies; indeed this is needed if application-specific properties are to be represented within the CC/PP framework. But for different applications to work together, some common vocabulary, or a method to convert between different vocabularies, is needed. (XML namespaces can ensure that different applications' names do not clash, but does not provide a common basis for exchanging information between different applications.) Any vocabulary that relates to the structure of a

CC/PP 2.0 profile must follow this specification. The appendices introduce a simple CC/PP attribute vocabulary that may be used to improve cross-application exchange of capability information, partly based on some earlier IETF work.

CC/PP 2.0 is designed to be broadly compatible with the UAProf 2 specification [UAPROF2] from the OMA (formerly known as WAP Forum), in the same way CC/PP 1.0 used to accommodate UAProf 1.0 profiles.

CC/PP is compatible with IETF media feature sets (CONNEG) [RFC2533] in the sense that all media feature tags and values can be expressed in CC/PP. However, not all CC/PP profiles can be expressed as media feature tags and values, and CC/PP does not attempt to express relationships between attributes.

Although the examples and use to date have been focused on device capabilities, CC/PP can also convey information about user preferences that, used sensibly, should be allow web servers to improve the accessibility of web sites. A fuller discussion of web site accessibility can be found in the Web Content Accessibility Guidelines [WAI].

4.3.2 Semantic Web Services

A major limitation of the Web services technology is that finding and composing services still requires manual effort. Web services technology do not include explicit descriptions of the functionality of a Web Service This becomes a serious burden with the increasing number of Web services. Moreover, the existing descriptions are represented syntactically and therefore do not depict the meaning of the information to be interchanged. Semantic Web Services (SWS) applies Semantic Web technology to Web Services raising the level of discourse. More specifically, through the use of exhaustive semantic description frameworks SWS will support the provision of intelligent mechanisms for the discovery, composition, contracting, and execution of Web Services.

SWSL, WSMO and OWL-S were the two major efforts for specifying semantic information for Web services in order to enable automatic service discovery, composition and execution. However, du to the complexity of defining and verifying the semantic correctness of services descriptions using these languages, new and simpler approaches (SA-WSDL and SA-REST) were proposed to define meta data on services grounding description to facilitate the mapping and search operations based on the concepts defining the terms used to label services description components. Indeed, there are substantial differences between these languages, see details in the appendix (cf. [xx]).

The creation of semantic Web service descriptions is a time consuming and complex task whose automation is desirable, as signaled by many researchers in this field, for example [59]. This task can be broken down in two smaller tasks. First, acquiring a suitable Web service domain ontology is a prerequisite when creating semantic Web service descriptions. This is required to create a shared terminology for the semantic service descriptions. Second, the

actual process of annotating Web services with the concepts provided by the domain ontology (i.e., creating their semantic descriptions) has to be performed. Several OWL sub-languages were proposed in this decade modeling semantic web services as solution to lack of semantic description of web services1.

WSMO

Web Service Modeling Ontology (WSMO) is an ontology that describes using formal semantics web services. Based on the Web Service Modeling Framework (WSMF) [xxx], WSMO develops a language called WSML to allows describing web services using : (1) ontologies which provide the concepts and relationships used by other elements, (2) goals that define the users' objectives, i.e. the (potential) problems that should be solved by Web Services, (3) Web Services descriptions that define various aspects of a Web Service, and (4) mediators which bypass interoperability problems.

OWL-S

OWL-S is an upper ontology used to describe the semantics of services based on the W3C standard ontology OWL and is grounded in WSDL. It has its roots in the DAML Service Ontology (DAML-S) released in 2001, and became a W3C candidate recommendation in 2005.

OWL-S facilitates the description of services in terms of four different ontologies. The upper Service ontology links to the Profile, Service Model and Grounding ontologies. The Profile defines some non-functional properties of the service and exposes some of the functionality by referencing Inputs, Preconditions and Results from the Service Model. The Service Model describes the behaviour of the Service in terms of processes and control constructs. The Grounding binds processes, inputs and outputs in the process model to some transport protocol described in a WSDL document. The major concrete differences are explained in the following paragraphs. The grounding of a given OWL-S service description provides a pragmatic binding between the logic-based and XMLS-based service definitions for the purpose of facilitating service execution. Such a grounding of OWL-S services can be, in principle, arbitrary but has been exemplified for a grounding in WSDL to pragmatically connect OWL-S to an existing Web Service standard

SWSL

[1] http://www.w3.org/2005/04/FSWS/Submissions/1/wsmo_position_paper.html

SWSL is another proposal for a language for describing Semantic Web Services. SWSL has two parts, a rules language and a process ontology. The SWSL-Rules sublanguage is closely related to WSML-Rules. Both languages are largely based on F-logic and they mostly share the logical expression syntax. However, the two groups have pursued complementary goals.

SAWSDL

The standard language WSDL for Web Services operates at the mere syntactic level as it lacks any declarative semantics needed to meaningfully represent and reason upon them by means of logical inferencing. In a first response to this problem, the W3C Working Group on Semantic Annotations for WSDL and XML Schema (SAWSDL) developed mechanisms with which semantic annotations can be added to WSDL components. Based on its predecessor and W3C member submission WSDL-S, the key design principles for SAWSDL are that (a) the specification enables semantic annotations of Web Services using and building on the existing extensibility framework of WSDL; (b) it is agnostic to semantic (ontology) representation languages; and (c) it enables semantic annotations for Web Services not only for discovering Web Services but also for invoking them. Based on these design principles, SAWSDL defines the following three new extensibility attributes to WSDL 2.0 elements for their semantic annotation:

• An extension attribute, named modelReference , to specify the association between a WSDL component and a concept in some semantic (domain) model. This modelReference attribute is used to annotate XML Schema complex type definitions, simple type definitions, element declarations, and attribute declarations as well as WSDL interfaces, operations, and faults. Each modelReference identifies the concept in a semantic model that describes the element to which it is attached.

• Two extension attributes (liftingSchemaMapping and loweringSchema- Mapping) are added to the set of XML Schema element declarations, complex type definitions and simple type definitions. Both allow to specify mappings between semantic data in the domain referenced by Model Reference and XML, which can be used during service invocation.

SAREST

... TBD ..

4.3.3 Ontologies for Semantic Sensors and Actuators Network

To achieve that our Wireless Sensor and Actuators Network (WSAN) can be integrated into the WoO framework we need standards and computing reasoning including different languages for

describing the capabilities and measurements of different sensors and actuators and other important characteristics of the environment around us. Hence, we can obtain a totally semantic interoperability and integration between all technologies used in WoO.

With respect to Sensors domain modelling, Compton et al. [24] provided a survey of sensor ontologies, for instance, OntoSensor [23] that contains descriptions of properties common to several sensors while the SSN ontology, [xx], is one of the successful project of proposing a W3C standard using DOLCE DUL and OWL 2 representation.

The main tasks of a semantic sensor network are: the organization, installation, management, understanding and control the resulting data using high-level specifications. To implement a sensor ontology we need to take into account the limited power availability, limited memory, variable data quality and loose connectivity o the sensors/actuators [3].

4.3.3.1 The Semantic Sensor Network ontology (SSN)

Developed by the OGC group, the SSN ontology provide a framework for describing sensors. It contains 41 concepts and 39 object properties that allow for describing the capabilities of sensors, the act of sensing and the resulting observations. This consensual ontology is the result of more than four year of group discussion and votes.

The ontology is based around concepts of systems, processes, and observations. It supports the description of the physical and processing structure of sensors. Sensors are not constrained to physical sensing devices: rather a sensor is anything that can estimate or calculate the value of a phenomenon, so a device or computational process or combination could play the role of a sensor. The representation of a sensor in the ontology links together what it measures (the domain phenomena), the physical sensor (the device) and its functions and processing (the models). The design of the Semantic Sensor Network Ontology is the result of two iterations:

- Phase 1: development of the ontology modules and examples.
- Phase 2: alignment to the DOLCE Ultra Lite (DUL) upper ontology.

The SSN ontology can be used for a focus on any (or a combination) of a number of perspectives:

- A sensor perspective, with a focus on what senses, how it senses, and what is sensed.
- A data or observation perspective, with a focus on observations and related metadata.

- A system perspective, with a focus on systems of sensors.
- A feature and property perspective, with a focus on features, properties of them, and what can sense those properties.

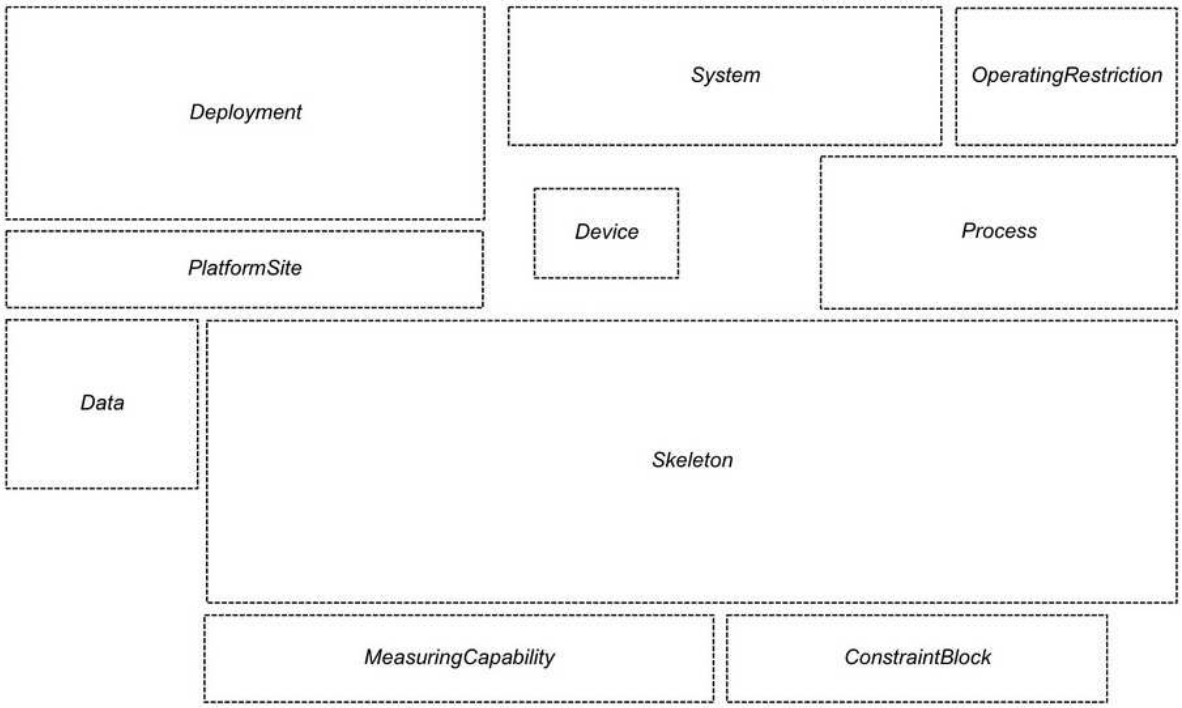


Figure: Overview of the Semantic Sensor Network ontology modules [3].

The ontology does not include a hierarchy of sensor types and the different modules contain the classes and properties that can be used to represent particular aspects of a sensor or its observations.

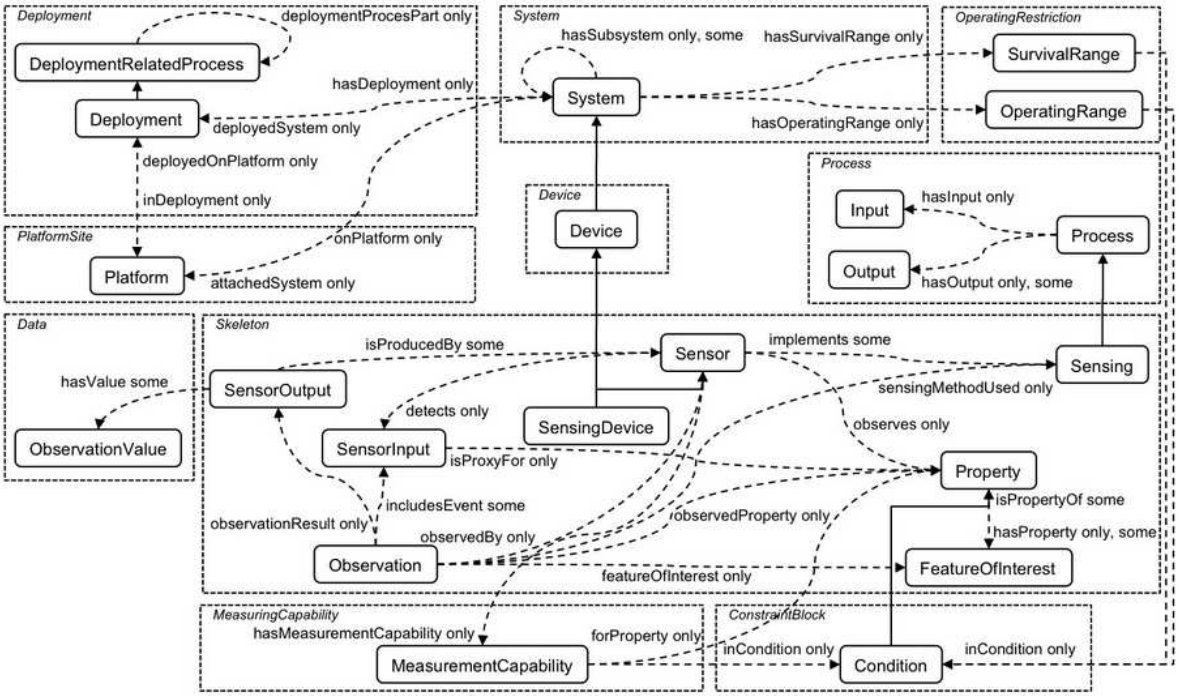


Figure: Overview of the Semantic Sensor Network ontology classes and properties [3].

The following figure presents the Enumeration of the measurement, environmental and survival properties. For example: sensors, observations, features of interest, the process of sensing (i.e. how a sensor operates and observes), how sensors are deployed or attached to platforms, the measuring capabilities of sensors, as well as their environmental, and survival properties of sensors in particular environments.

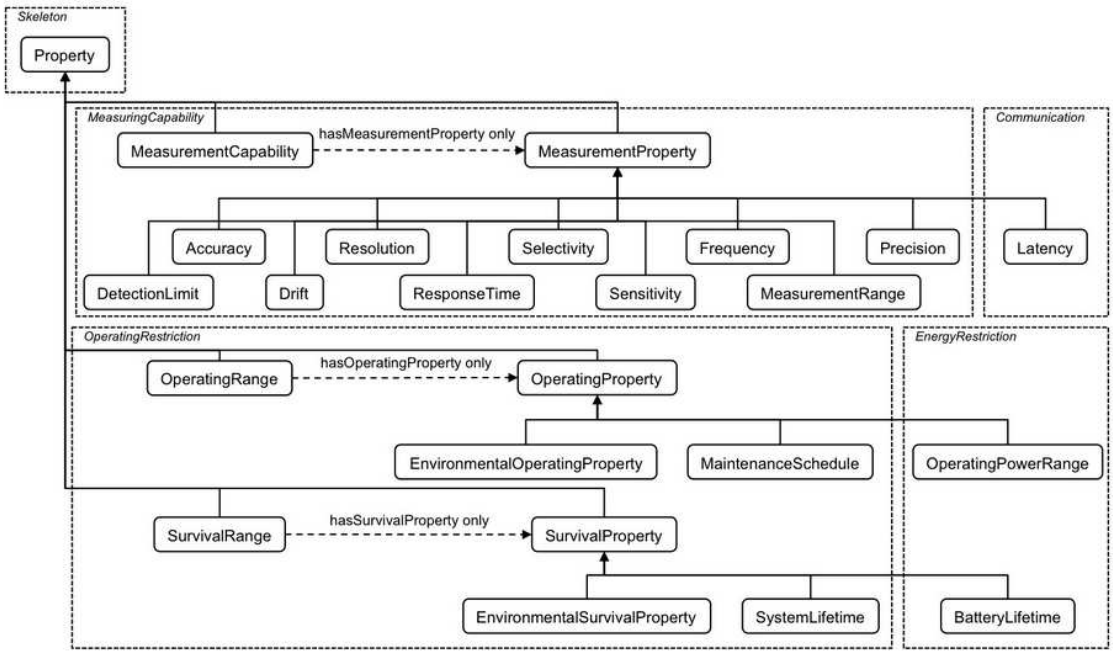


Figure: Enumeration of the measurement, environmental and survival properties [3].

Finally, the next figure shows how the ontology modules defined above support the use cases described in the previous section:

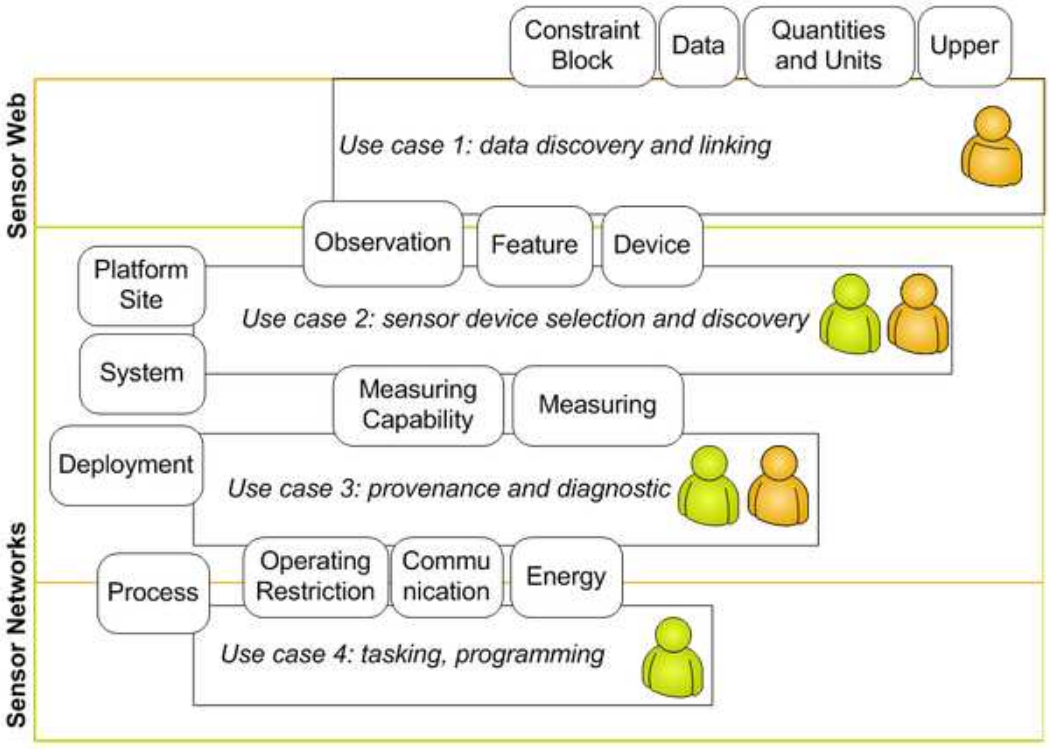
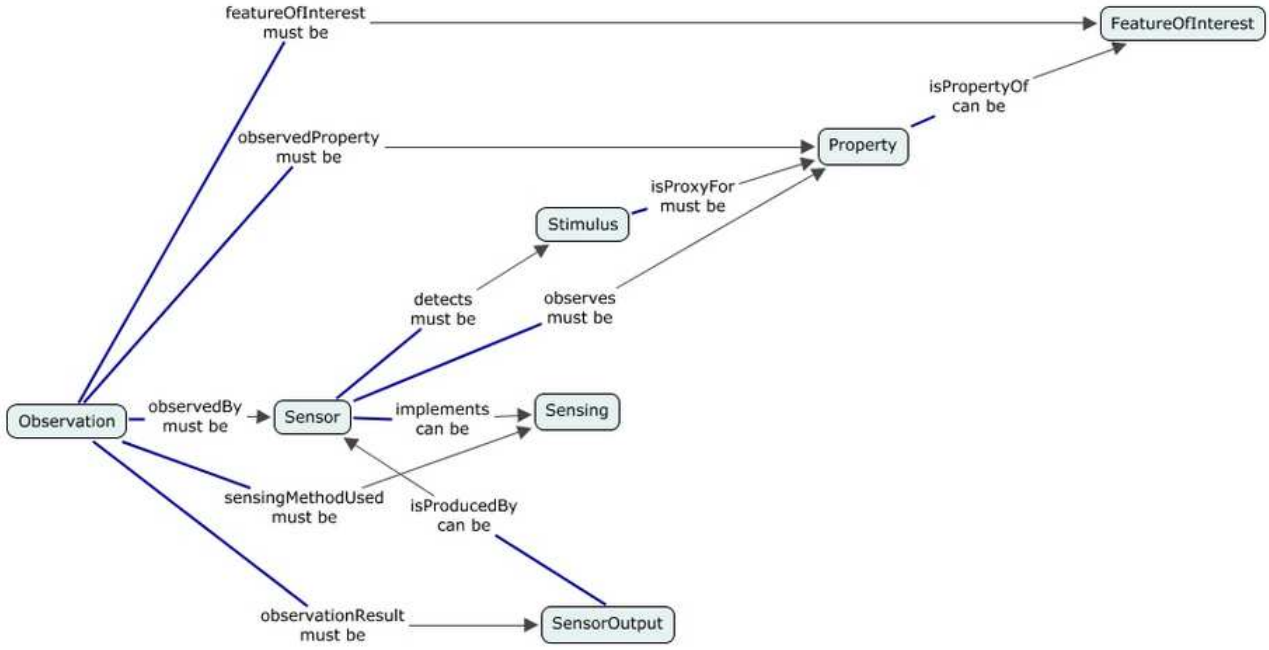


Figure: Use cases and ontology modules [3].

- Users developing Semantic Sensor Web applications, represented in orange in the figure, should be more specifically interested in the modules connected to the Data Discovery and Linking and Provenance and Diagnosis uses cases.
- Users developing Semantic Sensor Web applications, represented in green in the figure, also need the modules connected to the Device Discovery and Selection and Device Operation Tasking and Programming uses cases.
- A *Stimulus-Sensor-Observation (SSO) Ontology Design Pattern* has been defined to refine and improve the ontology skeleton and make it more easy to use it in conjunction with other ontologies, especially ones which are also based on a compatible "upper ontology" skeleton. Therefore, the SSN ontology skeleton is the sum of: a number of (mostly "local") ontology design decisions, plus re-engineering work done to align the ontology with SSO and DUL. The *Stimulus-Sensor-Observation Ontology Design Pattern* aims at all kind of sensor or observation based ontologies and vocabularies for the Semantic Sensor Web

and especially Linked Data. The pattern is developed following the principle of minimal ontological commitments to make it reusable for a variety of application areas. It is not aligned to any other top-level ontology and introduces a minimal set of classes and relations centered on the notions of stimuli, sensor, and observations.



- *Figure: The Stimulus-Sensor-Observation Ontology Design Pattern [3].*

4.3.4 Geospatial Ontologies for Indoor and Outdoor

Besides sensor ontologies, Chella et Al. [25] developed a spatial ontology for typical indoor environments. The model, called Ontology Description Model (ODM), allows one to represent environment knowledge in layers, and consequently, it supports multiple abstraction levels.

Domain ontology for describing both indoor and outdoor spaces is still on progress and there is no unified and standard ontology that can be adopted for any robotic navigation application [XX]. In fact navigation and sensing ontology will help robots to acquire sufficient semantic knowledge to navigate by taking into account the 3D view of any navigation space including their geometric properties, relationships and any other features of interest.

TBC

4.3.5 Ontologies for Policy Management

Within a Web Of Objects relying on semantic sensor networks (SSN) it is very important to control how to access to the information in a safe way. Several models can be used in this topic according to the targeted application and the sensitivity of the data and the operations on the objects. For instance, KAoS [xx], Rei [xx], MULTIPOL [xx], SAML-XACML [xxx] and SS-RBAC [11] are models proposes approaches to define abstract access control rules using ontology to allow for secure query processing under SSN and Authorization management in general. They are represented using RDF and XML Schema. In the delievrable 5.1 we present the description of the important policy languages that can be of particular importance in the context of this project.

5. WOO Semantic Modeling Framework

TBD Introductory text,

In this section we present a modeling framework that will be used to describe from the one hand the capabilities of objects according to the applications and deployment requirements in the WoO and from the other hand the representation language of schema of respectively data or knowledge that is delivered by the objects.
The modelling components that are proposed in the scope of the project are the following:

- XMPP
- DPWS
- REST
- Micro-Concept over RDF-S and OWL2 over RDF-S.

In the reminder of this section we detail how we used these components for describing the objects and their capabilities and how representations of the same object can be combined to augment the functionalities of the WoO platform. We use sensors as examples to figure out the modelling features of each component.

5.1.1. XMPP-DWPS Compliant Description of Objects

Explain shortly with simple example how Sensor services can be described in XMPP and DPWS. Since the gateway will make a transparent conversion.

TBD UPEC IGM XMPP Data Model used in BEC3

5.1.2. Linked Data - REST Compliant Description of Objects

The service petitions can be represented in different formats, for example in XML or JSON. We propose the JSON alternative because is simple and has lightweight. In addition, JSON doesn't need a validation as is the case of XML, which uses XMLSchemas to validate the correct format of the information.

In order to supply existing services or/and new ones, we can use REpresentational State Transfer (REST). It is a style of software architecture for distributed hypermedia systems, initially described in the context of HTTP, but is not limited to that protocol.

RESTful architectures (conforming to the REST constraints) are based on HTTP protocol, but can be based on other application layer protocols as long as they provide a representational state transfer. The basics are a client/server architecture where the clients send requests to

servers. Servers receive and process the request and return the response. The basic information unit is the resource: a unique object that can be addressed through a Unique Resource Locator (URL).

With respect to the integration of semantic description of linked data, we need to achieve how to integrate several heterogeneous components (sensors and actuators). One solution could be using RESTful services and JSON-LD to solve some constraints that REST is not able to manage.

JSON-LD is a serialization format for Linked Data based on JSON. The basic idea of JSON-LD is to create a description of the data in the form of a so called context. It adds semantics to existing JSON documents by linking data in the document to their description. Furthermore, it allows values to be type-coerced and language tagged. A context can either be directly embedded in a JSON-LD document or put into a separate file and referenced from different documents. JSON-LD can be chosen for interlinking resources provided by RESTful [15] [16] [17].

…

One interesting possibility is to support high-level semantic interaction without requiring multiple steps on multiple devices. The task is to provide a way for users to physically interact with devices on a high level of semantic abstraction without being bothered with the low-level details [14].

TBD Linked data description of REST services for sensor objects LISSI and UPM

5.1.3. REST-COAP Compliant Description of Objects

Explain shortly with simple example how we can make services I/O description of COAP sensor Thales and/OR IGM

5.1.4. Template Examples with JSON-LD or XMPP/DPWS

Template for Scene Understanding and Visual Tracking Services

Describing the Camera Tracking object as REST Service provider in JSON…(CEA, Thales ???) Information, Event ???, Provide a meta data description using JSON-LD (UPM, LISSI, Thales ?).

Describing the Camera Tracking object as XMPP/DPWS Service provider according to the BEC3 model …(CEA, IGM, Sogeti ???) Information, Event ???,

Template for Ambient Sensor and Actuator Services

… (Temperature, Fire, Smoke and Gas Sensor, RFID Presence Sensor)

?? Candidate

Template for Robots Control

… (Fire, Smoke and Gas Sensor, RFID Presence Sensor)

5.1.5. Semantic Description of Objects for Advanced Reasoning

We propose an OWL 2 compliant description model for modeling Objects and their relations using two possible representations: OWL/RDF-S or Micro-Concept RDF-S depending on the original description of objects, the targeted applications and reasoning. For instance, in case of matchmaking operations we advice using reasoning tools an OWL-RDF-S while in case of business monitoring of services orchestration we advice the use of business rules with Microconcept description instead of using OWL 2 mediation platforms.

This latter allows on the one hand, to build an explicit representation of Conceptual Objects and Their Corresponding Real world Objects using the Unique Name Assumption. It allows on the other hand for describing the actions that can be performed on the or by these objects. The Micro-Concept model also is considered as a support for the semantic language ‘’Smart Rules’’ to model the reasoning process for driving any process operating on the Objects, such as: inferring and reacting to contexts, trigger a business operation, etc.

The following figure shows a first draft of the general description of the objects and their relation.

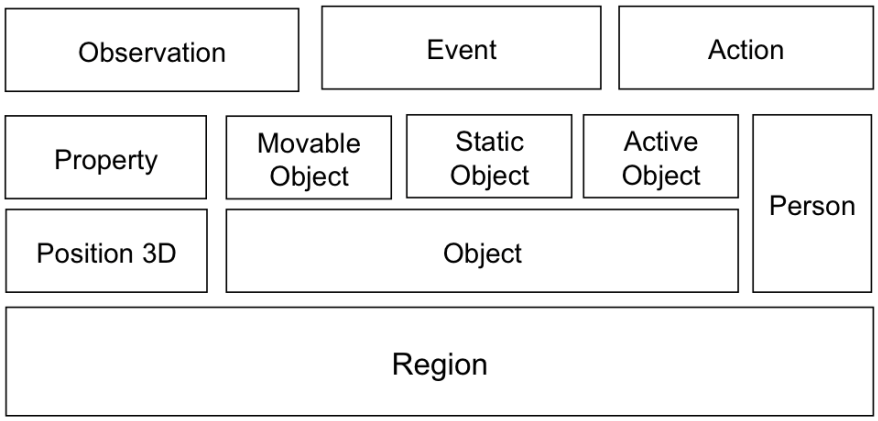


Figure 3 Main concepts for the Description of WoO Objects.

Our approach of aligning this model on the SSN domain ontology and respectivy the DOLCE DUL upper ontology is beneficial since we will reuse the main concepts and properties introduced in DUL so we can guarantee a minimum level of interoperability with exiting description of sensing objects. The second motivation behind the alignement with DOLCE is that it provides a guide for structuring application ontologies, especially regarding the taxonomic relationships.

The overall structure of the ontology part describing the objects consists of the following main elements:

Person: The domain of "Person" is the basic element around which other ontology domains are constructed. This domain can be described by a set of properties:

- The profile of the user, such as name, age, sex, etc.
- His medical history, such as type of disease, date of beginning of treatment, doctor, etc.
- The list of people to contact in case of emergency, such as its family, friends, etc.

Static Objects: These are physical objects that are used as resources for processes and activities. We can consider such as, a wall, a ceiling, a toilet, a sink, a mirror, etc.. In general, the environment is built around these objects via the interaction between humans and these objects.

Active Objects: They represent Physical Objects or Non Physical Objects that delivers services, which can be invoked using Service Grounding protocol in order to perform an operation for Reading Data or Triggering an Action on an Object such as opening or closing a Door, Switching On or Off an appliance or a power plant, rotate a camera, etc.. Non-physical objects are objects that can be used to process or store data. For instance, database of weather information.

Action: The actions are the result of interactions between objects or between human and Physical Objects. For example, a human can turn on the coffee maker.

Event: Represents the information generated by humans, by controllable objects / sensors, such as, sending a message, detecting the presence of a person, etc..

Position 3D and Region: Represents the coordinates and spatial relation of the region where an object is located. The concept of Region describes all the characteristics of the region even it is physical such as Building or virtual such as TV location, Shoping Mall, etc.

Modeling properties

Modeling properties (Quality of ontology DOLCE) in AmiOnt ontology is divided into two main categories: the first category is associated with relationships between concepts such as the localtedAt, isOpen, accessibleSpace, etc. The Locatedat property can be used to express for example: a robot is in the room, the isOpen property, can be used to represent for example: a door is opened and finally, accesssibleSpace property allows meanwhile, indicate that door giving access to a space is open. The second category of properties represent relationships between sensor networks. This category of property is imported from SSN ontology. In the latter, instances of the concept Sensor are associated with two basic properties: Detects and observed. Figure 3.11 shows the modeling of the knowledge and observation of the relationship of the concept Person. The Observation concept is a specialization of the Situation class defined in the DOLCE ontology. It represents either a raw data, such as degree of temperature in a space, or the detection of an event (Event), such as the presence of a person in a space.

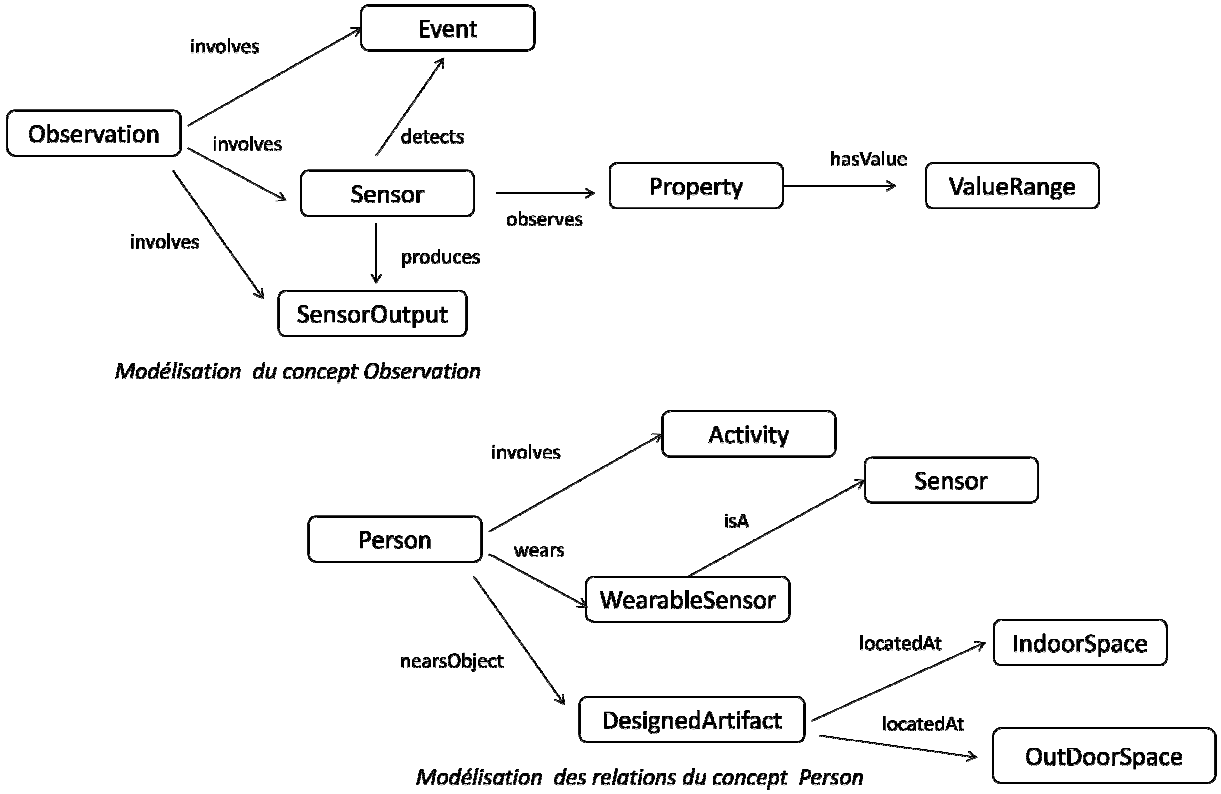


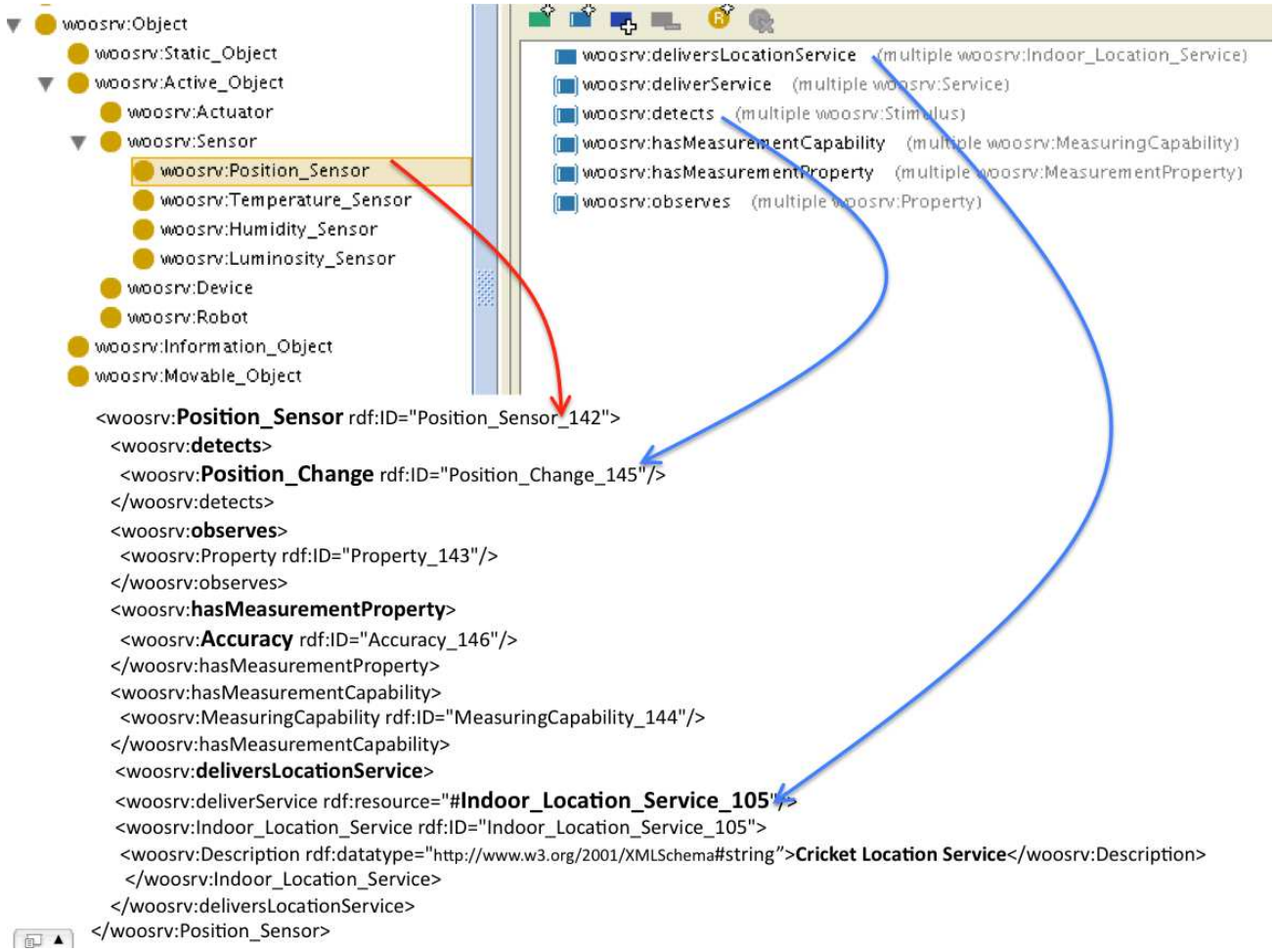
Figure 4 Structure of the AmIOnt ontology for modeling relationships between concepts

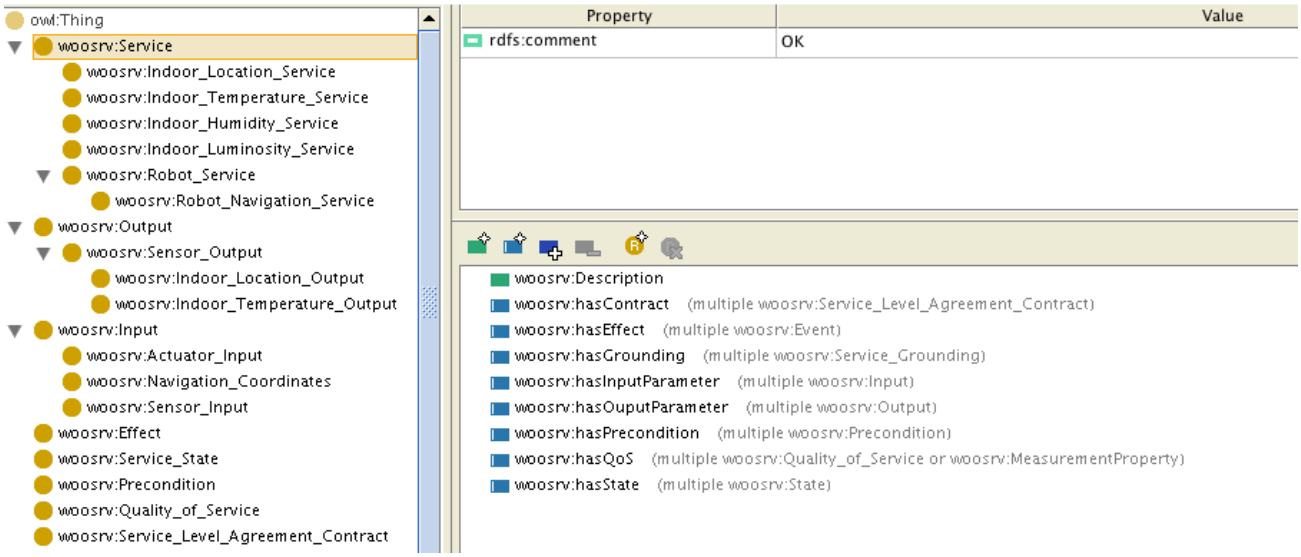
Modeling actions

In the Micro-Concept model, we introduced Action as fundamental branch (root) in the Micro-Concept Model. Thus, the choice of modeling actions avoids treating the hierarchy of actions by the inference mechanisms. Indeed, in the Micro-Concept model, we have just to define a relationship between the initiator of the action and the object being acted upon, and therefore every action is implicitly a sub-concept of the concept Action.

5.1.5.1. Template Examples for Semantic Description of Objects Capabilities

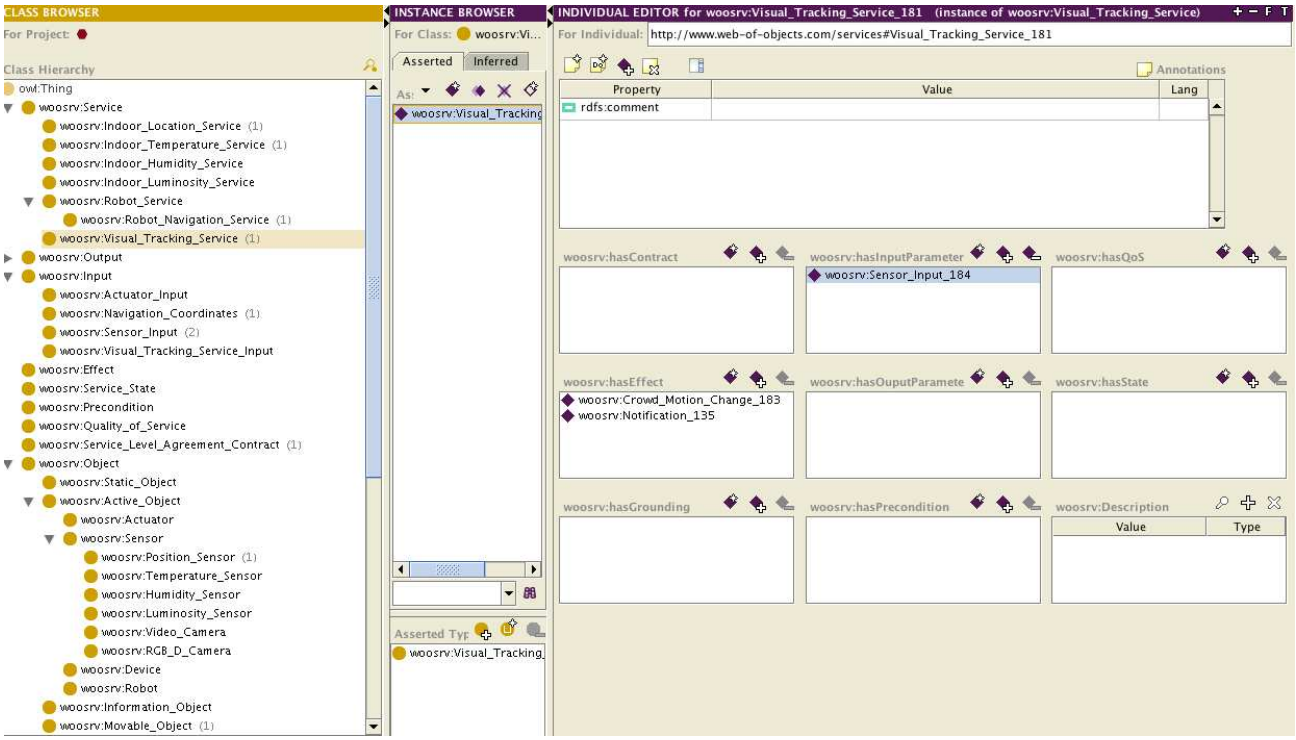
We propose the description of objects capabilities according to two abstraction views. The first view allow for describing the services delivered by objects for Discovery purpose, while the second view provides a technical description of the data representation and operations grounding in order to implement the applications that will act on or access to the Objects services. To allow for that we have been inspired from the OWL-S ontology model to define an ontology for object's services that is based on the General Concept of "Service".


woosrv:Object
woosrv:Static_Object
woosrv:Active_Object
woosrv:Actuator
woosrv:Sensor
woosrv:Position_Sensor
woosrv:Temperature_Sensor
woosrv:Humidity_Sensor
woosrv:Luminosity_Sensor
woosrv:Device
woosrv:Robot
woosrv:Information_Object
woosrv:Movable_Object
woosrv:deliversLocationService (multiple woosrv:Indoor_Location_Service)
woosrv:deliverService (multiple woosrv:Service)
woosrv:detects (multiple woosrv:Stimulus)
woosrv:hasMeasurementCapability (multiple woosrv:MeasuringCapability)
woosrv:hasMeasurementProperty (multiple woosrv:MeasurementProperty)
woosrv:observes (multiple woosrv:Property)
<woosrv:Position_Sensor rdf:ID="Position_Sensor_142">
<woosrv:detects>
<woosrv:Position_Change rdf:ID="Position_Change_145"/>
</woosrv:detects>
<woosrv:observes>
<woosrv:Property rdf:ID="Property_143"/>
</woosrv:observes>
<woosrv:hasMeasurementProperty>
<woosrv:Accuracy rdf:ID="Accuracy_146"/>
</woosrv:hasMeasurementProperty>
<woosrv:hasMeasurementCapability>
<woosrv:MeasuringCapability rdf:ID="MeasuringCapability_144"/>
</woosrv:hasMeasurementCapability>
<woosrv:deliversLocationService>
<woosrv:deliverService rdf:resource="#Indoor_Location_Service_105"/>
<woosrv:Indoor_Location_Service rdf:ID="Indoor_Location_Service_105">
<woosrv:Description rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Cricket Location Service</woosrv:Description>
</woosrv:Indoor_Location_Service>
</woosrv:deliversLocationService>
</woosrv:Position_Sensor>



Template for Scene Understanding and Visual Tracking Services

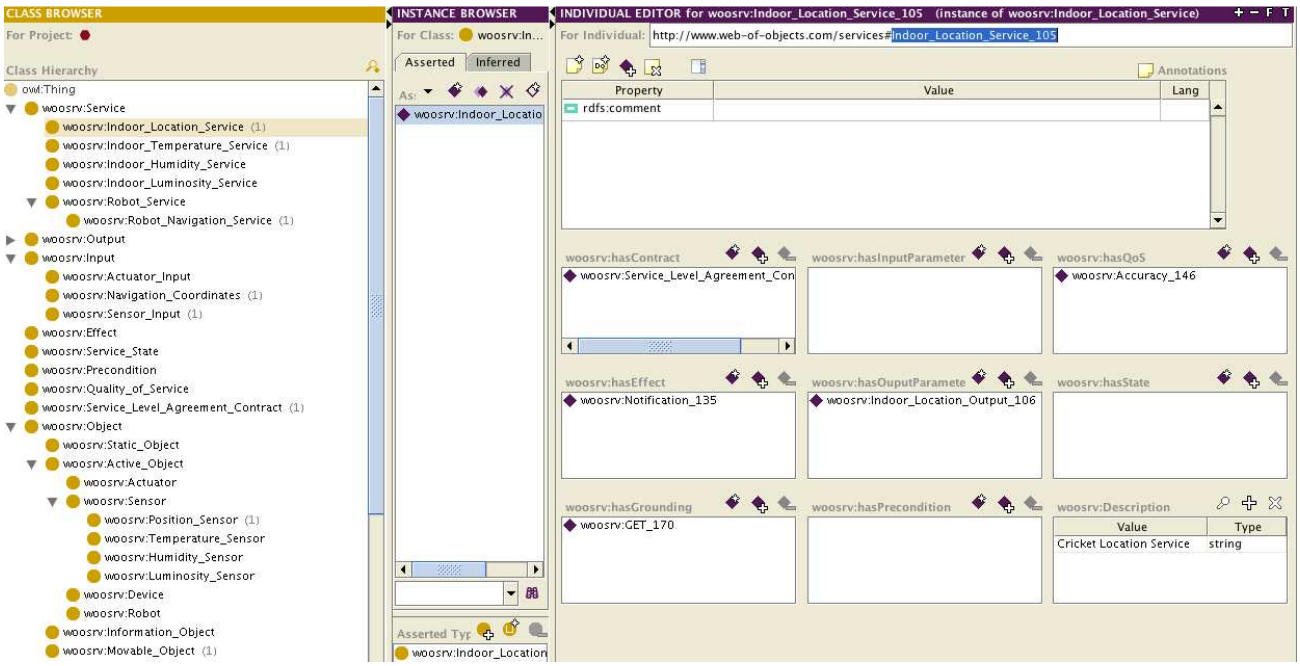
…LISSI, CEA ???



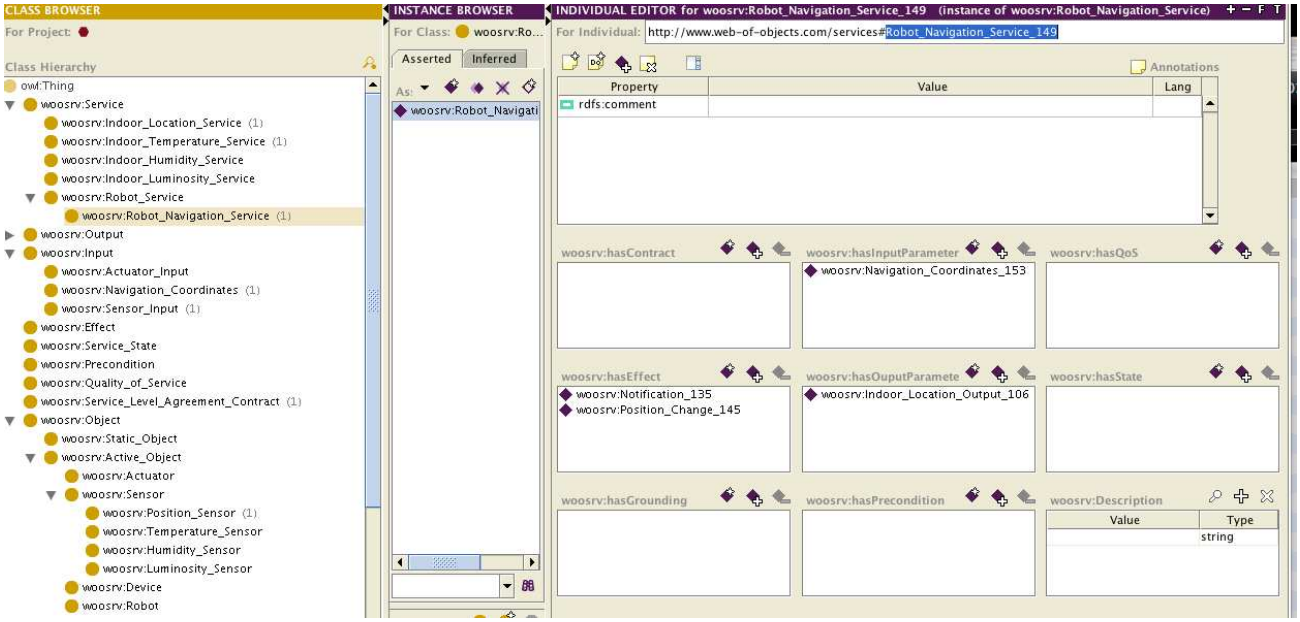
Template for Ambient Sensor and Actuator Services

… (Temperature, Fire, Smoke and Gas Sensor, Location Service, RFID Presence Sensor)

?? Candidate



Template for Robots Control



6. Guideline for the semantic modeling

7. Conclusion

8. References

- M. Schmidt-SchauB and G. Smolka. Attributive concept descriptions with complements. Artificial Intelligence, 48(1):1{26, 1991.
- U. Sattler. A concept language extended with different kinds of transitive roles. In G. Görz and S. Hölldobler, editors, Proc. of the 20th German Annual Conf. on Artificial Intelligence (KI'96), number 1137 in Lecture Notes in Artificial Intelligence,pages 333{345. Springer, 1996.
- Horrocks and U. Sattler. Ontology reasoning in the SHOQ(D) description logic. In Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001).
- Horrocks, O. Kutz, and U. Sattler. The even more irresistible SROIQ. In Proc.
- of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning
- (KR 2006), pages 57{67. AAAI Press, 2006.
- C. Lutz, C. Areces, I. Horrocks, and U. Sattler. Keys, nominals, and concrete domains. J. of Artificial Intelligence Research, .
- Horrocks and U. Sattler. A tableaux decision procedure for SHOIQ. In Proc. Of the 19 th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005), pages 448{453, 2005.

[1] http://www.json.org/

[2] http://www.json.org/xml.html

[3] http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/

[4] M. Botts OpenGIS Sensor Model Language (SensorML). OGC Implementation Standard OGC, 2007.

[5] S. Cox Observations and Measurements - Part 1 - Observation schema. OGC Implementation Standard OGC, 2007. S. Cox Observations and Measurements - Part 2 - Sampling Features. OGC Implementation Standard OGC, 2007.

[6] S. Havens OpenGIS Transducer Markup Language. OGC Implementation Standard OGC, 2007.

[7] OGC Network web page about TML: http://www.ogcnetwork.net/infomodels/tml.

[8] A. Na and M. Priest OpenGIS Sensor Observation Service. OGC Implementation Standard OGC, 2008.

[9] I. Simonis OpenGIS Sensor Planning Service Implementation Specification. OGC Implementation Standard OGC, 2007.

[10] I. Simonis OpenGIS Sensor Alert Service. OGC Candidate Implementation Specification OGC, 2006.

[11] Dongwon Jeong; Hyejin Jeong; Young-Sik Jeong; , "SS-RBAC: Secure Query Processing Model for Semantic Sensor Networks," Future Generation Communication and Networking, 2008. FGCN '08. Second International Conference on , vol.2, no., pp.352-355, 13-15 Dec. 2008.

[12] Malik, S.; Goel, A.; Maniktala, S.; , "A comparative study of various variants of SPARQL in semantic web," Computer Information Systems and Industrial Management Applications (CISIM), 2010 International Conference on , vol., no., pp.471-474, 8-10 Oct. 2010.

[13] Apache Jena web page:http://jena.apache.org/index.html

[14] Niezen, Gerrit; van der Vlist, Bram J.J.; Hu, Jun; Feijs, Loe M.G.; , "From events to goals: Supporting semantic interaction in smart environments," Computers and Communications (ISCC), 2010 IEEE Symposium on , vol., no., pp.1029-1034, 22-25 June 2010

[15] Markus Lanthaler and Christian Gütl. 2012. On using JSON-LD to create evolvable RESTful services. In Proceedings of the Third International Workshop on RESTful Design (WS-REST '12), Rosa Alarcon, Cesare Pautasso, and Erik Wilde (Eds.). ACM, New York, NY, USA, 25-32.

[16] JSON-LD web page: http://json-ld.org/spec/latest/json-ld-syntax/#data-model

[17] Eisfeld, A.; McMeekin, D.A.; Karduck, A.P.; , "Complex environment evolution: Challenges with semantic service infrastructures," Digital Ecosystems Technologies (DEST), 2012 6th IEEE International Conference on , vol., no., pp.1-6, 18-20 June 2012

Appendix

9. Appendix

9.1 RDF-S and OWL Constructs

In this section we will present the overall structure of RDFS and its main modeling primitives that are used for the OWL Language. We present after the sublanguages of the OWL languages

RDF-S constructs are the following :

• Core classes are rdfs:Resource, rdf:Property, and rdfs:Class. Everything that is described by RDF expressions is viewed as an instance of the class rdfs:Resource. The class rdf:Property is the class of all properties used to characterize instances of rdfs:Resource, i.e., each slot / relation is an instance of rdf:Property. Finally, rdfs:Class is used to define concepts in RDFS, i.e., each concept must be an instance of rdfs:Class.

• Core properties are rdf:type, rdfs:subClassOf, and rdfs:subPropertyOf. The rdf:type relation models instance-of relationships between resources and classes. A resource may be an instance of more than one class. The rdfs:subClassOf relation models the subsumption hierarchy between classes and is supposed to be transitive. Again, a class may be subclass of several other classes, however, a class can neither be a subclass of its own nor subclass of its own subclasses,

• Core constraints are rdfs:ConstraintResource, rdfs:ConstraintProperty, rdfs:range, and rdfs:domain. rdfs:ConstraintResource defines the class of all constraints. rdfs:ConstraintProperty is a subset of rdfs:ConstraintResource and rdf:Property covering all properties that are used to define constraints. Presently, it has two instances: rdfs:range and rdfs:domain that are used to restrict range and domain of properties. It is not permitted to express two or more range constraints on a property. For domains, this restriction is not enforced and, when more of a range constraint exists, this situation is interpreted as denoting the union of the domains.

As a conclusion, RDF is characterized by the presence of the following important concepts

- Resource: The resources being described by RDF are anything that can be named via a URI.
- Property: RDF entities are built up out of properties and their associated values. A property is also a resource that has a name.
- Statement: A statement consists of the combination of a Resource, a Property, and an associated value.

The main modeling primitives of RDF/RDFS concern the organization of vocabularies in typed hierarchies: subclass and subproperty relationships, domain and range restrictions, and instances of classes.

Mainly, the new capabilities added to ontology by OWL are:

- Compatibility with Web standards for accessibility and internationalization
- Scalability to Web needs
- Ability to be distributed across many systems
- Openness and extensibility

OWL retains the basic ability of RDF and the class- and property-structuring capabilities of RDF Schema and extends them in important ways. OWL can declare classes, and organize these classes in a subsumption ("subclass") hierarchy, as can RDF Schema. OWL classes can be specified as logical combinations (intersections, unions, or complements) of other classes, or as enumerations of specified objects, going beyond the capabilities of RDFS. OWL can also declare properties, organize these properties into a "sub-property" hierarchy, and provide domains and ranges for these properties, again as in RDFS.

The domains of OWL properties are OWL classes, and ranges can be either OWL classes or externally-defined datatypes such as string or integer. OWL can state that a property is transitive, symmetric, functional, or is the inverse of another property, here again extending RDFS.

OWL can express which objects (also called "individuals") belong to which classes, and what the property values are of specific individuals. Equivalence statements can be made on classes and on properties, disjointness statements can be made on classes, and equality and inequality can be asserted between individuals. OWL can define classes where a particular property is restricted so that all the values for the property in instances of the class must belong to a certain class (or datatype). At least one value must come from a certain class (or data-type); there must be at least certain specific values, and there must be at least or at most a certain number of distinct values.

OWL 1 includes three sublanguages:

OWL Lite: OWL Lite is a subset of OWL language that imposes several limitations on the use of the OWL's features. It is mainly intended for class hierarchies & simple constraints (cardinality 0 or 1, equality, etc).

OWL Lite excludes enumerated classes, disjointness statements and arbitrary cardinality (among others). The advantage of making use of these restrictions concerns the possibility of defining a language that is both easier to grasp (for users) and easier to implement (for tool builders). The disadvantage is linked of course to the restricted expressivity.

OWL DL: OWL DL includes every OWL Lite feature (any valid OWL Lite ontology is a valid OWL DL ontology, but not its inverse), and several new ones, but still imposes some

restrictions in order to be processed by a computer. OWL DL includes all OWL language constructs. It has a correspondence with description logics.

The well known Description Logic ALC (1) is the smallest propositionally closed DL. It contains :

- ✓ Concept operators: ∪ , ∩, ∃, ¬, ∀
- ✓ No role operators (only atomic roles)
- ✓ Concept axioms: ⊆ , ≡
- ✓ No role axioms

ALC extended with (role) transitivity transitively closed (S) primitive roles (2) where additional letters indicate various extensions, e.g.:

1. H : for role hierarchy
2. R : for role box
3. O : for nominals/singleton classes (e.g., {France})
4. I : for inverse roles
5. N : for number restrictions (e.g., >2equipedWithSensro)
6. Q : for qualified number restrictions (e.g., >2hasSensors.Camera)

=> **SHIQ** = **S** + role hierarchy + inverse roles + QNRs

These logics can also be extended with concrete domains (numbers, strings, etc) (5), which allow for the use of concrete types (e.g. integers) and integer comparisons. A simplified form of concrete domains, known as Datatypes, is denoted by appending (D) to the name of the logic, e.g., SHOIN(D).

In order to regain computational efficiency, OWL DL (short for: Description Logic) is a sublanguage of OWL Full that restricts the way in which the constructors from OWL and RDF can be used. We will give details later, but roughly this amounts to disallowing application of OWL's constructor's to each other, and thus ensuring that the language corresponds to a well studied description logic. The advantage of this is that it permits efficient reasoning support.

The disadvantage concerns the fact that we lose full compatibility with RDF: an RDF document will in general have to be extended in some ways and restricted in others before it can become a legal OWL DL document. Conversely, every legal OWL DL document is still a legal RDF document.

OWL Full: OWL Full allows for maximum expressiveness and all the syntactic freedom of RDF, but with no computational guarantees. Any valid OWL DL ontology is a valid OWL Full ontology, but not the inverse. It is unlikely that any software will be able to support every feature of OWL Full.

The advantage of OWL Full is that it is fully upward compatible with RDF, both syntactically and semantically: any legal RDF document is also a legal OWL Full document, and any valid

RDF/RDF Schema conclusion is also a valid OWL Full conclusion. The disadvantage of OWL Full concerns the fact that the language has become so powerful to turn out to be undecidable, dashing any hope of complete (let alone efficient) reasoning support.

9.2 Comparison of Semantic Web Services Ontologies

A first significant difference between the two approaches is that OWL-S does not separate what the user wants from what the service provides2. The Profile ontology is used as both an advertisement for the service and as a request to find a service. In WSMO, a Goal specifies what the user wants and the Web service description defines what the service provides through its capability.

In OWL-S, the non-functional properties in the Profile ontology (such as the service name, human-readable description and contact information) are not explicitly based on standard metadata specifications. WSMO recommends the use of widely accepted vocabularies such as Dublin Core [xx] and Friend of a Friend vocabulary [xx] metadata. Also, non-functional properties can be expressed in any element in WSMO, whereas in OWL-S this is restricted to the Profile ontology.

In OWL-S, the Service Model does not make a clear distinction between choreography and orchestration. It is not (at least explicitly) based on any formal model, albeit that there are external works defining the formal semantics of OWL-S processes. The Process ontology defines Atomic, Composite and Simple processes. In Composite processes, control constructs such as decisions, forks and loops can be used to model a workflow.

OWL-S defines only one Service Model per service, hence there is only one way to interact with the service. In WSMO, the choreography and orchestration are specified in the interface of the Web service description. A choreography describes the external visible behaviour of the service and an orchestration describes how other services are composed in order to achieve the required functionality of a service. Because we expect that there could be more than one way of interacting with a particular service, WSMO allows the definition of multiple interfaces for a single service.

To define logical expressions, OWL-S provides the choice between SWRL, DRS and KIF. By leaving the choice of the language to be used to the user, OWL-S contributes to the interoperability problem, rather than solving it. Furthermore, the interaction between the inputs

[2] http://www.w3.org/Submission/WSMO-related/#owl-s

and outputs, which have been specified as OWL classes, and the logical expressions in the respective languages, is not clear. Furthermore, DRS does not have a semantics associated with it and KIF is not web-compliant. WSMO provides a family of layered logical languages (see Web Service Modeling Language) which enables us to combine conceptual modeling with rules on only the required expressibility level, allowing descriptions to stay more explicitly within efficiently computable fragments.

To facilitate linkage of heterogeneous resources between one another, various kinds of mediation are required. For example, a service might need to make use of an ontology specified in OWL rather than WSML, requiring mediation (in this case ontology mapping) for the purpose of describing this service in WSMO.

WSMO was focused on the end user and developed a "conceptual syntax" for top-level descriptions of services, which we believe might make the specifications easier to read. WSMO also paid special attention to the issue of OWL compatibility. To this end, we defined WSML-Core as a subset of both OWL and WSML, which will serve as a common ground for ontology interoperability. In contrast, SWSL's focus was on extending the functionality of their rule-based language. In particular, SWSL-Rules supports meta-reasoning with its HiLog and reification extensions. It also supports prioritized defaults and classical negation by incorporating Courteous Logic Programming.

On the process description front, WSMO and SWSL are more divergent, but they nonetheless cover complementary parts of the problem space. WSMO has focused on describing Web service choreography through ECA rules, which are viewed as abstract state machines. In contrast, SWSL has developed a first-order PSL-based process ontology, which allows the description of process orchestration as well as message exchange among processes. Further work is needed to determine to what extent the two approaches can be combined.

OWL-S is clearly more mature in certain aspects, including the choreography and grounding specifications. However, WSMO provides a more complete conceptual model as it addresses aspects such as goals and mediators. However the main critics of OWL-S concern its limited expressiveness of service descriptions in practice, which in fact, corresponds to that of its underlying description logic OWL-DL. Only static and deterministic aspects of the world can be described in OWL-DL, since it does not cover any notion of time and change, nor uncertainty. Besides, in contrast to WSDL, an OWL-S process model cannot contain any number of completely unrelated operations. Mediators are a key element in describing Web services and therefore WSMO explicitly defines them in the conceptual model. OWL-S does not explicitly address the issue of mediation: it is considered to be handled by the underlying infrastructure.

Unlike OWL-S or WSML, SAWSDL does not specify a new language or upper ontology for semantic service description but simply provides mechanisms by which ontological concepts

that are defined outside WSDL service documents can be referenced to semantically annotate WSDL description elements. Major critic of SAWSDL is that it comes, as a mere syntactic extension of WSDL, without any formal semantics. In contrast to OWL-S and (in part) WSML, there is no defined formal grounding of neither the XML-based WSDL service components nor the referenced external metadata sources (via modelReference). Quoting from the SAWSDL specification: “Again, if the XML structures expected by the client and by the service differ, schema mappings can translate the XML structures into the semantic model where any mismatches can be understood and resolved.” This makes any form of logic-based discovery and composition of SAWSDL service descriptions in the semantic Web rather obsolete but calls for “magic” mediators outside the framework to resolve the semantic heterogeneities.

Another problem with SAWSDL today is its –apart from the METEOR-S framework by the developers of SAWSDL (WSDL-S) and related ongoing development efforts at IBM– still very limited software support compared to the considerable investments made in research and development of software for more advanced frameworks like OWL-S and WSMO world wide. However, the recent announcement of SAWSDL as a W3C recommendation does not only support a standardized evolution of the W3C Web Service framework in principle (rather than a revolutionary technology switch to far more advanced technologies like OWL-S or WSML) but certainly will push software development in support of SAWSDL and reinforce research on refactoring these frameworks with respect to SAWSDL.

9.3 SSN Ontology described module by module

Skeleton module:

A *Stimulus-Sensor-Observation (SSO) Ontology Design Pattern* has been defined to refine and improve the ontology skeleton and make it more easy to use it in conjunction with other ontologies, especially ones which are also based on a compatible "upper ontology" skeleton. Therefore, the SSN ontology skeleton is the sum of: a number of (mostly "local") ontology design decisions, plus re-engineering work done to align the ontology with SSO and DUL. The *Stimulus-Sensor-Observation Ontology Design Pattern* aims at all kind of sensor or observation based ontologies and vocabularies for the Semantic Sensor Web and especially Linked Data. The pattern is developed following the principle of minimal ontological commitments to make it reusable for a variety of application areas. It is not aligned to any other top-level ontology and introduces a minimal set of classes and relations centered on the notions of stimuli, sensor, and observations.

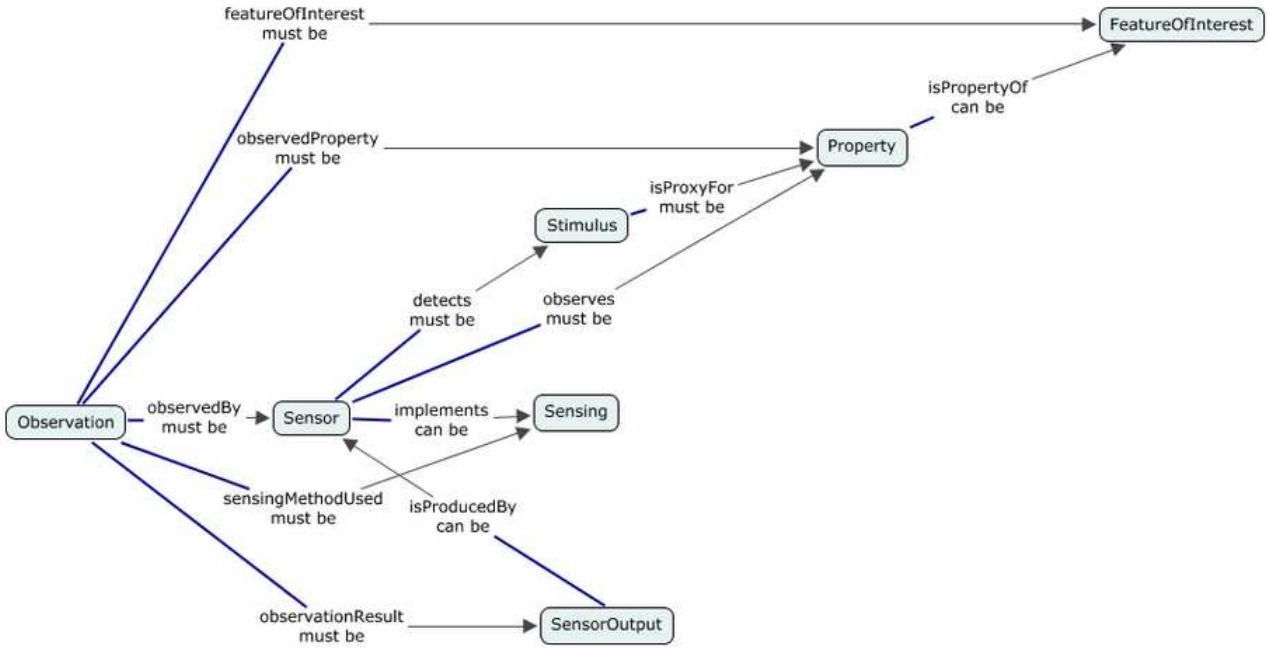


Figure: The Stimulus-Sensor-Observation Ontology Design Pattern [3].

System module:

The *ssn:System* class for parts of a sensing infrastructure. A system has components, its subsystems, which are other systems. The *ssn:hasSubSystem* property is used to relate a system to its sub-systems. The following figure shows an example where the system is a sensor network node (a "mote") which includes a Temperature sensor and a Humidity sensor (this is allowed because *ssn:SensingDevice* is a sub-class of System).

```
<owl:Ontology rdf:about="http://purl.oclc.org/NET/ssnx/uni/uni-deploy">
...
<!-- System (TelosB Mote) -->
<ssn:System rdf:about="http://purl.oclc.org/NET/ssnx/uni/uni-deploy#SN-Node-TSB-ABC01">
 <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A sample sensor network node. The data is for
 TELOS-B product and the data sheet is available at: http://www.willow.co.uk/TelosB_Datasheet.pdf </rdfs:comment>
 <ssn:hasSubSystem>
  <ssn:SensingDevice rdf:about="http://purl.oclc.org/NET/ssnx/uni/uni-deploy#TemperatureSensor-TSB-ABC01">
   <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">The embedded temperature sensor.</rdfs:comment>
  </ssn:SensingDevice>
 </ssn:hasSubSystem>
 <ssn:hasSubSystem>
  <ssn:SensingDevice rdf:about="http://purl.oclc.org/NET/ssnx/uni/uni-deploy#HumiditySensor-TSB-ABC01">
   <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">The embedded humidity sensor.</rdfs:comment>
  </ssn:SensingDevice>
 </ssn:hasSubSystem>
</ssn:System>
```

Figure: How to represent a system composed of sensors [3].

Process module:

The *ssn:Process* class in the SSN ontology is defined as the specification of the procedure implemented in a sensor. The next figure shows an example of how the process is used to define the characteristic transfer function of a Wind Sensor.

```
<owl:Ontology rdf:about="http://purl.oclc.org/NET/ssnx/meteo/phenonet">
...
<aws:UltrasonicWindSensor rdf:about="http://purl.oclc.org/NET/ssnx/meteo/phenonet#Wxt520Windcap">
  <dul:isPartOf>
    <ssn:System rdf:about="http://purl.oclc.org/NET/ssnx/meteo/phenonet#VaisalaWxt520"/>
  </dul:isPartOf>
  <ssn:implements>
    <ssn:Process rdf:about="http://purl.oclc.org/NET/ssnx/meteo/phenonet#Process24">
      <ssn:hasOutput>
        <ssn:Output rdf:about="http://purl.oclc.org/NET/ssnx/meteo/phenonet#SpeedAve">
          <dul:hasParameter>
            <dim:VelocityOrSpeedUnit rdf:about="http://purl.org/NET/ssnx/qu/unit#metrePerSecond"/>
          </dul:hasParameter>
          <phenonet:variable_code>SM</phenonet:variable_code>
          <phenonet:variable_name>Speed Ave</phenonet:variable_name>
          <phenonet:snlog_code>24</phenonet:snlog_code>
          <phenonet:sensor_name>WXT520 Windcap</phenonet:sensor_name>
          <phenonet:sensor_id>WIND</phenonet:sensor_id>
        </ssn:Output>
      </ssn:hasOutput>
      <dul:hasQuality>
        <dim:VelocityOrSpeed rdf:about="http://purl.oclc.org/NET/ssnx/cf/cf-property#wind_speed"/>
      </dul:hasQuality>
    </ssn:Process>
  </ssn:implements>
  ...
</aws:UltrasonicWindSensor
```

Figure: How to represent a process implemented by a sensor [3].

Measuring module:

The class *ssn:Sensor* in the ontology provides the structure to represent a concrete sensing object. The ontology defines several properties for instances of the class *ssn:Sensor*:

- *ssn:observes*: points to a property observed by a sensor (e.g., temperature, acceleration, wind speed). An object of this property must be an instance of the class ssn:Property.
- *ssn:hasMeasurementCapability*: Points to the description of the sensor's measurement capability expressed as an instance of the class *ssn:hasMeasurementCapability*. The

description of a measurement capability includes such parameters as frequency, accuracy, measurement range, etc.

The following figure shows an example of an accelerometer description:

```
 <owl:Thing rdf:about="#ExampleWiTilt30Accelerometer">
  <rdf:type rdf:resource="#WiTilt30Accelerometer"/>
  <rdfs:comment>
A specific instance of a WiTilt 3.0 accelerometer attached to a knife.
 </rdfs:comment></nowiki>
 <ssn:hasMeasurementCapability rdf:resource="#ExampleWiTiltAccelerometerMeasurementCapability"/>
 <ssn:onPlatform rdf:resource="#Knife_123"/>
 </owl:Thing>
</nowiki>
```

Figure: How to create a description of a sensor [3].

MeasuringCapability module:

The measurement capabilities of a sensor are collected using the class *ssn:MeasurementCapability*. Each instance of the class describes a set of measurement properties of a sensor in specific conditions. Possible measurement properties of a sensor are represented as subclasses of the class *ssn:MeasurementProperty*. Currently, the ontology defines the following types of measurement properties: *ssn:Drift*, *ssn:Sensitivity*, *ssn:Selectivity*, *ssn:Accuracy*, *ssn:MeasurementRange*, *ssn:DetectionLimit*, *ssn:Precision*, *ssn:ResponseTime*, *ssn:Frequency*, *ssn:Latency*, *ssn:Resolution*.

In figure *** is represented an example of how to describe capabilities of a sensor.

```
<owl:Thing rdf:about="#ExampleWiTiltAccelerometerMeasurementCapability">
  <rdf:type rdf:resource="#WiTilt30AccelerationMeasurementCapability"/>
  <ssn:hasMeasurementProperty rdf:resource="#WiTilt30BinaryModeFrequency"/>
  <ssn:hasMeasurementProperty rdf:resource="#WiTilt30MeasurementRange_1"/>
</owl:Thing>
```

Figure: *How to create a describe capabilities of a sensor [3].*

Observation module:

The class *ssn:Observation* provides the structure to represent a single observation. An observation is a situation that describes an observed feature, an observed property, a sensor

and method of sensing used and a value observed for the property: that is, an observation describes a single value attributed to a single property by a particular sensor.

The SSN ontology defines several properties for instances of the class *ssn:Observation*:

- *ssn:featureOfInterest*: points to the observed feature of interest. A feature of interest can be any observed real-world phenomenon (e.g., geographic entity).
- *ssn:observedProperty*: points to the specific quality (properties in the ontology are qualities that can be observed by a sensor; qualities, on the other hand, can also abstract, qualities of abstract things, or in some other way not able to be sensed) of the feature of interest which was observed (e.g., temperature, acceleration, etc.).
- *ssn:observedBy*: points to a sensor which produced the observation (an instance of the class *ssn:Sensor*).
- *ssn:sensingMethodUsed*: points to a method used to produce the observation (an instance of the class *ssn:Sensing*). This could describe, for example, a particular way in which the sensor is used to make the observation.
- *ssn:observationResult*: points to a result of the observation expressed as an instance of the class *ssn:SensorOutput*.
- *ssn:qualityOfObservation*: points to the adjudged quality of the result. This is complementary to the measurement capability information expressed for the sensor itself.
- *ssn:observationResultTime*: points to the time when the observation result became available.
- *ssn:observationSamplingTime*: points to the time when the observation result applies to the feature of interest.

The result of an observation is expressed by an instance of the class *ssn:SensorOutput*. The ontology defines the following properties applicable to the class:

- *ssn:isProducedBy*: points to a sensor which produced the output (an instance of the class *ssn:Sensor*).
- *ssn:hasValue*: points to the actual value of the observation (e.g., "30℃", "60 mph", etc.). This is expressed as an instance of the class *ssn:ObservationValue*. The ontology does

not restrict the format of an observation value: the actual properties can be defined by the user or imported from a third-party ontology.

The following picture shows a representation of a sensor observation.

```
<owl:Thing rdf:about="#KnifeCuttingObservation_435782677">
  <rdf:type rdf:resource="#AccelerationObservation"/>
  <rdfs:comment>An example of the acceleration observation produced by our ExampleWiTilt30Accelerometer. </rdfs:comment>
  <ssn:observationResult rdf:resource="#KnifeSensorOutput_2355676"/>
  <ssn:featureOfInterest rdf:resource="#Knife_123"/>
  <ssn:observedBy rdf:resource="#ExampleWiTilt30Accelerometer"/>
</owl:Thing>
...
<owl:Thing rdf:about="#KnifeSensorOutput_2355676">
  <rdf:type rdf:resource="#AccelerationSensorOutput"/>
  <rdfs:comment>An example of the acceleration sensor output produced by our ExampleWiTilt30Accelerometer. </rdfs:comment>
  <ssn:isProducedBy rdf:resource="#ExampleWiTilt30Accelerometer"/>
  <ssn:hasValue rdf:resource="#ZAxisAccelerationMeasurementValue"/>
</owl:Thing>
...
<AccelerationValue rdf:about="#ZAxisAccelerationMeasurementValue">
  <hasQuantityValue rdf:datatype="&xsd;float">0.98</hasQuantityValue>
</AccelerationValue>
```

Figure: How to represent an observation of a sensor [3].

Deploy module:

The main classes and properties related to deployment of a network of sensors in the ontology:

- *ssn:hasDeployment*: Points to deployment description of sensor expressed as an instance of the *ssn:Deployment* class. The description of a Deployment refers to *ssn:System* and it is also a subclass of *ssn:DeploymentRelatedProcess* and inherits all the properties from this class.
- The *ssn:DeploymentRelatedProcess* class groups various processes related to deployment. For example, it includes installation, maintenance and deployment features.
- The *ssn:System* class for parts of a sensing infrastructure. A system has components, its subsystems, which are other systems. A system is deployed on a Platform.
- *ssn:deployedOnPlatform* points to platform on which the system is deployed.
- *ssn:Platform* includes different components that can be attached to Sensor and also different features such as measurement properties, operating properties and system settings.
- *ssn:deployedSystem* provides relation between a deployment and a deployed system.

In the following OWL example, the defined Platform handles two sensors participating in a deployment scenario and belonging to the same system. The system is a sensor network node (a "mote") which includes a Temperature sensor and a Humidity sensor.

```
<owl:Ontology rdf:about="http://purl.oclc.org/NET/ssnx/uni/uni-deploy">
...
<!-- Deployment -->
<ssn:Deployment rdf:about="http://purl.oclc.org/NET/ssnx/uni/uni-deploy#ABC01Deployment">
 <ssn:deployedOnPlatform>
  <ssn:Platform rdf:about="http://purl.oclc.org/NET/ssnx/uni/uni-deploy#UNiS-TSBPlatform"/>
 </ssn:deployedOnPlatform>
 <ssn:deployedSystem>
  <ssn:System rdf:about="http://purl.oclc.org/NET/ssnx/uni/uni-deploy#SN-Node-TSB-ABC01">
   <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A deployment which includes a temperature and a humidity
   sensor on the same sensor network node.</rdfs:comment>
  </ssn:System>
 </ssn:deployedSystem>
</ssn:Deployment>
<!-- Platform (and Operating conditions) -->
<ssn:Platform rdf:about="http://purl.oclc.org/NET/ssnx/uni/uni-deploy#UNiS-TSBPlatform">
 <DUL:hasLocation>
  <DUL:PhysicalPlace rdf:about="http://purl.oclc.org/NET/ssnx/uni/uni-deploy#PhysicalPlace_UNiSTestBED-BA03A">
   <DUL:isLocationOf rdf:resource="http://purl.oclc.org/NET/ssnx/uni/uni-deploy#UNiS-TSBPlatform"/>
  </DUL:PhysicalPlace>
 </DUL:hasLocation>
 </ssn:Platform>
 <!-- System (TelosB Mote) -->
 <ssn:System rdf:about="http://purl.oclc.org/NET/ssnx/uni/uni-deploy#SN-Node-TSB-ABC01">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A sample sensor network node. The data is for
  TELOS-B product and the data sheet is available at: http://www.willow.co.uk/TelosB_Datasheet.pdf </rdfs:comment>
  <ssn:hasDeployment>
   <ssn:Deployment rdf:about="http://purl.oclc.org/NET/ssnx/uni/uni-deploy#ABC01Deployment"/>
  </ssn:hasDeployment>
  <ssn:hasSubSystem>
   <ssn:SensingDevice rdf:about="http://purl.oclc.org/NET/ssnx/uni/uni-deploy#TemperatureSensor-TSB-ABC01">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">The embedded temperature sensor. </rdfs:comment>
   </ssn:SensingDevice>
  </ssn:hasSubSystem>
 <ssn:hasSubSystem>
  <ssn:SensingDevice rdf:about="http://purl.oclc.org/NET/ssnx/uni/uni-deploy#HumiditySensor-TSB-ABC01">
   <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">The embedded humidity sensor.</rdfs:comment>
  </ssn:SensingDevice>
 </ssn:hasSubSystem>
</ssn:System>
<!-- Temperature sensor -->
<ssn:SensingDevice rdf:about="http://purl.oclc.org/NET/ssnx/uni/uni-deploy#TemperatureSensor-TSB-ABC01">
 <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A specific instance of temperature sensor. </rdfs:comment>
 <ssn:hasDeployment>
  <ssn:Deployment rdf:about="http://purl.oclc.org/NET/ssnx/uni/uni-deploy#ABC01Deployment"/>
 </ssn:hasDeployment>
<!-- Humidity sensor -->
<ssn:SensingDevice rdf:about="http://purl.oclc.org/NET/ssnx/uni/uni-deploy#HumiditySensor-TSB-ABC01">
 <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A specific instance of humidity sensor. </rdfs:comment>
 <ssn:hasDeployment>
  <ssn:Deployment rdf:about="http://purl.oclc.org/NET/ssnx/uni/uni-deploy#ABC01Deployment"/>
 </ssn:hasDeployment>
</ssn:SensingDevice>
```

Figure: How to represent a deployment of a network of sensors [3].

PlatformSite module:

The SSN ontology defines relationships between classes such as Deployment, System and Platform. However it does not provide some aspects such as spatial attributes for the Platform class. There are three options to represent locations. When the DOLCE Ultra Lite alignment is