ebbits

Enabling the business-based
Internet of Things and Services

(FP7 257852)

D7.2 Event and data structures, taxonomies and ontologies

Published by the EBBITS Consortium

Dissemination Level: Public

Project co-funded by the European Commission within the 7th Framework Programme
Objective ICT-2009.1.3: Internet of Things and Enterprise environments

Document version: 1.0 Submission date: 24.2.2011

Document control page

Document file: D7.2_Event_and_data_structures_taxonomies_and_ontologies_v1.0 (1).doc
Document version: 1.0
Document owner: TUK

Work package: WP7 – Event Management and Service Orchestration
Task: T7.2 Event and data structures, taxonomies and ontologies
Deliverable type: R

Document status: ☒ approved by the document owner for internal review
☒ approved for submission to the EC

Document history:

Version	Author(s)	Date	Summary of changes made
0.1	Peter Kostelnik (TUK) Jan Hreno (TUK)	20.12.2010	Initial structure, Time plan, Tasks distribution
0.2	Peter Kostelnik (TUK)	10.1.2011	State of the art in sensor semantic models and database-ontology mapping.
0.3	Peter Kostelnik (TUK)	17.1.2011	Preliminary design of event-management support ontology
0.4	Alcaraz Gonzalo, Claudio Pastrone, Maurizio Spirito (ISMB), Thomas Nejsum Madsen (TNM), Roberto Checcozzo (COMAU)	26.1.2011	State of the art in real-world data structures.
0.5	Peter Kostelnik (TUK)	28.1.2011	Update of ontology design according to real-world data structures SOTA.
0.6	Marek Paralič (IS), Karol Furdík (IS)	6.2.2011	State of the art in publisher/subscriber architectures.
0.7	Peter Kostelnik (TUK)	8.2.2011	Update of ontology design according to publisher/subscriber architectures SOTA, introduction text and final formatting.
0.8	Matts Ahlsen (CNET)	16.2.2011	Executive summary
0.9	Alcaraz Gonzalo, Claudio Pastrone, Maurizio Spirito (ISMB)	23.2.2011	Section 3 updated to include a sub-section on event and alarm data structures in the traceability scenario.
1.0	Peter Kostelnik (TUK)	24.2.2011	Improved formulations and formatting according to the review comments. Final formatting.

Internal review history:

Reviewed by	Date	Summary of comments
Matts Ahlsen (CNET)	17.2.2011	Approved with comments
Martin Knechtel (SAP)	22.2.2011	Approved with comments

Index:

1. Executive summary

Event management is central in the ebbits architecture as a conceptual model and execution mechanism for the integrating of physical world events with enterprise systems, pursuing the vision for the IoTS. This report provides a background to and a basis for the design and implementation of the ebbits event management subsystem.

The architecture style of ebbits can be characterized as *Event-driven SOA*, integrating intelligent services with advanced semantic event processing and business rules. Thus events can range from lower level atomic signals to higher level semantically enriched message carriers. On a higher abstraction layer ebbits events are mapped to business rules, which can make use of intelligent services, to implement the business logic for reporting, actuation of devices, as well as for further event generation. In this way the ebbits platform will be able to support the concept of *closed loop eventing* on different levels of abstraction.

The first level of event processing is the mapping of events to the perimeter systems which are generating the event stimuli and providing entry points for actuation. Thus, this report takes its starting point in the data structures and communications protocols employed in the two application domains in the project, the *factory automation* scenario and the *food traceability* scenario respectively. In the former domain there is a certain level of standardization, in terms of factory automation protocols and standards for information exchange, in farming and food systems, proprietary and special purpose systems dominate but ISO standards are available for information exchange.

We then present our view of the state-of-the-art in event management and in semantic support for sensor and event systems, with relevance for ebbits. A number of event management systems have been proposed, many of which are based on the well-known *publish and subscribe* event processing pattern. Semantic support for sensor and event systems include the use of ontologies and mark-up languages for the modelling of stimuli, sensor devices, events and actors in such systems. The middleware developed within the Hydra project, is one of the baseline technologies for ebbits, which also provides a publish & subscribe event management solution as well as semantic support in terms of ontologies over events, services and devices.

Finally, we describe our proposal for an initial event model for ebbits, based on the Hydra ontologies in combination with the SSN – Sensor model promoted by the W3C. This model is subject to further design and implementation.

2. Introduction

The ebbits platform aims to support interoperable business applications with context-aware processing of data separated in time and space, information and real-world events (addressing tags, sensor and actuators as services), people and workflows (operator and maintenance crews), optimisation using high level business rules (energy or cost performance criteria). The key requirements for the business rules execution is that the ebbits platform needs to be able to recognise and respond to physical world events. The information acquired from events from physical world generated by various devices create the basis for decision making at the several levels of the ebbits architecture, including data fusion, situation patterns recognition, complex event processing, analysis of historical acquired data, etc.

All of above mentioned requirements needs to work with the large amount of information related to the devices generating the events or providing the services for further processing by event/service orchestration, decision or business rules. In some cases it must be possible to use this information to analyse the historical data generated by particular events.

All parts of decision making process will be supported by the rich semantic model enabling the flexible knowledge representation of all included events, roles, services and processes.

2.1 Purpose, context and scope of this deliverable

The ebbits infrastructure will have to deal with the various types of events, which will be processed and mapped to the real-world actions or to the recommendations for the human manual intervention. The purpose of this deliverable is to design the preliminary semantic model describing the low-level events generated by the devices. Although the ebbits will have to be aware also of higher level events describing e.g. system statuses or human actions, the scope is bounded only to the modeling of events related to the devices. The design of the preliminary ontology is based on the state of the art in the area of real-world data structures commonly used by the devices, overview of publisher/subscriber event management architectures and the assumptions based on the past experiences from the Hydra event management framework. The first ontology design will cover all assumed requirements derived from this information. The content of ontology will, of course, evolve with the continually growing user requirements and requirements on the software architecture.

3. Physical world data structures

This chapter provides an analysis of the state-of-the-art on data structures currently considered within the "physical world" (including devices and sub-systems) of both the manufacturing and the food traceability scenarios.

3.1 Data structures in manufacturing scenario

Along the years, many attempts have been made by different standardization bodies to provide comprehensive solutions for networked monitoring and control in industrial scenarios. Nowadays, the proposed communication technologies are being widely adopted, actually building environments where different heterogeneous solutions coexist and are not always interoperable. As far as the data structures are concerned, several standards have been presented but their widely adoption is still far from being real. In fact, end-users commonly use data structures highly customized for specific manufacturing scenarios and for very specific purposes.

A major reason for having such a custom approach being adopted by industrial manufacturers is that the computerization of industrial plants has been pursued on top of already existing automation solutions. In fact, the integration of the new technologies is sometimes hindered by its remarkable impact on work organization and human resources. In addition, the available standards could be less optimized compared to customized solutions defined for a specific manufacturing plant.

In the following, the data structures considered in the Common Industrial Protocol [CIP, 2010] will be described as they can be adopted in several industrial networks including EtherNet/IP, DeviceNet, CompoNet and ControlNet [ODVA, 2011]. Moreover, an example of customized solution will be provided.

3.1.1 *Common Industrial Protocol (CIP)*

CIP is a media independent, connection-based, object-oriented protocol designed for automation applications. While the main focus of the proposed protocol supported by the Open DeviceNet Vendors Association (ODVA) is on the definition of communication architecture suitable for industrial automation, CIP also supports interoperability at application level by using abstract object modelling. Accordingly, industrial automation devices can be seen as CIP nodes described by a collection of objects that actually specify nodes functionalities.

CIP objects expose specific attributes and services and can be grouped into three main types according to the class of functionality considered: *general-use*, *application-specific* and *network-specific*. Objects can be either *publicly defined* or *vendor-specific*.

Moreover, CIP objects representing the same type of system component are grouped in a *class*. A specific representation of an object belonging to a given class is called *instance* and is characterized by distinct attributes values. It is worth mentioning that the object attributes are expressed using the data types defined in standard IEC 61131-3 [IEC, 2003]. According to this standard, the data types can be elementary (basic data types) or structured (array or structure of elementary data types). Common data types used in CIP include:

- 1-bit (encoded into 1-byte) – Boolean, BOOL;
- 1-byte – Bit string/8 bits/BYTE, Unsigned 8-bit integer/USINT, Signed 8-bit integer/SINT;
- 2-byte – Bit string/16-bits/WORD, Unsigned 16-bit integer/UINT, Signed 16-bit integer/INT;
- 4-byte – Bit string/32-bits/DWORD, Unsigned 32-bit integer/UDINT, Signed 32-bit integer/DINT.

CIP also introduces *device profiles* that allow to model manufacturing devices by means of a specific collection of objects. Different manufacturing devices have been already modelled e.g., fluid flow controllers, inductive proximity switches, process control valves. For each of the

modelled devices, the standard specifies the set of mandatory and optional objects that need to be supported. For instance, a motor starter is composed by an *identity object*, an *overload object* and a *discrete output object* [CIP, 2001]. The structure of identity object is reported in the table 1.

Identity Object	
Mandatory Attributes	**Optional Attributes**
Vendor ID (UINT)	State (UINT)
Device Type (USINT/UINT)	Configuration Consistency Value
Product Code (UINT)	Heartbeat Interval (UINT)
Revision (USINT)	Languages Supported
Status	
Serial Number (DWORD)	
Product Name (ASCII string)	

Table 1. CIP Identity Object.

The resulting CIP *object oriented data structures* contained within each device can be accessed by using an addressing scheme organized as follows: *[Class ID] [Instance ID] [Attribute ID, if required]*. The figure 1 shows a CIP object addressing example.

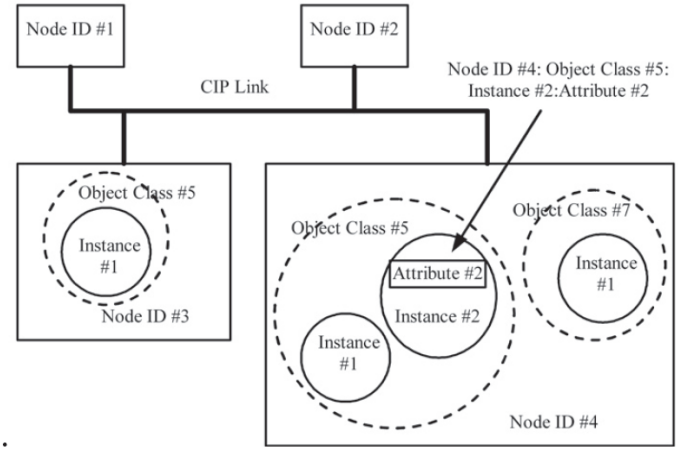

Figure 1. CIP object addressing example - Copyright © ODVA

3.1.2 Customized solution

In this subsection, an example of customized solution provided by COMAU is presented.

The considered scenario relates to a *production line* composed of different *stations*, each controlled by a "*station PLC*". Such PLC has the responsibility to control all the operations at station level including robot synchronization, units clamping and system transportation. In addition, the same PLC periodically transfers information about the station functioning to a "manufacturing line server" that is in charge of managing all the stations of the considered production line.

In fact, the interaction among the *station PLC* and both *sensors* and actuators present in the station is usually handled only at PLC station level. With respect to the possible integration of the manufacturing line into the ebbits platform, the main focus is then on the communication between the station PLC and the manufacturing line server and the underlying data structures. More specifically, the only information exposed by a PLC is a high level description of the production status and system anomalies. In practice, this represents the basic information required to calculate OEE (Overall Equipment Efficiency) [OEE; 2011] performance and prepare general reports.

Custom data structures are adopted to manage the following information:

- *Station status*: is the current status of the station being controlled by the PLC. Station status is associated to a priority level and possible relevant values include Residual, Emergency stop, Blocked, Empty line, Manual and Wait for cycle start.
- *Anomalies*: refer to anomalies and alarms in the monitored manufacturing line. The provided data includes the type of anomaly, the location where the anomaly has been detected, a brief description of the anomaly and a relevant identifier.
- *Cycle time*: is the time required by the system to complete the specific manufacturing process. The count starts from the beginning of the operation and stops every time the system is not producing. The cycle is restarted for every piece of product.
- *Counters of the production process*: are a set of counters and timers providing additional information on the production status. Counters include e.g. the number of produced elements within 1 hour/24hour, the number of good parts and scraps. Timers are associated to e.g. Shift, Residual and production times.

Standard data types are used to define the custom data structures.

3.2 Data structures in food traceability scenario

Food traceability covers the entire life-cycle of the food, from the farm to the fork. However, in this document, the focus is mainly on the farm environment.

In such environment, current management solutions are based on stand-alone computers and require the same data to be manually entered into and collected from diverse embedded systems. This represents a laborious task which becomes superfluous when computers are interconnected and able to automatically share and exchange information. The resulting networked environment would allow effective data exchange between embedded systems used to monitor and control the farm process and a centralized management server.

The main goal is to ease the life of the farmer by avoiding worthless double entry of the same data and, if possible, by making specific sub-processes autonomous. For instance, if a pig dies and is removed, the farmer should only input such piece of information into the feeding computer located in the pen. Automatically, the change should reflect on a farm management program behavior and eventually on a climate controller. Another example could relate to feeding computers able to transfer summations of feeding data to the farm management program. Moreover, event-based asynchronous communication capabilities should be required to handle events and alarms in an effective way.

These are generally triggered by devices not easily accessible by the farmer. In that case, the farmer would go to the place where the alarm was activated to get additional details, thus loosing valuable time that could be used for other affairs. Instead, the farmer should be able to monitor and handle alarms from any interactive device within the farm.

In order to support the expected interaction among the different components participating in the farm management process, reference data structures should be adopted. The *ISO 11788* [ISO, 2000] standard could be a suitable solution: it specifies an *Agricultural Data Element Dictionary* (ADED) consisting of data structures that can be used to interchange information electronically between management systems and stationary computers in dairy, pig and poultry farming.

ADED is closely linked to *Agricultural Data Interchange Syntax* (ADIS), specified by *ISO 11787* [ISO, 1996] standard and intended for non real time data communication. ADED in combination with ADIS makes electronic data interchange possible. In addition to ISO 11788 standard, an extension [TNM, 2009] has been developed within the Datastandard project[1] in order to include event and alarm data exchange in ADED-based farming systems.

[1] http://www.datastandard.dk/

3.2.1 The Agriculture Data Element Dictionary

Given a specific farm environment, the ADED is organized in *areas*, *entities* and *data elements*. For instance, the ADED for pig farming includes three main areas: area of pigs, area of feed and area of climate. For each area, the standard specifies entities intended as reference data structures which group basic data elements (items).

In fact, an ADED could represent a specified sub-set of a larger dictionary. This document subsection focuses only on a basic layout; however, the complete ADED including some proposals for extensions can be found in [ISO, 2000].

In an ADED, entities and items are univocally described and identified by a unique code composed of six digits. The international codes have "9" as the first digit; e.g. the ISO approved codes end up in this category. Digits 1 through 8 indicate whether a code is defined on national level and the digit 0 is reserved for internal communication on vendor level. More specifically, entities having a unique code beginning with "99" are already approved by ISO, whereas entities whose code begins with "xx" are *Draft International Standards* still to be endorsed by the ISO technical committee. Similarly, items having a unique code beginning with "90" are approved by ISO, whereas items whose code begins with "xx" are still under approval.

Moreover, specific items in the *ISO 11788* standard can be described by additional code sets. The code set can be either *normative* or *informative*. A normative code set is specified in an International Standard, while an informative code set only gives an example of possible values.

It is worth noting that a specific code set can be valid only at national level e.g. when there is an agreement on the item description, but the value of the code set differs according to the considered country. The national code sets are listed in the national data dictionary.

The different types used to define the items are the following: *datetime*, *int*, *char* and *double*.

In fact, an entity can be compared to a class (C\#, Java), structure (C) or record (Pascal) in a programming language and contains a set of items which in turn is comparable to fields in a class/structure/record. For instance, a "Feed component" entity (struct) has items (fields) such as the "Dry matter content" and the "Feed component name".

In a database context, the entities could be seen as table specifications and the items as the columns of the table. This close correspondence leads to a web service API with features resembling simple SQL statements.

ADED entities

In [ISO, 2000], the complete list of existing and proposed entities is reported. An example is shown in table 2.

The "Feed component" in the ISO standard is described as an entity belonging to "area of feed" and whose unique code is *990012*. The "K" in the first column of the table specifies that the "Feed component number" field is a key (identifier) item that must be present in the entity. In general, the values of the key items uniquely identify an instance of an entity. In the provided example, "Feed component" is uniquely identified by the values of "Feed component number" and "Feed lot number". The "M" in the first column indicates a mandatory item that is not part of the key and thus not necessarily unique. Within a given entity, the items not labelled with "M" or "K" are considered optional.

According to the code rule reported above, Feed component is an internationally recognized entity. Moreover, while "Feed lot number" item has been already approved by ISO, the "Methionine per kilogram" item is still at proposal stage.

990012	Feed component			
Give information on the major characteristics of a feed component (a feed component could be a mix of several feed)				
O	ADED-nr	Name	Type	Unit
K	900099	Feed component number	int	Unique within farm
K	900100	Feed lot number	int	
	900101	Feed type number	int	
	900102	Feed component name	char	
M	900103	Dry matter content	Double	% of feed component
	900104	Energy Type	char	
	900105	Energy per kilogram	Double	MJ/kg
	900106	Crude protein per kilogram	Double	g
	900107	Phosphorus per kilogram	Double	g
	900108	Calcium per kilogram	Double	g
	900109	Lysine per kilogram	Double	g

Table 2. 990012 - Feed component.

3.2.2 Event and alarm data exchange

While the ISO 11788 standard already covers several aspects of farm management, support for events and alarms is not included. In order to address this issue, the Datastandard project has proposed an extension [TNM, 2009] focused on event and alarm data exchange and relevant data structures in ADED-based farming systems. The main goal is to provide a communication pattern enabling the exchange of event and alarm data between *farm process computers/embedded systems* and *farm management software* made available from different vendors.
In the specification, data structures for events and alarms are proposed.
More specifically, *events* are described by a set of parameters useful to univocally identify and describe the event occurred. Location and time information is included. Table 3. summarises the information specified for a given event notification.

906011		Event Notification	
O	DD	Name	Type
K	906001	Device Entity ID	Int
K	990001	Location Entity	String
M	906051	Event Vendor ID	String
M	906052	Event ID	Int
	906053	Event Description	String
	906131	Creation Time	Time
	907132	Updated Time	Time

Table 3. 906011 – Event notification.

On the other hand, alarms are indicated as particular events which need higher attention and that can be handled by human beings or specific management software. The description of an alarm should include the following information:

- *Priority* - this parameter is used to detail the level of importance of the alarm thus allowing the definition of specific rules in handling the different incoming alarms. The proposed classification, ranging from non critical to critical (important) alarms, is shown below:
  - Test and Debug (used for test and debugging purposes)

  - Info (referring to not critical events, still useful to properly monitor the status of the system)
  - Warning (used for alarms that require immediate notification)
  - Alarm (used for alarms that require to be managed in order to avoid additional problems)
  - Fatal (used for the most critical alarms)
- *Model* - this parameter is used to describe the type of operations required to handle the various alarm states. The following alarm models are taken into account:
  - Manual (alarms requiring human interaction)
  - Automatic (alarms handled by automatic logic)
  - Local (alarms requiring local reset)
- *State* - this parameter relates to the different possible alarm states introduced to support the above handling schemes. Five states are defined:
  - Normal (not active alarms)
  - Silenced (alarms without abnormal process condition, still requiring acknowledgement)
  - Active (alarms denoting abnormal process condition)
  - Suppressed (active alarms, already acknowledged but not yet solved, that are shelved for a given amount of time)
  - Disabled (alarms disabled)

The considered standard extension thus describes alarms by means of models which define the services that must be implemented depending on the current alarm state. The transitions between states are performed by the exchange of *messages* defined as *entities* which contain *items* with the relevant information. For instance, an important item described in the specification is the *Alarm State* whose value (integer type) contains the current state of the alarm.
The complete list of entities and items can be found in [TNM, 2009].

3.3 Data Structures used in Wireless Sensor Networks

In ebbits project, it is expected that monitoring, control and optimization of processes in manufacturing and food traceability scenarios are co-supported by the adoption of Wireless Sensor Network (WSN) technology. In fact, WSN represents a very promising technology and offers interesting characteristics in terms of flexibility, adaptability and efficiency. In addition, some of the available WSN standards already provide frameworks dealing with event management, application logic and relevant data structures. Nevertheless, it is worth mentioning that such technology is not meant to be included as part of the automation or farm networks but as a complement. Accordingly, the adoption of WSN technology would definitely simplify the integration of new features in already existing systems where the introduction of new elements could be a troublesome task. For instance, actual automation networks adopt customized protocols optimized for specific purposes and the addition of new devices would need to be carefully evaluated.

WSN technology could be profitably used to gather energy consumption information, possibly fused with other relevant data made available to ebbits platform, thus enabling event-based manufacturing process monitoring and optimization. This would allow to achieve one of the ebbits project objectives which is to introduce also the energy component in the computation of the Overall Equipment Efficiency (OEE) index, usually considered to measure the effectiveness of the production.

The following subsections present data structures adopted in two standard solutions endorsed by the ZigBee Alliance called ZigBee Smart Energy 2.0 profile and the ZigBee Cluster Library. The ZigBee Energy 2.0 profile could be used in ebbits project to support energy related events management i.e. asynchronous exchange of information concerning energy monitoring and control. Finally, the ZigBee Cluster Library provides a list of data structures that could be of interest and adapted to support novel application level features (identified from requirements analysis) in both the manufacturing and food traceability scenarios.

3.3.1 ZigBee Smart Energy 2.0 Profile

Smart Energy Profile (SEP) 2.0 [SEP, 2010] is a joined standard being designed by the ZigBee Alliance and HomePlug Powerline Alliance. The main objective is to provide a networking and application layer platform supporting the interaction between customer devices and energy services providers. Such solution would also promote the adoption of monitoring and control features to reduce global energy consumption. SEP 2.0 is intended to run on any network relying on IPv6 protocol. As far as WSN technologies are concerned, IEEE 802.15.4 has been taken into consideration jointly with the adoption of the IETF 6LoWPAN standard as the relevant adaptation layer.

SEP 2.0 basically adopts a web paradigm. A SEP 2.0 device can be considered as a server hosting specific application level capabilities exposed as web resources. The implementation of this web approach actually relies on a HTTP-based RESTful architecture where the application level functionalities are exposed as Uniform Resource Identifiers (URIs). SEP 2.0 also proposes as an alternative solution to adopt the Constrained Application Protocol (CoAP), being defined within Core IETF WG.

The *data models* used in SEP 2.0 are defined in the International Electro technical Commission's (IEC) *61970-301* [IECa, 2009] and *61968-11* [IECb, 2009]. IEC 61970-301 is a semantic model in charge of describing the components of a power system at an electrical level and the relationships between components. IEC 61968-11 extends the previous model and specifically addresses data exchange between power systems focusing on functionalities like asset tracking, work scheduling and customer billing. These two standards are jointly known as the *Common Information Model* (CIM) for power systems [IEC, 2007]. Their principal goal is to facilitate the exchange of energy related information among companies and among applications operating within the same company.

According to the above CIM, information is organized into *classes*, *sub-classes* and *attributes*. Within a system, a class represents a specific type of object being modelled. Each class can have its own internal attributes and relationships with other classes. Actually, the relations among the classes are established by leveraging on object-oriented methodologies including inheritance, aggregation, association and composition, thus reflecting the hierarchical structure of the modelled information.

In this context, IEC 61968-9 [IECc, 2009] is also taken into account. The standard defines the information content of a set of message types used for meter reading and control, meter events, customer data synchronization and costumer switching, thus supporting business functions related to Advanced Metering Infrastructure (AMI). While it is mainly intended for electrical distribution networks, IEC 61968-9 can also be adopted for gas and water networks.

Moreover, concepts called *Function Set* and *Devices types* have been introduced in SEP 2.0. A function set provides a set of resources and associated transactions, while devices types are logical devices whose main capabilities are specified by particular function sets. For instance, the device type "Meters" is expected to include the "Metering" function set that reflects the capability of exchanging information about the meter read, meter status and tariff information. In the figure 2, the Metering object model is reported.

3.3.2 ZigBee Cluster Library

The ZigBee Cluster Library (ZCL) [ZigBee, 2007] has been defined in the ZigBee specification in order to support data exchange at application layer. More specifically, ZCL provides a collection of pre-defined application messages, also called *clusters*. Each cluster specifies the *attributes* and *commands* defining the communication interface offered by a specific application level functionality.

The attributes are data entities associated to peculiar application level features exposed by a ZigBee device. For instance, an attribute could be the data entity containing the value of a temperature measurement performed by ZigBee-enabled thermometer or the status information of a ZigBee-enabled device, capable of being remotely controlled.

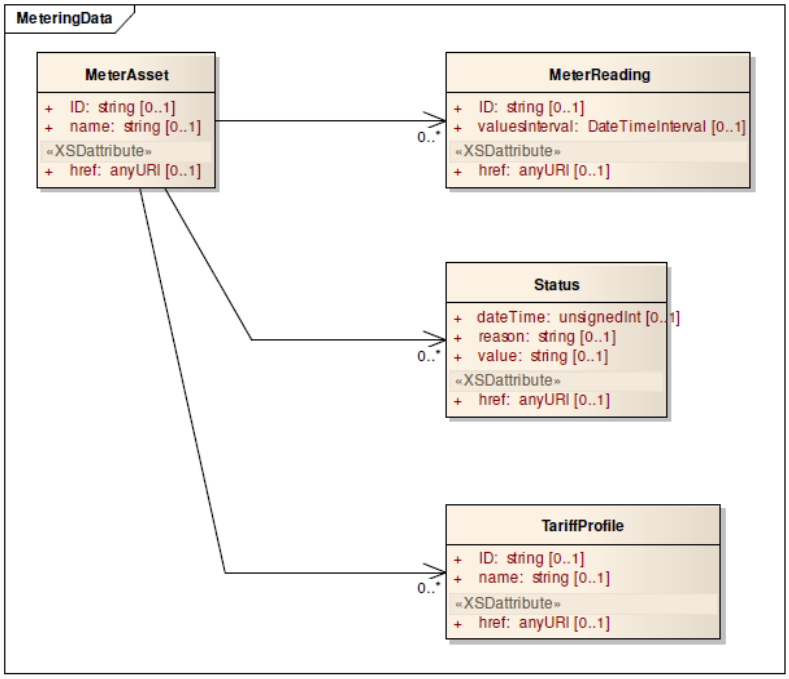

Figure 2. Metering Data Object Model [SEP, 2010].

The principal data types used to describe attributes in the ZCL are the following: *General Data* (8-bit until 32-bit data), *Logical* (Boolean), Bitmap (8-bit until 32-bit bitmap), *Enumeration* (8-bit and 16-bit), *Integer* (Unsigned and Signed), *Floating Point* (Semi, Single and Double precision), *String* (Octet and character String), *Time* (Time of day and date), *Identifier* (16-bit or 32-bit) and *Miscellaneous* (64-bit).

Attributes can be also grouped into *sets*. Then, *commands* can be used to read/modify the value of the attributes, to perform a discovery of the attributes exposed within a cluster and even to receive asynchronous notifications of some attribute-related events (e.g. in case, the temperature change exceeds a certain threshold).

As an example, a brief description of *Basic cluster* is provided. It defines attributes and commands needed to retrieve or set basic information about a device and to enable or reset the device to factory configuration. The cluster is organized into two attribute sets: *Basic Device Information* and *Basic Device Settings*. The attributes contained in the first set include *ZCLVersion, ApplicationVersion, StackVersion, HWVersion, ManufacturerName, ModelIdentifier, DateCode, PowerSource, LocationDescription, PhysicalEnvironment, DeviceEnabled* and *AlarmMask*. It is worth noting that this cluster has similar characteristics to the identity object described in CIP.

Within ZigBee standard, the clusters are actually used to define a device in terms of mandatory and optional offered capabilities. Devices belonging to the same application area are then grouped in *Application profiles*.

Home Automation profile [ZigBee, 2010] defines specific devices that could be of interest for the ebbits project and that relate to the following application areas: lighting, heating, Ventilating and Air Conditioning (HVAC), environmental monitoring, energy management, safety, and security. For instance, among the HVAC devices, the temperature sensor is defined. The table 4 shows the clusters required for the device.

Server Side	Client Side
Mandatory	
Temperature Measurement	None
Optional	
None	Groups

Table 4. Clusters supported by the Temperature Sensor device [ZigBee, 2010]

The *Temperature Measurement* cluster has one attribute set called *Temperature Measurement Information* containing the attributes presented in table 5.

Attributes of the Temperature Measurement Information Attribute Set		
Name	**Type**	**Mandatory/Optional**
MeasuredValue	Signed 16-bit Integer	M
MinMeasuredValue	Signed 16-bit Integer	M
MaxMeasuredValue	Signed 16-bit Integer	M
Tolerance	Unsigned 16-bit Integer	O

Table 5. Attributes of the Temperature Measurement Information attribute set [ZigBee, 2007].

4. Publisher / subscriber architectures

The publish/subscribe schema provides two principal players, the publisher (the information producer) and the subscriber (the information consumer). The publisher entity publishes the events on a software bus (or event channel), while the subscriber expresses his/her interest in a specific part of information (an event or in a pattern of events), submitting related subscription. The subscriber is notified when the submitted subscription matches an occurred event or a pattern of events. This communication model, schematically depicted in figure 3, provides simple and efficient methods for distributing information and guarantees decoupling in terms of time, space and synchronization, between publisher and subscriber [Eugster et al, 2003] [Etzion 2011])

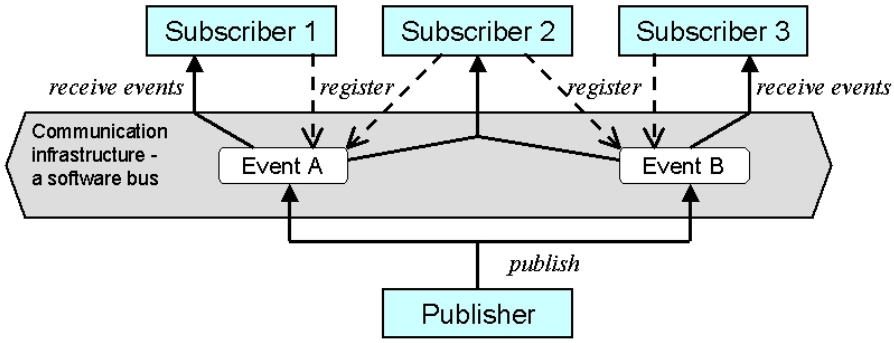

Figure 3. Publisher-subscriber architecture schema.

Large variety of emerging applications benefit from the expressiveness, filtering, distributed event correlation, and complex event processing capabilities of content-based publish/subscribe systems. These applications include RSS feed filtering, stock market monitoring engines, system and network management and monitoring, algorithmic trading with complex event processing, business process management and execution, business activity monitoring, workflow management and service discovery [Jacobsen et al, 2009].

An example of an open source distributed content-based publish/subscribe system is Padres [Padres, 2011], [Jacobsen et al, 2010], [Jacobsen et al, 2009] developed by the Middleware Systems Research Group at the University of Toronto. Content-based routing is enabled in cyclic overlays. Cyclic overlays provide redundancy in routes between sources and sinks and thus produce alternative paths between them. PADRES also implements other efficient load balancing and recovery algorithms to handle load imbalances and broker failures. The PADRES publish/subscribe broker is based on a content-based matching engine that supports the subscription language, including atomic subscriptions, the various forms of historic subscriptions, composite subscriptions with conjunctive and disjunctive operators, the isPresent operator, variable bindings, and event correlation with different consumption policies. PADRES includes a number of tools to help manage and administer a large publish/subscribe network, e.g. a monitor that allows a user to visualize and interact with brokers in real time, and a deployment tool that simplifies the provisioning of large broker networks.

Another interesting framework was created in context the Internet of Things (IoT) research effort – MAGIC Broker 2 (MB2) developed at the Media and Graphics Interdisciplinary Centre, University of British Columbia [Blackstock et al 2010]. MB2 middleware platform offers a simple, uniform web-based API for building IoT applications and offers developers three built-in programming abstractions: publish-subscribe event channels, persistent content and state storage, and brokerage of services via remote-procedure call. MB2 system was used to create a range of IoT applications involving spontaneous device interaction such as between mobile phones and public displays, and opportunistic or shared sensing and control of devices using a web-based sensor actuator network called Sense Tecnic (STS). The STS platform also includes facilities to process sensor data,

effectively creating higher-level sensors. A complex event-processing engine [Esper] is used to process lower-level sensor events, which are sent back into MB2 for output to higher-level derived sensor feeds that can be used by applications and visuals.

The mentioned system Esper [Esper] is an Event Stream Processing (ESP) and event correlation engine (CEP, Complex Event Processing) – i.e. it supports requirement to process events (or messages) in real-time or near real-time. Targeted to real-time Event Driven Architectures (EDA), Esper is capable of triggering custom actions written as Plain Old Java Objects (POJO) when event conditions occur among event streams. It is designed for high-volume event correlation where millions of events coming in would make it impossible to store them all to later query them using classical database architecture.

Generic Event Architecture GEAR [Casimiro et al 2007] is an architecture to provide the possibility of integration of physical and computer information flows in large distributed systems interacting with the physical environment and being composed from a huge number of smart components - systems-of-embedded-systems. GEAR utilises COoperating Smart devices (COSMIC) middleware [Kaiser et al 2005] as an appropriate event model. It allows specifying events with attributes to express spatial and temporal properties. This is complemented by the notion of Event Channels (EC), which are abstractions of the underlying network and enforce the respective quality attributes of event dissemination. Event channels reserve the needed computational and network resources for highly predictable event systems. The COSMIC middleware, maps the channel properties to lower level protocols of the regular network and defines an abstract network which provides hard, soft and non real-time message classes.

Finally, the Hydra middleware, also implements a publish and subscribe event management solution. Event channels are modelled by a "Topics" structuring sets of events. Event producers and consumers can then publish and subscribe to these topics. The producers and consumers of events are loosely coupled and topics/event types can be chosen at will.

5. Semantic technologies for event management

This section aims to provide the state of the art in the area of semantic technologies directly related to the modelling of the events produced by devices. The section is divided into two parts: overview of existing semantic models of sensors and the technologies for mapping between relational databases and ontologies.

The events related to the devices are mostly part of existing ontologies for modelling the sensors. Many times, the concepts, which correspond to the concept of *event* in ebbits are modelled as the observations or measurements made by the sensors. Therefore the overview of sensor ontologies is provided, but with the focus on modelling of events, or concepts corresponding to the understanding the *event* concept in ebbits. The analysis of existing ontologies modelling the events will serve as the basis for the identification of relevant parts of events and the design of the ebbits preliminary event model. There are also a lot of existing semantic models for higher level events, such as calendar events (meeting scheduled on concrete time to concrete room with concrete participants) or domain dependent events, e.g. system statuses (plant has passed the functionality tests). These higher level events are also planned to be part of the ebbits eventing architecture, but they are out of scope of this deliverable. The semantic models of higher-level events are planned to be created in WP3 or WP6.

As the data produced by device events and the events themselves will be stored for the purpose of further usage (e.g. historic data analysis), there is a need to enrich the model of events with the semantic information representing: what data were produced by which event generated by which device, where and how are those data stored. There is an assumption, that data produced by events (and the events themselves) will be stored in relational databases (also for the reason, that the semantic storages are not suitable to persist a lot of raw data). Following this assumption, there is a need to create semantic support for integrating the semantic data with existing database storages (in some cases not only for storing the events related data). For this reason, an overview of existing technologies for mapping between ontologies and relational databases is provided.

5.1 Semantic models of sensors

5.1.1 SensorML

SensorML [SensorML, 2007] is part of an Open Geospatial Consortium (OGC) initiative to contribute to the development of a Sensor Web through which applications and services will be able to access sensors of all types over the Web and was approved by OGC as an international, open Technical Specification on June 23, 2007. SensorML provides standard models and an XML encoding for describing the processes, including the process of measurement by sensors and instructions for deriving higher-level information from observations. Processes are discoverable and executable. All processes define their inputs, outputs, parameters, and method, as well as provide relevant metadata. Models detectors and sensors as processes that convert real phenomena to data.

Sensor systems or processes can make themselves known and discoverable. SensorML provides a rich collection of metadata that can be mined and used for discovery of sensor systems and observation processes. This metadata includes identifiers, classifiers, constraints (time, legal, and security), capabilities, characteristics, contacts, and references, in addition to inputs, outputs, parameters, and system location. Complete and unambiguous description of the lineage of an observation is also provided.

Process chains for geolocation or higher-level processing of observations can be described in SensorML, discovered and distributed over the web, and executed on-demand without a priori knowledge of the sensor or processor characteristics. This was the original driver for SensorML, as a means of countering the proliferation of disparate, stovepipe systems for processing sensor data within various sensor communities. This processing is enabled without the need for sensor-specific software.

SensorML descriptions of sensor systems or simulations can be mined in support of establishing OGC Sensor Observation Services (SOS), Sensor Planning Services (SPS), and Sensor Alert Services (SAS). SensorML defines and builds on common data definitions that are used throughout the OGC Sensor Web Enablement (SWE) framework.

SensorML enables the development of plug-n-play sensors, simulations, and processes, which seamlessly be added to Decision Support systems. The self-describing characteristic of SensorML-enabled sensors and processes also supports the development of auto-configuring sensor networks, as well as the development of autonomous sensor networks in which sensors can publish alerts and tasks to which other sensors can subscribe and react.

Finally, SensorML provides a mechanism for archiving fundamental parameters and assumptions regarding sensors and processes, so that observations from these systems can still be reprocessed and improved long after the original mission has ended. This is proving to be critical for long term applications such as global change monitoring and modeling.

The essential elements include: components, systems, process models, chains and methods, detectors and sensors.

SensorML is currently encoded in XML Schema. However, the models and encoding pattern for SensorML follow Semantic Web concepts of Object-Association-Object. Therefore, SensorML models could easily be encoded for the Semantic Web. In addition, SensorML makes extensive use of soft-typing and linking to online dictionaries for definition of parameters and terms.

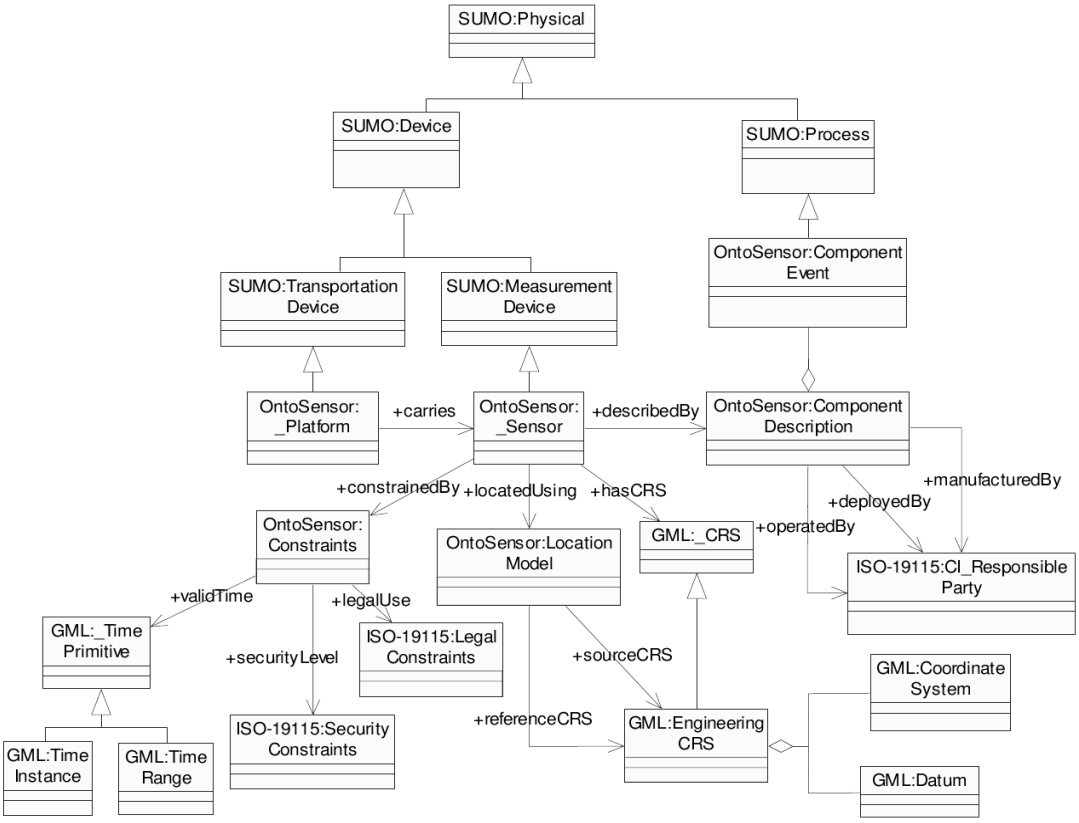

Figure 4. The part of OntoSensor ontology.

5.1.2 OntoSensor

OntoSensor [OntoSensor, 2005][OntoSensor, 2006] is a Semantic Web compatible ontology, which references and extends the IEEE Suggested Upper Merged Ontology (SUMO) [SUMO, 2001], which defines general concepts and associations. OntoSensor is based in parts upon SensorML [SensorML, 2007], which defines associations and properties common to sensors. OntoSensor deviates from SensorML since it lacks the semantic richness, such e.g. by axiomatically defined terms, which may be required for automated data fusion and inference in a distributed sensing environment. Currently,

OntoSensor includes knowledge models for the data acquisition boards, sensing elements, and processor/radio units, as well as preliminary definitions of a variety of imaging sensors. OntoSensor contains a hierarchy of sensor classes and describes sensor attributes, capabilities, and services. OntoSensor concepts and associations are instantiated in distributed repositories and updated by the base stations of the network. The data model for a given sensor contains meta data such as sensitivity, performance range, and accuracy for the sensing elements, as well as physical characteristics such as weight, radio frequencies, dimensions, and power supply information for the platform.

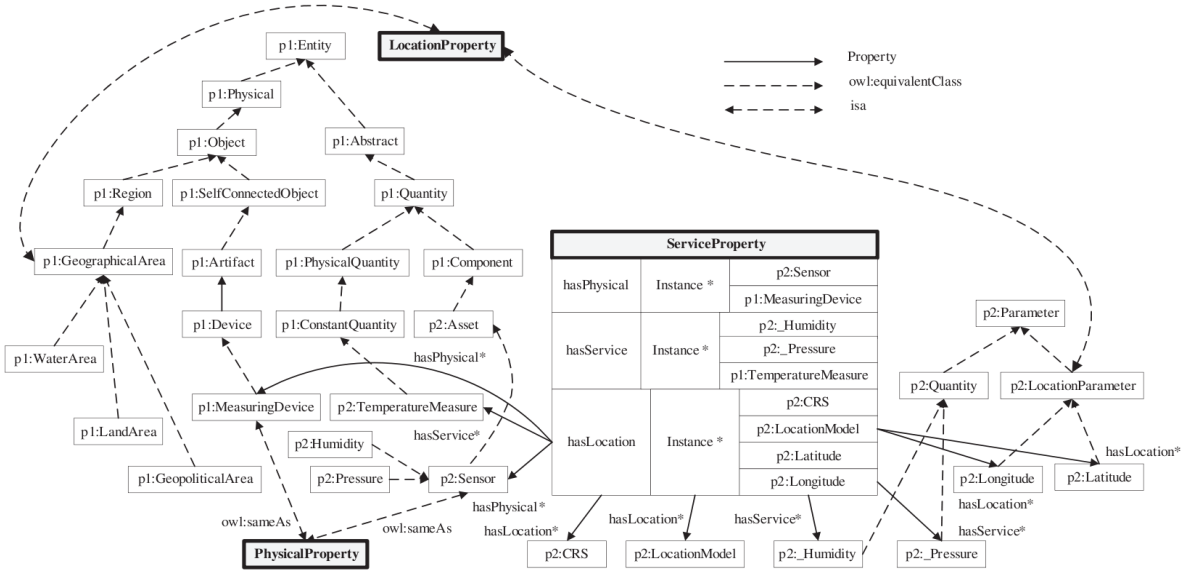

Figure 5. Improvement of OntoSensor ontology.

The objective of OntoSensor design was to faithfully replicate the concept hierarchy of SensorML in OWL. Some implementation compromises and workarounds needed to be made during the creation of ontology due to the dependency of certain SensorML terms on concepts from the Geographic Markup Language (GML) [GML, 2004].

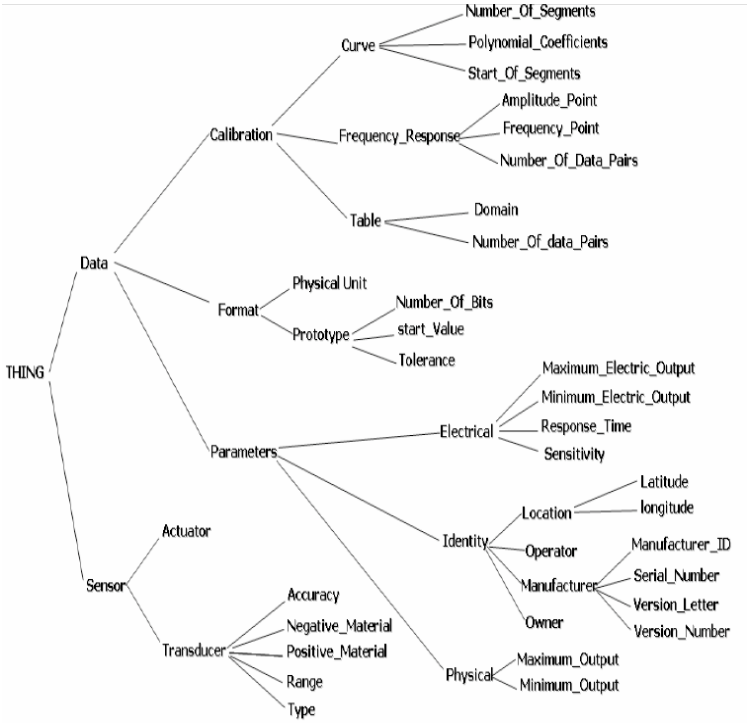

Figure 6. The part of SDO ontology.

OntoSensor extends the IEEE SUMO upper-level ontology [SUMO, 2001] by making some OntoSensor classes extensions of classes defined in SUMO. Furthermore, the SensorML specification references some concepts that are defined in ISO 19115. The ISO 19115 standard defines schema required for geographic information and services. Both the SUMO and ISO 19115 have OWL implementations that are referenced by OntoSensor. Figure 4 shows an aspect of the concept hierarchy of the OntoSensor ontology which includes formal definitions of SensorML concepts, extends IEEE SUMO and uses concepts from ISO 19115 in some of its relations. The OntoSensor ontology also contains the hierarchy of sensor devices. Generally, OntoSensor can be viewed as a middle-level ontology that extends the concepts from a high-level ontology (SUMO) and whose concepts can be used by more specialized ontologies that model specific domain sensors.

Certain SensorML concepts pertaining to the location model of sensors, coordinate reference systems and transformation procedures are dependent upon concepts from the Geographic Markup Language (GML). These SensorML concepts are necessary for specifying the location of sensors and sensor observations and to perform spatial transformations to relate these to the location and reference systems of other sensors, platforms and central monitoring and processing systems. Implementation of this dependency upon GML was suggested by referencing the OWL ontology with formal definitions of GML concepts created by [Defne, 2004].

The OntoSensor ontology was adopted as the basis for the design of the sensor ontology enabling service-oriented services in future computing [Kim, 2008]. The ontology design focuses the service-oriented interpretation of data sensed by the sensor. The main sources for collecting commonly used terms in the service domain were GML, SensorML, SUMO and OntoSensor. The novel sensor ontology was extended by the ServiceProperty, LocationProperty and PhysicalProperty classes representing the service-oriented properties. The meaning of the service-oriented property is sensing data, which is a difference to sensor data. The ontology is shown in figure 5.

5.1.3 Sensor Data Ontology (SDO)

The proposed universal SDO ontology comprises four components: the SUMO ontology, the Sensor Hierarchy Ontology (SHO), the Sensor Data Ontology (SDO), and the Extension Plug-in Ontologies (EPO) [Eid, 2006][Eid, 2007].The SHO and SDO ontologies reference and extend the SUMO ontology to facilitate automatic data fusion and inference in distributed and heterogeneous sensing environments.

The sensor hierarchy ontology includes knowledge models for the transducer (sensors and actuators) elements, data acquisition units, and data processing and transmitting units. It contains a hierarchy of transducer classes and describes its attributes and capabilities. The data model for a given transducer contains meta-data such as the measurement and/or output range, accuracy and type, as well as physical properties and calibration methods. A snapshot of the implementation of the transducer SHO is shown in figure 6.

The goal of the sensor data ontology is to describe the dynamic and observational properties of transducer data that goes beyond just describing individual transducers. The ontological model describes the context of a sensor with respect to spatial and/or temporal observations. Furthermore, the SDO utilizes the notion of virtual transducer as a group of physical ones to provide abstract measurements/operations. For instance, a temperature sensor, a humidity sensor, and a wind speed sensor may collectively monitor weather as weather sensors.

To extend the capabilities and behavior of the proposed universal ontology, the Extension Plug-ins Ontologies allow to integrate domain-specific ontologies with the universal ontology. Each plug-in ontology should implement the knowledge representation for a particular domain of sensor data and networks and establish the connection with the SUMO ontology. This enables interoperability and knowledge sharing among sub-ontologies in the ontology architecture.

5.1.4 Coastal Environmental Sensor Networks (CESN) Ontology

The purpose of the CESN ontology is to describe the relationships between sensors and their measurements. The main concepts found in the CESN sensor ontology are similar to the terminology described in SensorML [SensorML, 2007] and to some of those emerging in the Marine Metadata Interoperability device ontology project [MMI, 2009], and CSIRO Sensor Ontology [CSIRO, 2009a]. The core concepts in the CESN sensor ontology are the physical sensor devices themselves, Sensor; the PhysicalProperty that a Sensor can measure; and the measurement that a sensor has taken, PhysicalPropertyMeasurement, see figure 7. Not shown are important constraints, expressed in OWL, on this core. For example, a Sensor object can measure only one physical property. Objects that can contain Sensors and so measure more than one physical property are modeled by a class named Instrument. In turn, an instrument is usually deployed on some kind of Platform, which typically constrains its relationship to the environment in which it is deployed. Also not shown is the class Deployment, which represents the deployment of an instrument at a particular time and place,

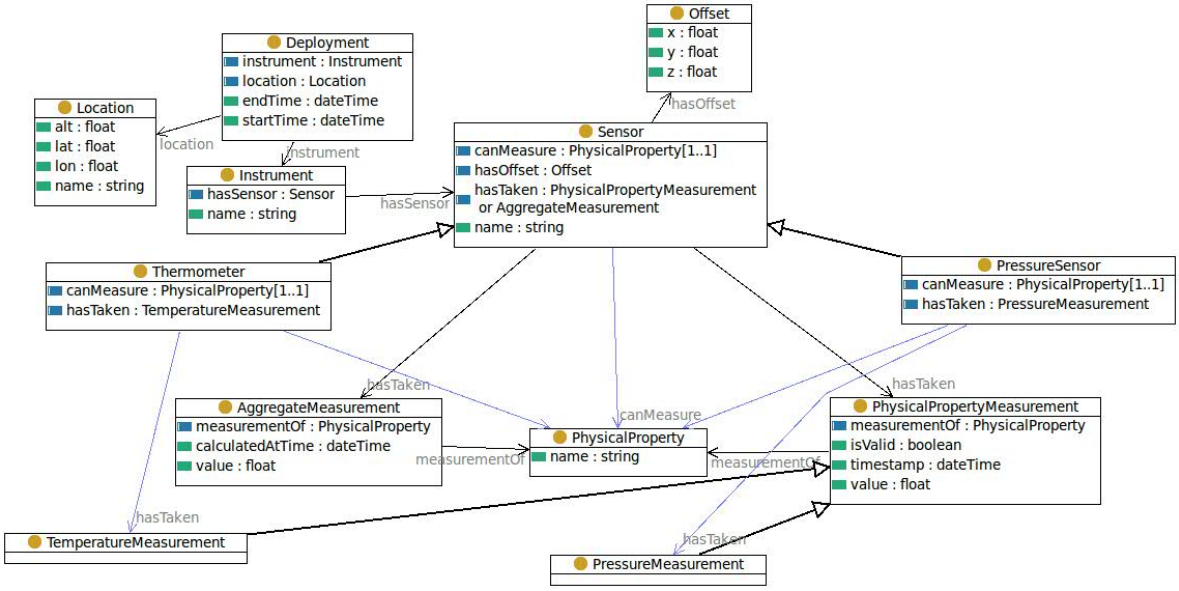

Figure 7. The core of CESN ontology.

and so can be used to relate instrument readings to expected or unexpected events putatively signaled by the data modeled by the ontology. Deployment attributes of individual instruments are particularly important in the real world of movable instruments.

5.1.5 Agent-based Middleware for MME (A3ME) Ontology

A3ME ontology specifies the rich classification of mixed mode environments (MME - environments with different dimensions of heterogeneity: heterogeneous devices, heterogeneous software, and heterogeneous communication technologies) [A3ME, 2008][A3ME, 2009]. The different kinds of devices were classified to allow mapping of a specific devices to a general class of devices with common characteristics. Also, the different capabilities were classified into groups of related capabilities, e.g. sensing capabilities, actuator capabilities, etc. These can than be further sub classified into more specific subtypes. The basic classification deals with different aspects needed to be classified in MME. Those are IDs, devices, capabilities, services, data, properties and other. Some of the concepts, such as devices, are further subclassified. The part of classification is shown in figure 8.

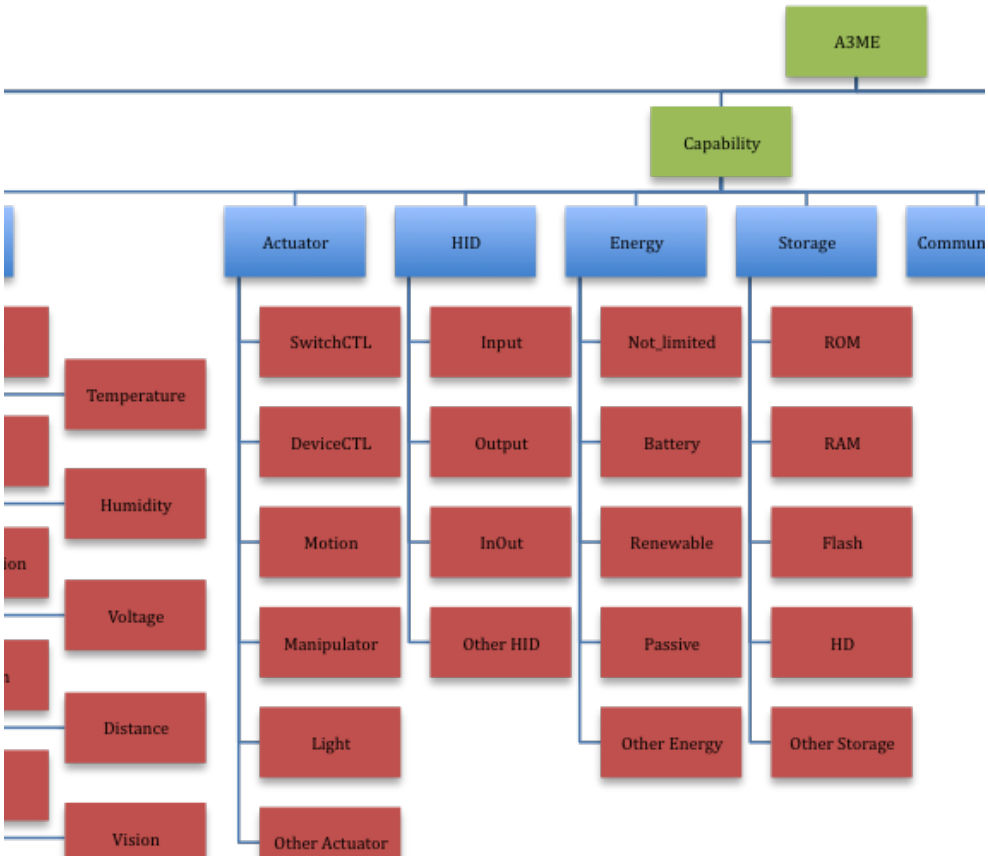

Figure 8. Part of A3ME classification.

5.1.6 Ontonym

Ontonym is the set of ontologies modelling the concepts of pervasive computing including uncertainty, provenance, sensing and temporal properties. Ontonym contains the ontologies describing devices, sensing, locations, people, provenance, events and time.

Ontonym's sensing ontology is concerned with the description of sensors and the data they generate. The characteristics and uncertainty of sensed data are represented by four properties: frequency, coverage, and a set of accuracy and precision pairs. Frequency is defined as the sample rate – how often the sensed data is updated. Coverage is defined as the amount of the potentially sensed context about which information is delivered. Precision defines a value range and accuracy is the percentage of how often the precision is achieved. Precision and accuracy can have different semantics depending on the sensor under consideration. Sensors may specify as many precision and accuracy pairs as required. Each sensor reading includes observation-specific information, meta-data characterising the observation, a reference to the sensor that generated the reading, a timestamp indicating when the observation was made, and more. The data associated with a reading is considered to be the union of the observation and sensor properties, with the observation properties taking precedence. This allows static properties of an observation to be specified as part of a sensor's properties.

The provenance ontology in Ontonym models three things: the creator or author of data, the time the data has been created or modified, and the source from which new data has been derived.

The event ontology provides a means of describing activities that have (at least) a temporal dimension and can be associated with the location. The event ontology also models roles that are played by entities in the activity, and the properties to associate an entity (person, device, etc.) with an event. All of these concepts are designed for extension to describe domain-specific events.

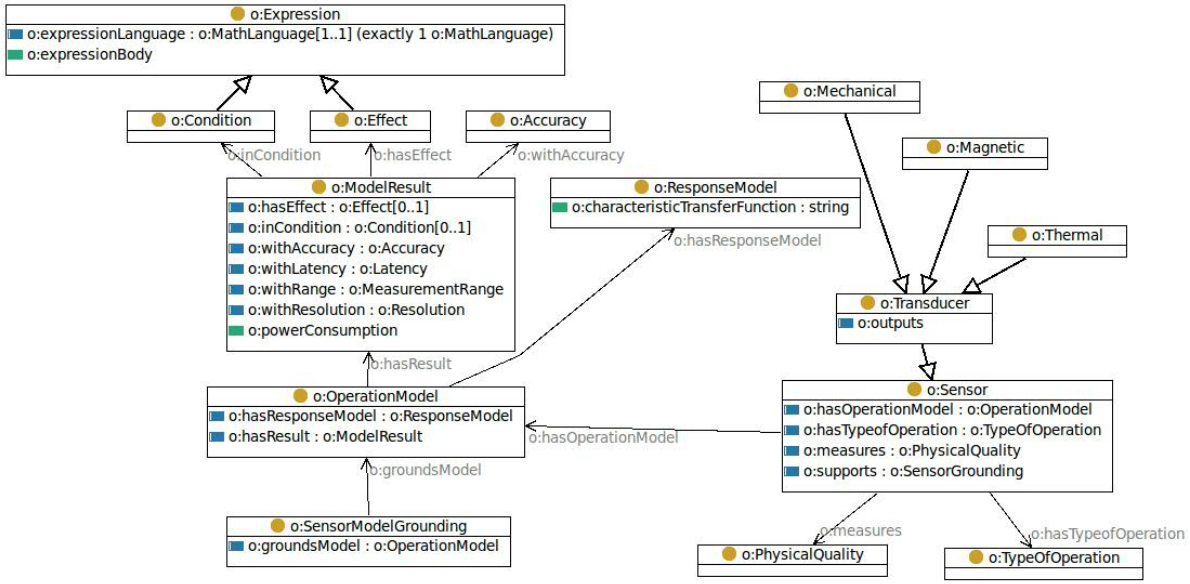

Figure 9. Part of CSIRO ontology.

5.1.7 CSIRO Sensor Ontology

CSIRO OWL ontology [CSIRO, 2009a][CSIRO, 2009b] was created as the basis for the semantic representation of sensors and as the formal description for reasoning about sensors and observations. While semantics can assist in searching for existing sensors, the CSIRO ontology was used to realize a more advanced task of automatic composition of a sensor satisfying a query if no such sensor exists. Once composed, such a sensor should be available as a sensor in its own right, and available to be used as a component of new virtual sensors. For example, a query such as "report event E each time condition C is reached in time period P ", requires finding or composing a sensor that can detect C, finding or making a process that builds an E from a C and understanding the constraints in availability, power, and eventual degradation of the sensors over P.

The ontology is built around the central notion of a sensor, and three important clusters of concepts referenced by the sensor: domain concepts, abstract sensor properties and concrete properties. The abstract properties specify sensors' functions and capabilities. The concrete properties ground the abstract by providing, for example, the interface details to the functions. It is essentially the difference between specifying the properties of a sorting algorithm and giving a path to a binary. Separating concrete and abstract aspects of sensors means that descriptions of types of sensors, functions and the like can be shared among specifications and also that a single sensor type can have multiple concrete descriptions, promoting reuse and allowing for differences in deployment. The domain ontology is left unspecified, any external domain ontology can be referenced.

Each concrete sensor has the sensor grounding models representing the concrete realisation of a sensor. The grounding represents, in the case of an instrument, its physical implementation, including size, shape, materials and location. The grounding also models the concrete aspects of accessing data from the sensor, including the types and expected formats of input when calling functions, the format of output and other details for accessing the sensor (radio, network or physical access, for example).

In general, each sensor may have any number of operation models describing the operations (functions) of the sensor, how the measurements are made and properties of the measurements. A response model represents a sensor's responses to stimuli under various conditions. Each operation model may have a number of results that specify properties such as the function that the operation computes (its effect), accuracy and latency under various conditions.

The core part of the ontology is shown in figure 9.

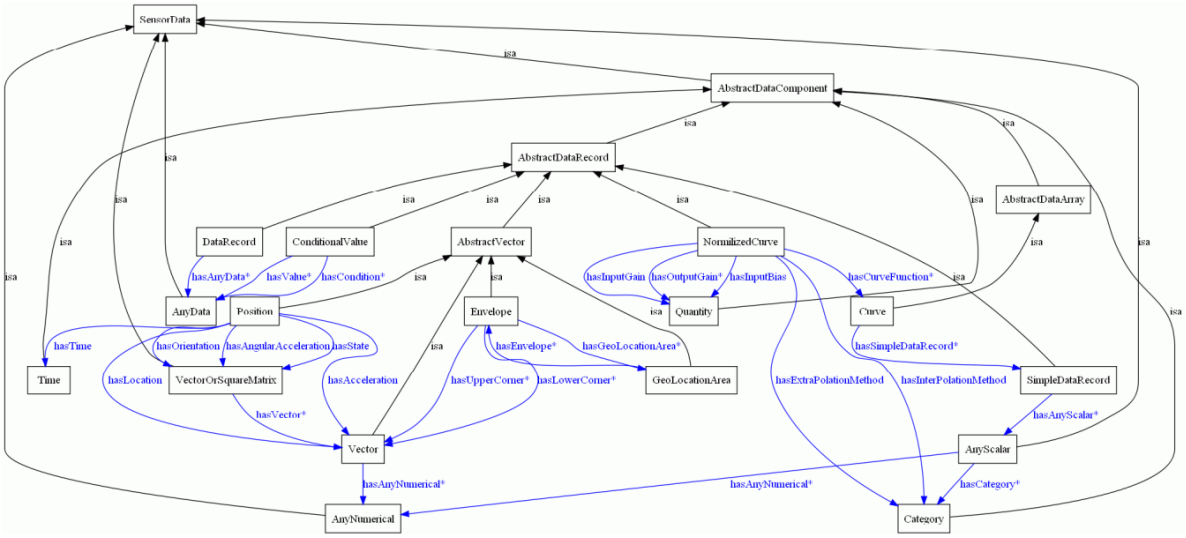

Figure 10. Part of Sensei ontology.

5.1.8 Sensei Observation and Measurement Ontology

The Sensei ontology [Sensei, 2009a][Sensei, 2009b] provides an ontology based approach to the structuring of data obtained from different types of sensors. Most of the current work on providing semantic data for sensor networks is focused on using semantic description for sensor nodes and elements which support advanced analytics and situation and context awareness in sensor networks. Creating a semantic sensor data model for sensor data related to measurements and observations is another important aspect in designing highly scalable and advanced heterogeneous sensor network applications. This should also support creating advanced data mining and knowledge extraction methods for real-time or archived sensor data. The Sensei ontology proposes a framework for a semantic data description model which provides interoperability and facilitates deriving additional knowledge from real-time and/or stored sensor data. The semantic data model in collaboration with a semantic sensor network architecture should support designing smart applications using sensor networks. A part of the Sensei ontology is illustrated in figure 10.

In [Sensei, 2009b] it is assumed, that observed and measured information will be transmited by the sensors in the semantic format. At first sight, it seems that the represented data in OWL form adds some complexity to the data representation structure and the extra information needs to be transmitted by the sensor nodes. Considering the fact that sensors nodes have limited processing and memory capabilities, the data representation could appear as a bottleneck to the design. To address this issue, the assumption is, that each sensor node will utilise a gateway or a similar solution to wrap the observation data in the particular data type which is measured by the sensor without requiring to be aware of the whole ontology structure. This means, that essentially the measurement and observation data from a sensor node will be in a format which complies with the Sensei ontology.

The data analysis and using ontology-based reasoning to extract additional knowledge from the data will only occur in processing nodes which have more powerful processing capabilities. The major cost of using the proposed method will be an increase in volume of the transmitted data from the sensor node.

5.1.9 Semantic Sensor Network (SSN) Ontology

The W3C Semantic Sensor Network Incubator Group provides a formal OWL DL ontology for modelling sensor devices (and their capabilities), systems and processes [SSN, 2010]. 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 than that, 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).

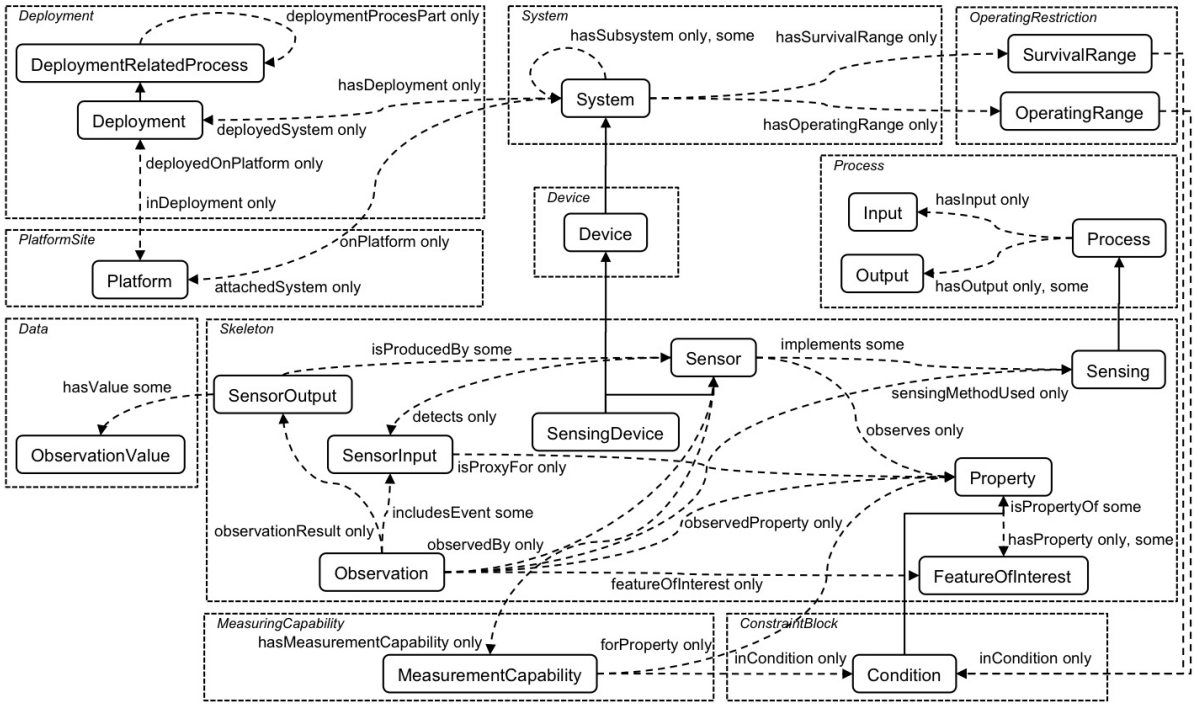

Figure 11. Architecture of SSN ontology.

In general, the sensors observe the stimuli to infer information about environmental properties and construct features of interest. The sensors themselves may have particular properties, such as an accuracy in certain conditions, or may be deployed to observe a particular feature, and thus the whole SSN ontology unfolds around this central pattern, which at its heart relates what a sensor detects to what it observes. The SSN ontology revolves around the central Stimulus-Sensor-Observation pattern, which acts as the upper core-level of the Semantic Sensor Network ontology. While the pattern is an integral part of the SSN ontology, it is also indented to be used separately as a common design pattern for 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. To ease the interpretation of the used primitives, to boost ontology alignment and matching as well as to facilitate reuse and interoperability, the pattern is aligned to the ultra light version of the DOLCE foundational ontology [DUL, 2007].

Several conceptual modules build on the pattern to cover key sensor concepts, such as: basic skeleton, devices, measuring capabilities and constraints, energy consumption, data, processess, operating restrictios, platforms, deployment and systems containing the sensors. The ontology does not include a hierarchy of sensor types; these definitions are left for domain experts, and for example could be a simple hierarchy or a more complex set of definitions based on the workings of the sensors. The modules contain the classes and properties that can be used to represent particular aspects of a sensor or its observations: 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. The overview of general SSN ontology structure is illustrated in figure 11.

5.2 Mapping between the relational databases and ontologies

5.2.1 Mapping approaches

Approaches to mapping between RDB and ontologies can be generally divided into the two basic categories: automatic mapping generation, vs. creation of domain specific mapping.

Many automatic mapping generation systems realise the automatic generation of mappings with the RDB table as a RDF or OWL class node and the RDB column names as RDF predicates [Virtuoso, 2007][D2RQ, 2007][SquirrelRDF, 2007]. Automatically generated mappings are often not able to capture the complex domain semantics that are required by many applcations, these mappings serve as a useful starting point to create more customized, domain-specific mappings. Even thought, there exist also approaches using existing domain ontologies as the basis for an automatic creation of the simple mappings. The simple mappings are checked for consistency and subsequently more contextual mappings are constructed [Hu, 2007].

Creation of domain specific mappings is often implicit or does not have to be captured at all in a RDB schema. Domain specific mappings often take advantage of existing domain models and lead to a drastic reduction of mapped triples, taking into account only relevant information contained in the domain model [Byrne, 2008]. Domain mapping is often realized as the process of domain ontology population, where data from a RDB is transformed into the instances of a particular domain ontology. Also many existing mapping tools, such as [D2RQ, 2007], allow users to define custom mapping rules in addition to automatically generated rules.

5.2.2 Mapping tools

D2RQ

The D2RQ [D2RQ, 2007] platform uses mapping to enable applications to access a RDF-view on a non-RDF database through the Jena and Sesame APIs, as well as over the Web via the SPARQL protocol and as Linked Data. The D2RQ consists of the D2RQ Mapping Language, a declarative mapping language for describing the relation between an ontology and an relational data model, the D2RQ Engine, a plug-in for the Jena and Sesame Semantic Web toolkits, which uses the mappings to rewrite Jena and Sesame API calls to SQL queries against the database and passes query results up to the higher layers of the frameworks and D2R Server. A HTTP server that can be used to provide a Linked Data view, a HTML view for debugging and a SPARQL protocol endpoint over the database. The mappings may be manually defined by the user, which allows the incorporation of domain semantics in the mapping process.

Virtuoso RDF View

The Virtuoso RDF View [Virtuoso, 2007] uses the "Table as RDFS Class and Column as Predicate" approach and takes into consideration special cases such as whether a column is part of the primary key or foreign key. The foreign key relationship between tables is made explicit between the relevant classes representing the tables. The RDB data are represented as virtual RDF graphs without physical creation of the RDF datasets. The Virtuoso RDF views are composed of quad map patterns that define the mapping from a set of RDB columns to triples. The quad map pattern is represented in the Virtuoso meta-schema language, which also supports SPARQL-style notations.

Triplify

Triplify [Auer, 2009] is a small plug-in for Web applications, which reveals the semantic structures encoded in relational databases by making database content available as RDF, JSON or Linked Data. It is based on the definition of relational database queries for a specific Web application. Triplify also demonstrates that a tool connecting a relational database to the world of semantics can be very lightweight. The approach does not support SPARQL, but it includes a method for publishing update logs to enable incremental crawling of linked data sources. Triplify is complemented by a library of

configurations for common relational schemata and a REST-enabled datasource registry. Despite its lightweight architecture Triplify is usable to publish very large datasets.

Datagrid

Dartgrid Semantic Web toolkit [Datagrid, 2006] provides the tools for the mapping and querying of RDF generated from RDB. The mapping is basically a manual table to class mappings where the user is provided with a visual tool to define the mappings. The mappings are then stored and used for the conversion. The construction of SPARQL queries is assisted by the visual tool and the queries are translated to SQL queries based on the previously defined mappings. A full-text search is also provided.

R2O

R2O [R2O, 2006] is a XML based declarative language to express the mappings between RDB elements and an ontology. R2O mappings can be used to "detect inconsistencies and ambiguities" in mapping definitions. The ODEMapster engine uses a R2O document to either execute the transformation in response to a query or in a batch mode to crate a RDF dump.

6. Prototype of event management support ontology

The ebbits platform aims to support interoperable business applications with context-aware processing of data separated in time and space, information and real-world events (addressing tags, sensor and actuators as services), people and workflows (operator and maintenance crews), optimisation using high level business rules (energy or cost performance criteria). The key requirements for the business rules execution is that the ebbits platform needs to be able to recognise and respond to physical world events. The information acquired from events from physical world generated by various devices create the basis for decision making at the several levels of the ebbits architecture, including basically:

- data fusion,
- situation patterns recognition,
- complex event processing,
- analysis of historical acquired data,
- and possibly more ..

All of above mentioned requirements needs to work with the large amount of information related to the devices generating the events or providing the services for further processing by event/service orchestration, decission or business rules. In some cases it must be possible to use this information to analyse the historical data generated by particular events.

As the description of devices, their services and events may change in time following the growing and changing requirements, the semantic model is the very flexible way, how to represent and query all the relevant information. For more, there already are existing models defining the properties of devices, services and events, which may be reused or adopted in a very flexible way, when using semantic technologies as the knowledge representation method.

Semantic knowledge representation may be realised using various languages and notations. To keep the semantic model as simple as possible and to be able to response to the queries in the real time, the good practice is to use the most simple language as possible, of course, with respect to the requirements on the knowledge represented. The OWL language [OWL, 2009] was selected as the knowledge representation technology, more precisely, the most simple dialect of OWL depending on the semantic engine (some semantic triplestores implement even more reduced expresivity of OWL, e.g. BigOWLIM enables to use OWL-Horst [BigOWLIM, 2010]). In the most cases it is possible to use the OWL-Lite dialect, which is recommended (until some higher expresiveness of the language is required).

All relevant models in ebbits are planned to be build upon the existing Hydra project ontologies (name changed from the HYDRA project) [HYDRA, 2010], which contain the rich description of devices, services and their various properties and capabilities, partly including the description of events. Based on the analysis of the state of the art in the area of existing sensor models, event-management systems and database – ontology mapping techniques, the Hydra ontologies have to be extended and aligned with the requirements on the scenarios of ontology usage. In the next section the requierements on the ontology usage will be outlined, the Hydra ontology will be briefly introduced and the relevant models of device, service and event will be focused and analysed. The Hydra ontology extensions based on the state of the art will be proposed and the preliminary semantic model will be designed.

6.1 Requirements on the ontology usage

The primary role of the semantic model is to answer the questions according to the semantics of the stored knolwedge. So, the main goal of the ontology design is to find and define the effective way of the knowledge representation. In the context of the workpackage, the main focus here is to design the effective representation of knowledge about the events and services provided by the various

devices. The necessary point, where to start, is the definition of meaning of the basic terms used in the scope of this deliverable:

- Service – is the functionality provided by the device, which must be proactively called by the agent using this service (e.g. getTemperature service of Thermometer sensor; agent may be the software using the service or the person)
- Event – is the functionality provided by the device, which autonomously publishes some data to some working environment (depending on the used event-management technology). This data is acquired from the environment (e.g. measurement data from sensor). The data acquisition is triggered by the occurence of some environment observation (e.g. the door lock sensor publishes the „door was closed" notification event, when this situation happens), or the data acquisition continually happens according to the defined frequency (temperature sensor publishes the measured temperature value each 5 seconds). In some cases, the functionality provided by the event may be also tied with the service, which may be proactively called, when needed (e.g. when the software agent or person needs to know the actual temperature in the room, it may call the getTemperature service, which will return the actual measurement value also provided by the particular event of device).

Of course, in the context of ebbits, there are also the concepts of higher level events, which represent the occurence of real-wold situations including also people or business events (such as meetings, occurence of situations started or caused by the people, complex situations representing the system states, etc.). The same counts for the services, which also may have a higher level interpretation (such as enterprise services, etc.). This interpretation of the events and services are the part of ebbits eventing architecture, but are out of scope of this deliverable and are planned to be covered in the WPs 3 and 6.

In most of the cases, it is required to answer the questions related to the devices, services and the events related information. The starting preliminary requirements were derived partly from the state of the art and partly from the assumed functionality to be provided by ebbits. Features, which has to be modelled can be outlined in the form of questions as follows:

- service properties:
  - what is the name and the (taxonomy) type of the service?
  - what are are the service inputs and outputs, including: names and datatypes of I/O parameters; units of I/O parameters?
  - what are the service capabilities (capabilities can extend the information about the service in more human readable way, but still standardized also for software agent, e.g. „measures temperature" or „sends SMS")?
- event properties:
  - what is the name and the (taxonomy) type of the event?
  - how is the event triggered? is it triggered by the particular stimuli (the occurence of some situation in the environment) or is it triggered in specified frequency?
  - if the event is triggered by the stimuli, what do we know about this stimuli?
  - what is the feature of interest of the event? what properties of the environment does the event observe?
  - what is the structure of the result returned by the event? what are the names, data types and units of the particular parts of this result?
  - where are the data acquired by this event stored and how can they be accessed?
  - what are the event-management infrastructure related properties of the event (e.g. in the case of subscriber-publisher architecture, the related properties could be: topics, where the event is published and the set of key-value pairs holding the published values)?

- which service is tied to the functionality provided by the event, if any?

- device context information – where is the device located and who owns the device?

The semantic model representing the above requirements will enable to answer also very complicated questions, such as: Which events flushed by devices in manufacturing hall observe the energy consumption of welding machines in units of Celsius?

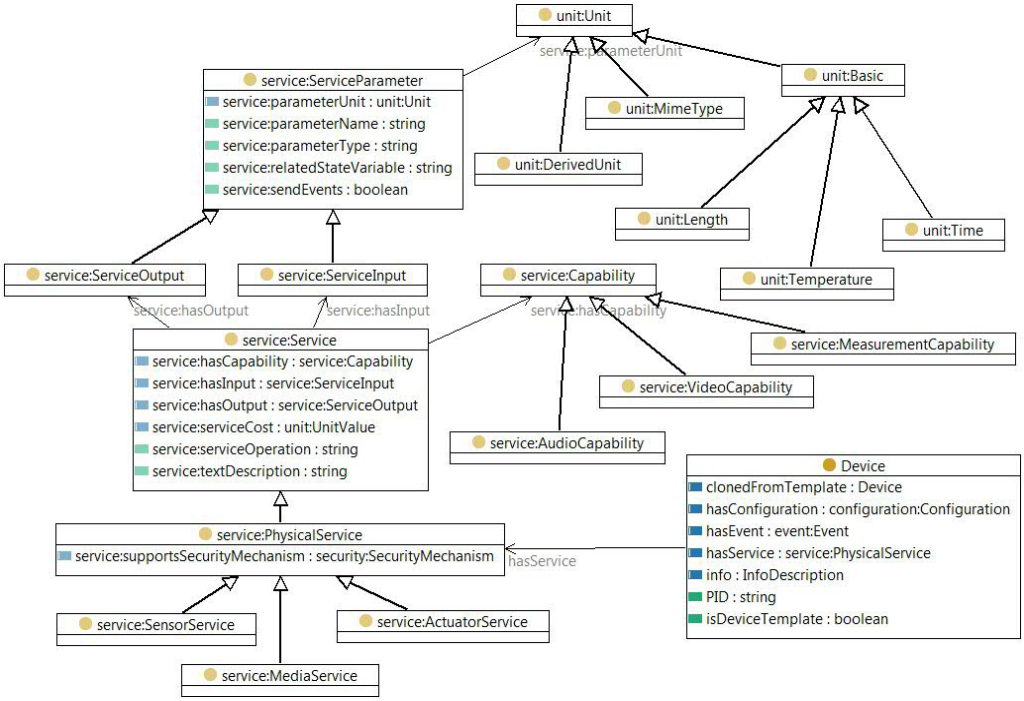

Figure 12. Service model of Hydra ontology.

6.2 Hydra ontologies

The Hydra device ontology represents the concepts describing device related information, which can be used in both design and run time. The basic ontology is composed of several partial models representing specific device information. The components of the Device Ontology can be shortly described as follows:

- core ontology: contains a taxonomy of various device types and the basic device description, manufacturer and model information.
- device capabilities: represent the hardware properties and software description
- device services: describes the models of device services in the terms of operation names, inputs and outputs. Services may have various capabilities and are also connected to the quality of service ontology used to annotate the services and their parameters to several quality factors.
- semantic discovery support ontology: containing all information relevant for semantic resolution of low-level device discovery information serving as the base for semantic device model identification
- energy profiles ontology: represents the device energy consumption related information.
- application ontology: contains the application domain dependent context information including models of people, ownership, locations, etc.
- events ontology: contains the full information about the events provided by device.
- semantic devices ontology: represent the logical aggregates of composed devices to provide more advanced application related functionality.

- security ontology: represents the various security properties, such as protocols, algorithms or objectives, which may be attached to devices or services.

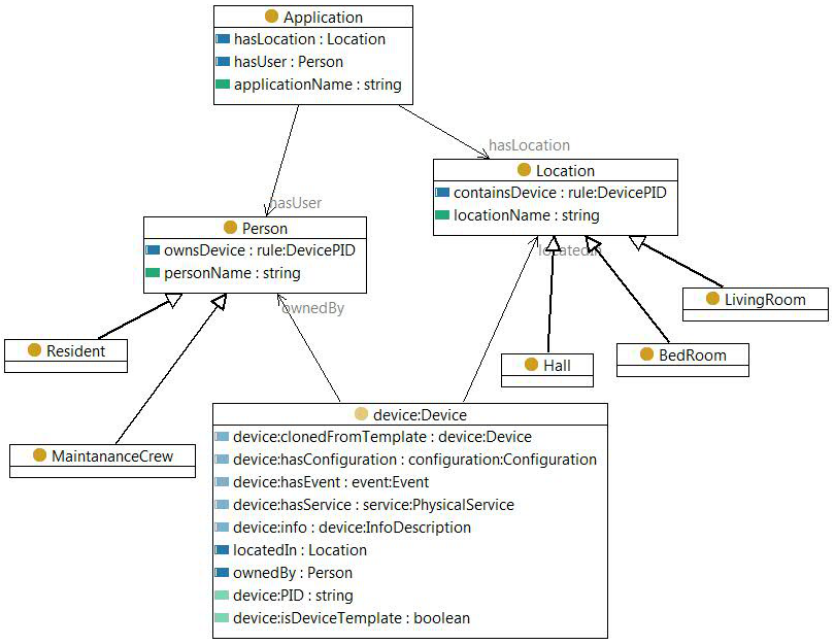

Figure 13. The application model of Hydra Ontology.

Hydra service model was created with the focus on the same requirements as listed above. Services are tied to devices, the have the description of I/O parameters including name, datatype and unit. Services may have attached more static capabilities represented in the related taxonomy. The semantic model for service composition or orchestration usually includes the richer description containing also the service execution preconditions and postconditions, the models for composition/orchestration processes or grounding to concrete implementation (such as in the well known OWL-S [OWL-S, 2004] or WSMO [WSMO, 2005] semantic service ontologies). As there is planned the dedicated service composition/orchestration framework implementation in WP7, there is no need for service processes modelling in the ontology and the prototype model of services will be reused from Hydra ontologies without extensions. The Hydra service model is illustrated in figure 12.

The problem is, how the event is represented in the Hydra ontology. The only reason for adding the events to the device description was to provide this information to the developer, when designing the application. Events could be possibly added just manually and the event model in Hydra ontology have only informative purpose.

Context information about the locations and ownership of devices is in Hydra ontologies handled by the application domain ontology following the requirements (device domain context is illustrated in figure 13).

6.3 Prototype of event management support ontology design

With respect to the actual semantic model provided by the Hydra, the most important extension is to provide the semantic model of events and all related information. Following the requirements on the ontology prototype, the model of event must be extended by: the core taxonomy of events, models of event results, model of event stimulus (e.g. the occurence of the real-world situation or the specified value of the frequency of measurements), event capabilities, knowledge about the data storage used to store the event results and possibly the extensions enabling semantic support of event-management framework.

The next sections will describe design of ebbits event ontology for particular requirements.

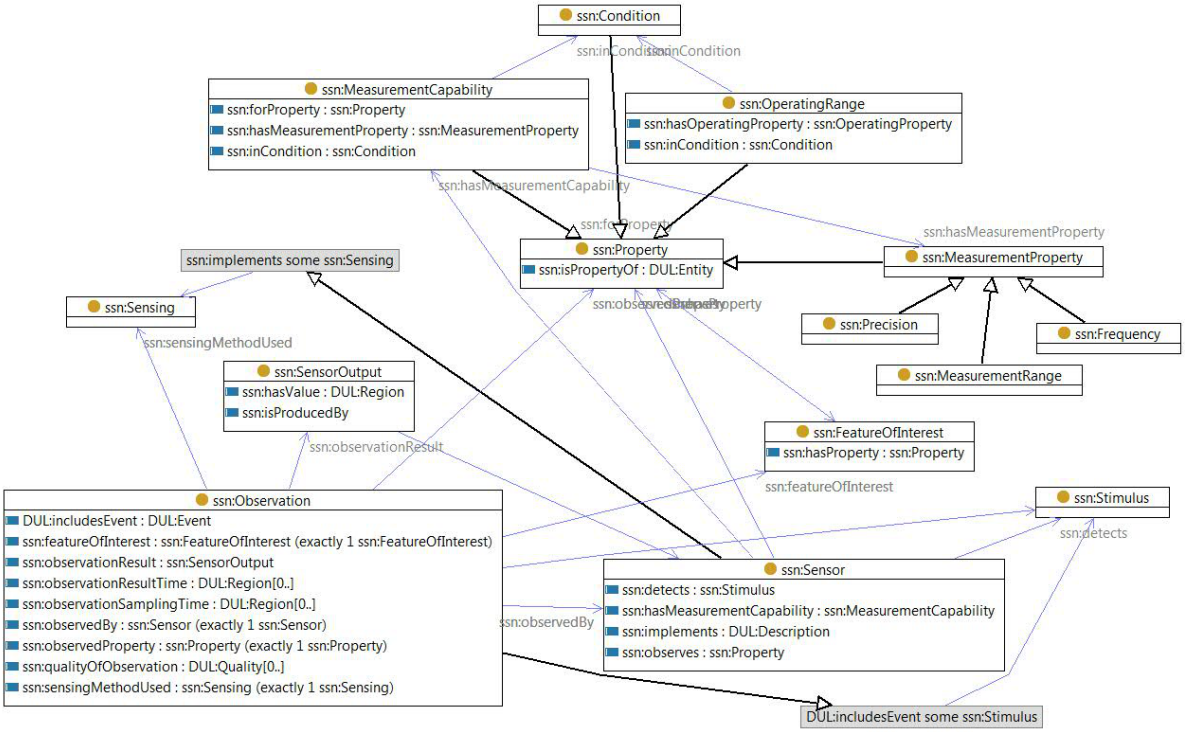

Figure 14. The core of SSN ontology.

6.3.1 Event model

The existing sensor models provide various approaches for the event interpretation, many times aligned with the application domain, where the particular model was used. Based on the analysis and the requirements, the SSN ontology [SSN, 2010] was selected as the best candidate to be used when designing the ebbits event model (the ontology core is illustrated in figure 14). There are the two main reasons for this selection:

- The scope of the model: SSN ontology design is based on the wide state of the art analysis from which were derived concepts and relations related to sensors in order to standardize knowledge in this area; and the SSN ontology contains models of all concepts and features relevant for ebbits event ontology the best matching the specified requirements.
- The activity status: SSN ontology is the product of W3C Semantic Sensor Network Incubator Group activity, which is productive and alive community updating reports, deliverables and models practically on the daily basis.

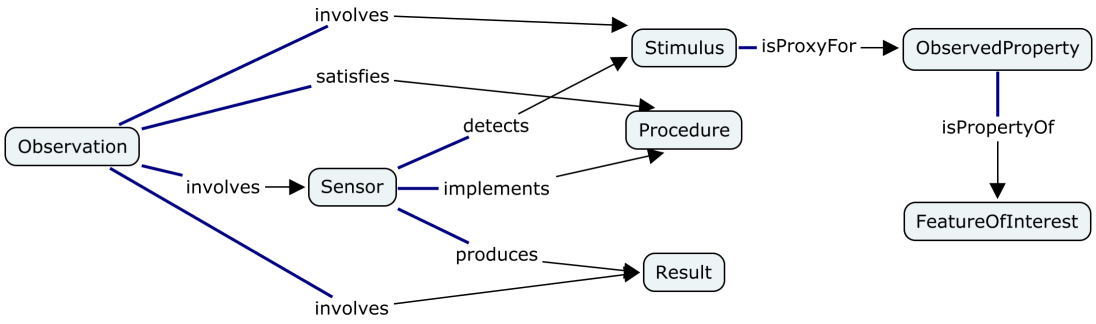

Figure 15. Stimulus-sensor-observation pattern

The most important starting step was to identify the relevant concepts in SSN ontology and to align the interpretation of this concepts to the ebbits event ontology. The SSN ontology skeleton is based on so-called Stimulus-Sensor-Observation pattern (see figure 15), which acts as the upper core-level

of the ontology. While the pattern is an integral part of the SSN ontology, it was also indented to be used separately as a common design pattern for all kind of sensor or observation based ontologies and vocabularies. Already from the first view of the model conception it is quite clear, that the representation is very related to the ebbits needs. Identified most relevant concepts and properties of SSN ontology and their interpretation are summarized as follows:

- *Sensor* represents any object with the sensing capability (e.g., in some cases a human observer can be a sensor). Sensor has following relevant properties:
  - *detects Stimulus*: Relation to the stimuli, which sensor can detect and which may trigger the sensor.
  - *observes Property*: Relation to the property, which sensor can observe. Each property is the quality of the real world. Observed properties may compose the stimulus, which triggers the sensor.
  - *madeObservation Situation*: relation between sensor and the observations, which were made. In materialized model, the observations are linked to instances of the Observation concept, which is the sub-class of the Situation concept.
  - *hasMeasurementCapability MeasurementCapability*: Relation to the measurement capabilities of the sensor.
- *Observation* ia s situation in which a sensing method has been used to estimate or calculate a value of a property of a feature of interest. Observation has the following relevant properties:
  - *observationResult SensorOutput*: Relation to the value produced by the observation.
  - *featureOfInterest FeatureOfInterest*: Relation to the feature of interest of the concrete observation.
  - *observedBy Sensor*: Relation to sensor, which realized the observation.
  - *observedProperty Property*: Relation to he properties, which can be observed via stimuli by a certain type of sensors. They inhere in features of interest and do not exist independently. While this does not imply that they do not exist without observations, the domain is restricted to those observations for which sensors can be implemented based on certain procedures and stimuli.
- *FeatureOfInterest* is an abstraction of the real world phenomena (thing, person, event, etc). Features of interest are entities in the real world that are the target of sensing. The relavant property here is:
  - *hasProperty Property*, which should take into account the relation between stimuli, observed properties and the properties, which are in the feature of interest of the sensing.
- *SensorOutput* is piece of information (an observed value) produced by the sensor observation. Relevant properties are:
  - *isProducedBy Sensor*: Which sensor produced the result.
  - *hasValue ObservationValue*: The result is a symbol representing a value as outcome of the observation. Results can also act as stimuli for other sensors and can range from counts and Booleans, to images, or binary data in general.
- *MeasurementCapability* collects together measurement properties (accuracy, range, precision, etc) and the environmental conditions in which those properties hold, representing a specification of a sensor's capability in those conditions. Relevant properties:
  - *forProperty Property*: A relation between some aspect of a sensing entity and a property. For example, from a sensor to the properties it can observe or from a measurement capability to the property the capability is described for.

- *hasMeasurementProperty MeasurementProperty*: Relations to the particular properties of the measurement. MeasurementProperty class it the root of the taxonomy of sensor properties, including e.g.: Accuracy, Frequency, Latency, Precission and many more.
- *inCondition Condition*: Relation to the measurement conditions used to specify ranges for qualities that act as conditions on a system/sensor's operation. For example, wind speed of 10-60m/s is expressed as a condition linking a quality, wind speed, a unit of measurement, metres per second, and a set of values, 10-60, and may be used as the condition on a *MeasurementProperty*, for example, to state that a sensor has a particular accuracy in that condition.

- *Stimulus* represents an event in the real world that triggers the sensor. The properties associated to the stimulus may be different to eventual observed property. It is the event, not the object that triggers the sensor. Documentation also says, that observed properties are the only connection between stimuli, sensors, and observations on the one hand, and features of interests on the other hand. As the Stimulus inherits all properties from super-classes, it is not quite clear, what is the connection between observed properties and the stimulus attached to the sensor.
- *Property* is an observable quality of an event or object. That is, not a quality of an abstract entity as is also allowed by DUL's *Quality*, but rather an aspect of an entity that is intrinsic to and cannot exist without the entity and is observable by a sensor.

In the context of ebbits, the concept of SSN Sensor corresponds to the concept of Device, which produces the low-level events; and the concept of SSN Observation best matches the Event concept in ebbits. In this context, the most of the properties tied with SSN Sensor concept, such as detected Stimulus or MeasuredCapability would radher belong to the Event concept in ebbits. The reason for this is to be able to specify all of this properties for particular device Events provided by the device separately. The SSN ontology is primarilly designed as the semantic model of sensors. In ebbits, the description of devices (including sensors) is much more general and the models of events are just the part of description of devices. In this sense, it seems, that it would be more easy to understand the interpretation of the model, if it would be clearly defined, that the each particular Event (not the sensor or device in this case) is triggered by some stimulus, observes concrete properties in the environment, produces the results and has specific measurement capabilities in the different circumstancies. The concepts designed in SSN are very similar to those to be used in ebbits event model, but the interpretation is different according to the purpose of the both models.

The SSN ontology is very large and generic (also because it is aligned with DUL upper ontology), the interpretation of concepts and relations differs, and the preliminary requirements on ebbits event model are not yet based on the real needs of the ebbits platform, but mainly on the expectations. In this case the better decistion for creating the preliminary event model in ebbits seems to be to take the SSN ontology as the inspiration instead of reusing the whole ontology. The model of events in ebbits takes the advantages of the concepts proposed by SSN ontology. Some of the concepts are simplified, some interpretation of relations is adapted to the needs of ebbits.

The parts of design of ebbits event ontology will be described in more details. The preliminary basic model of event is illustrated in figure 16.

The core taxonomy

The core taxonomy of events has two main sub-classes according to the form of the event stimulus, the taxonomy will be further extended in the future according to the devices and events used in the ebbits:

- *SituationTriggeredEvent* is triggered by the occurence of situation in the real-world. This type of event has linked the *Situation* class through *situationStimulus* property. The model of situation can be quite complex, as it can also contain the logical relations of certain atomic situations in the wold. As there is not enough precise specification, what this situation should describe, the model for this kind of stimulus is left empty for the prototype and should be specified in the future following the real cases.

- *PermanentlyTriggeredEvent* is not triggered by the occurence of situation, but continually produces the results in some frequency. This type of event has linked the *PermanentStimulus* class through the *permanentStimulus* property. The model, describing when and how the event is triggered, will be extended in the future according to the real cases.

As in some cases, the functionality provided by the event can be tied with the service, which may execute this functionality using the service call. In the ontology, the *Service* class is associated with the *Event* class using *executesEvent* property.

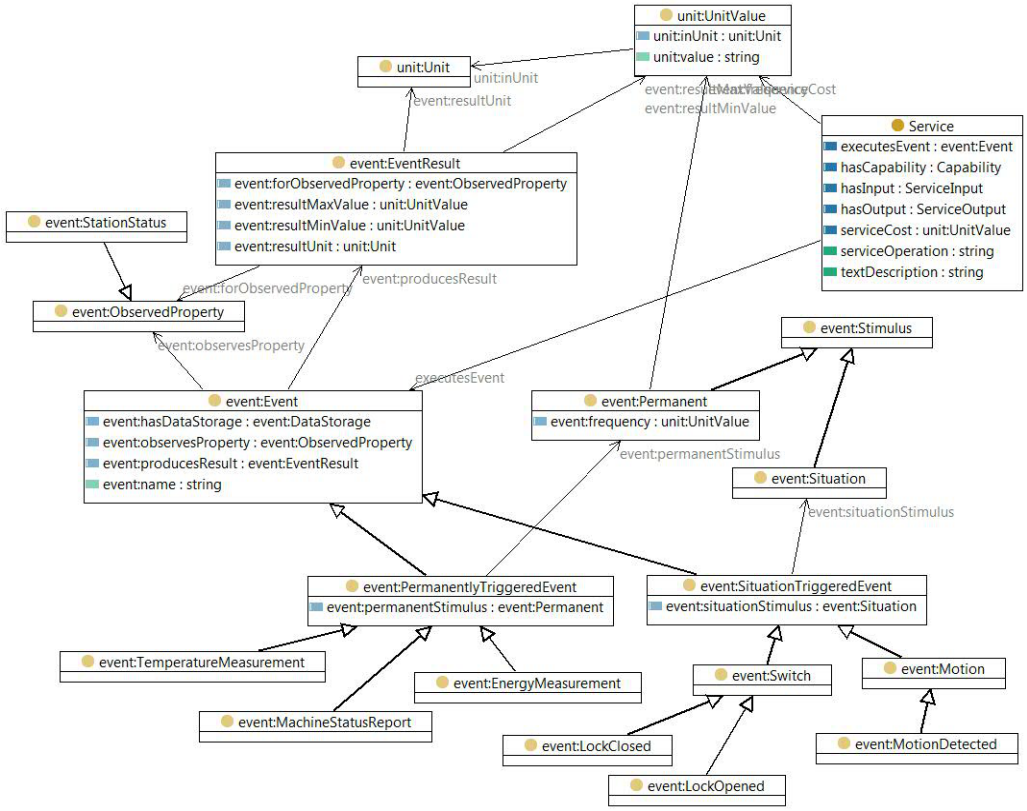

Figure 16. The ebbits event model design.

Observed properties

The observed properties represent the qualities of the world, which the device is capable to measure using the event. *Event* class can have multiple associations to the *ObservedProperty* class instances through the *observesProperty* relation. Observed properties serve as the feature of interest for the particular event.

Based on the analysis of use cases provided by the real-world data structure state of the art, it seems to be enough to provide the suitable representation of the observations in the form of static taxonomy describing, what is observed with respect to the application domain. This taxonomy can be further extended to satisfy the evolving requirements to the knowledge needed for proper decision making inside of the ebbits.

Observation result

The result of event is composed of the set of *EventResult* instances produced by the sensor and linked to the *Event* class through the *producesResult* property. The instances of *EventResult* class may have the association to the *ObservedProperty* instances attached to *Event* through the *forObservedProperty* relation. Using this relation it is possible to tie the produced results to the particular observed qualities of the real-world. In addition, each event result specifies the unit of the result and may also specify the maximum and minimum expected value, each also tied to the unit of the result. The unit specified for the result, maximum and minimum value must be the the same to keep the interpretation of the resulting value consistent.

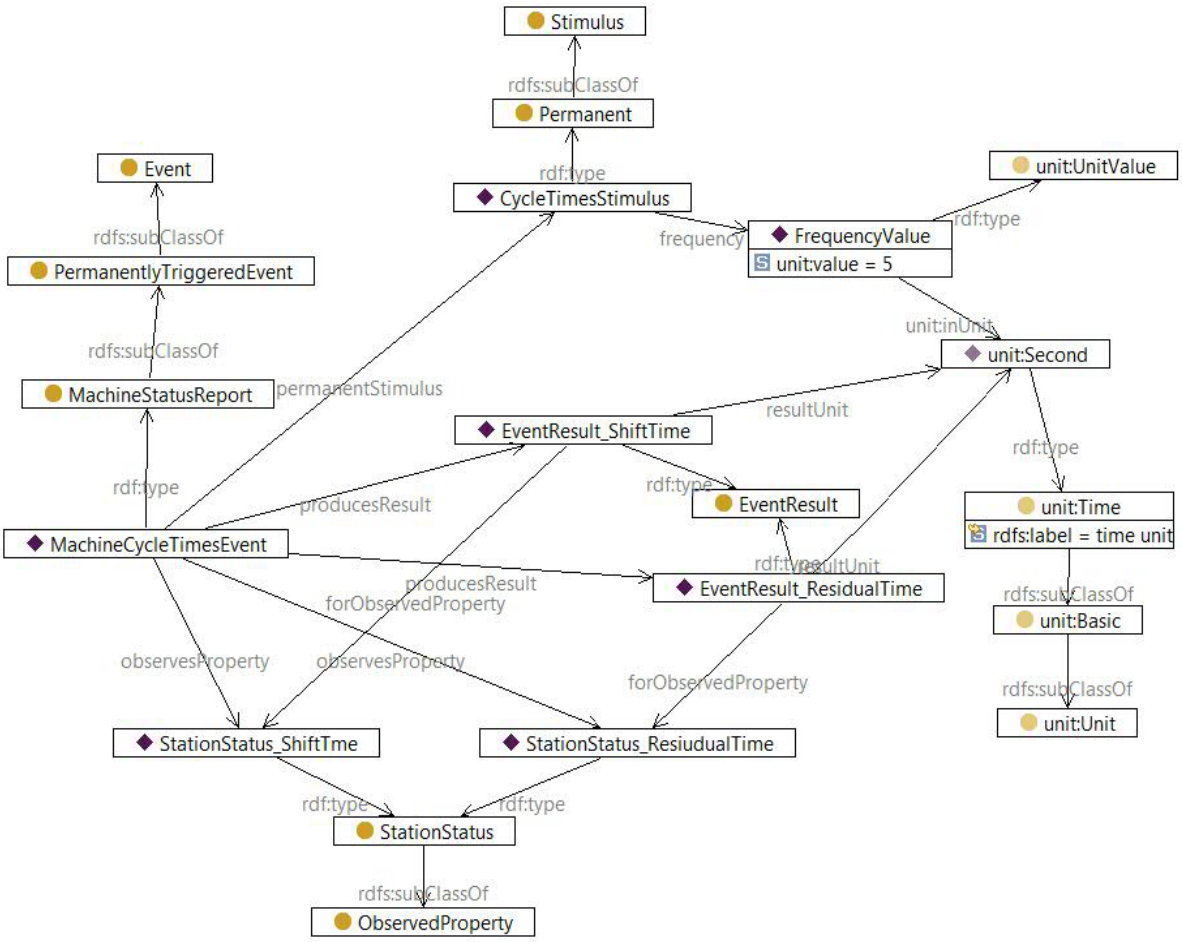

Figure 17. Example of semantic model of the concrete event.

Based on the analysis of use cases provided by the real-world data structures state of the art, the data in events are in both manufacturing and agriculture domain expected to be the basic data types, such as booleans or strings. Anyway, the state of the art also proposes the usage of higher level standards for the result description, such as Common Information Protocol or Common Information Model used within the ZigBee WSN standards. In this stage of the design this standards are just the proposition to be investigated in the future. Thus, the preliminary design of the event ontology represents the event results in the very basic way using only the basic data types. Example of model of event handling the two observations of the machine status in manufacturing domain is illustrated in figure 17.

The real values produced by the event are not modelled in the ontology. The reason is, that the ontology serves only as the model describing the particular events. The values, which are produced are caught by the ebbits infrastructure and ontology serves only as the decision support component providing all information relevant for the events. Anyway, sometimes the results produced by events and the events themselves are stored and have to be retrieved e.g. for the purposes of historical data analysis. Description, how this information is represented will be provided in the next section.

6.3.2 Semantic support of event management framework

In most of the cases, the basic benefit of semantic for underlying event management architecture used, is to be able to answer the questions or to retrieve all required information supporting decision making processes. The semantic support will be implemented in the form of services provided by the semantic infrastructure, which would be universally used by any component in the ebbits. In some cases, the semantic support will have to be implemented on the level of event management architecture depending on the selected technology (e.g. in the form of build-in predicates used in the EPL queries of Esper framework; or by the dedicated services used when handling events provided by standard Java JMS technology; etc.).

6.3.3 Event data storage and access support

In some cases, the results produced by the events must be stored in order e.g. to perform analysis using the historical data. This leads to the requirements on the semantic model to be able to provide all necessary information about where and how data are stored and how could they be retrieved. The most important issue here is, that data from some physical storage, such as relational database, should not be replicated in the ontology. It is required to be able to access the stored data natively from its storage. In the ideal scenario, the ontology would represent only basic metadata about the data storage location and access information; and character of data. In case of relational database, the location and access information would be the connection string and database name and description of metadata of used tables. This information can be tied with the specific instance of the event. Once the historical data are needed, the ontology is asked for required storage information and internal ebbits infrastructure would enable to access and retrieve the data using the ontology representation.

In order to design the appropriate knowledge representation of data storage, the technologies for mapping of relational databases to semantic models were investigated. The D2RQ framework seems to be one the best candidates for the mapping in the case, when the data storage is the relational database. D2RQ enables to specify the database connection properties and to design the custom mapping rules beween the database meta-data and the ontology model. Using this mapping, the SPARQL query on the ontology is translated to the SQL queries and executed against the database. The results are serialized into the standardized SPARQL result. When using the D2RQ mapping, it is enugh to extend the event model with the mapping file created using D2RQ rules. As the mapping file can be serialized into the ontology notation language, the model can be easily extended. The D2RQ engine is required for query execution.

An alternative approach, especially when using the different storage than relational database would be to provide the whole model of stored data attached to the event and then implement the custom component inside of the ebbits infrastructure, which would use the semantic representation of the data, automatically translate it into the query specific for the data storage and process the results. The model describes the database connection, the tables used and the meta-data for each particular table. This information would be enough to be able to compose the SQL query quite easily. This approach would work for the simple cases, when the data are stored in the single table for the event. The extension for joining the data from various tables must be further investigated, when some use case would require it.

6.3.4 Future work

This deliverable proposes only the preliminary semantic model of low-level events produced by the devices as the result of specific environment observations. The requirements for the model design were derived only from the state of the art focused on existing models and technologies; and from the requirements specified based on the assumptions and lessons learned from previous experiences. The appropriate real use cases for using the event models were not available in this phase of design. The most of future work will be focused on analysis and implementation of real use cases and requirements, so the model will evolve with changing requirements.

Another very important issue is to increase the interoperability of the ebbits ontologies. The narrower integration of ebbits ontologies with existing models (such as SSN ontology or upper ontologies, such as DUL) should be considered, if possible.

7. References

[A3ME, 2008] Herzog, A., Jacobi, D., Buchmann, A., A3ME - An Agent-Based Middleware Approach for Mixed Mode Environments. In The Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2008), 2008.

[A3ME, 2009] Herzog, A., Buchmann, A., Predefined Classification for Mixed Mode Environments. Technische Universitat Darmstadt, 2009

[Auer, 2009] Auer, S., et. al., Triplify-Lightweight Linked Data Publication from Relational Database, In proceedings of WWW 2009, Madrid, Spain, 2009.

[BigOWLIM, 2010] , OWLIM – Semantic Repository for RDF(S) and OWL, Ontotext AD, OWLIM Primer, 2010

[Blackstock et al 2010] Michael Blackstock, Nima Kaviani, Rodger Lea, Adrian Friday: MAGIC Broker 2: An Open and Extensible Platform for the Internet of Things. Internet of Things (IOT), 2010, Tokyo, Japan, Pages: 1-8, 2010

[Byrne, 2008] Byrne, K., Having Triplets – Holding Cultural Data as RDF, Proceedings of the ECDL 2008 Workshop on Information Access to Cultural Heritage, Aarhus, Denmark, 2008.

[CIP, 2001] CIP: Not Just Another Pretty Acronym, Rockwell Automation Inc, July 2001

[CIP, 2010] Common Information Protocol CIP, http://www.odva.org/Home/ODVATECHNOLOGIES /CIP/tabid/65/lng /en-US/language/en-US/Default.aspx, ODVA, 2010

[Casimiro et al 2007] Antonio Casimiro, Jörg Kaiser, Paulo Verissimo: Generic-Events Architecture: Integrating Real-World Aspects in Event-Based Systems. Architecting Dependable Systems Lecture Notes in Computer Science, 2007, Volume 4615/2007, Pages: 287-315

[CSIRO, 2009a] Neuhaus, H., Compton, M., The Semantic Sensor Network Ontology: A Generic Language to Describe Sensor Assets. In AGILE Workshop Challenges in Geospatial Data Harmonisation, 2009.

[CSIRO, 2009b] Compton, M., Neuhaus, H., Taylor, K., Tran, K., Reasoning about Sensors and Compositions. In Proceedings of the 2nd International Workshop on Semantic Sensor Networks (SSN09) at ISWC 2009, 2009.

[D2RQ, 2007] Bizer, C., Cyganiak, R., D2RQ — Lessons Learned, Position paper for the W3C Workshop on RDF Access to Relational Databases, Cambridge, USA, 2007.

[Datagrid, 2006] Wu, Z., et. al., Dartgrid: a Semantic Web Toolkit for Integrating Heterogeneous Relational Databases, Semantic Web Challenge at 4th International Semantic Web Conference (ISWC 2006), Athens, USA, 2006.

[Defne, 2004] Defne, A., Islam, A., Piasecki, M., Ontology for Geography Markup Language (GML3.0) of Open GIS Consortium (OGC), 2004

[DUL, 2007] Laboratory for Applied Ontology, DOLCE-UltraLite, 2007, available at: http://wiki.loa-cnr.it/index.php/LoaWiki:DOLCE-UltraLite

[Eid, 2006] Eid, M., Liscano, R., El Saddik, A., A Novel Ontology for Sensor Networks Data. In Proceedings of 2006 IEEE International Conference on Computational Intelligence for Measurement Systems and Applications, 2006.

[Eid, 2007] Eid, M., Liscano, R., El Saddik, A., A Universal Ontology for Sensor Networks Data. In 2007 IEEE International Conference on Computational Intelligence for Measurement Systems and Applications, 2007.

[Esper] Esper - Complex Event Processing.", http://esper.codehaus.org/, retrieved 02/2011.

[Etzion, 2011] Etzion, O. and P. Niblett (2011). Event Processing in Action. Stamford, Manning Publications Co.

[Eugster et al, 2003] Eugster, Felber, Guerraoui, Kermarrec, The Many faces of Publish/Subscribe, ACM Computing Surveys, Volume 35 , Issue 2 table of contents, Pages: 114 - 131, 2003

[IEC, 2003] IEC 61132-3 ed2.0, Programmable controllers - Part 3: Programming languages, IEC, Jan 2003

[IEC, 2007] An Introduction to IEC 61970-301 & 61968-11: The Common Information Model, University of Strathclyde, January 2007

[IECa, 2009] IEC 61970 Energy Management System Application Program Interface (EMS-API) - Part 301: Common Information Model (CIM) Base, 2009

[IECb, 2009] IEC 61968 Application integration at electric utilities - System interfaces for distribution management- Part 11: Common Information Model (CIM), 2009

[IECc, 2009] IEC 61968 Application integration at electric utilities – System interfaces for distribution management – Part 9: Interfaces for meter reading and control, IEC, 2009

[ISO, 1996] ISO 11787:1995, Machinery for agriculture and forestry. Data interchange between management computer and process computers. Data interchange syntax, ISO, January 1996

[ISO, 2000] ISO 11788-3:2000, Electronic data interchange between information systems in agriculture. Agricultural data element dictionary. Pig farming, ISO, July 2000

[GML, 2004] Cox, S. et. al., OpenGIS® Geography Markup Language (GML) Implementation Specification , OpenGIS Recommendation Paper OGC 03-105r1, 2004

[Hu, 2007] Hu, W., Qu., Y., Discovering Simple Mappings Between Relational Database Schemas and Ontologies, In Proc. of 6th International Semantic Web Conference (ISWC 2007), 2nd Asian Semantic Web Conference (ASWC 2007), LNCS 4825, Busan, Korea, 2007.

[HYDRA, 2010] Kostelnik, P., et. al., D6.9 Device Ontologies Maintenance Tools Prototype Update, Hydra Project Deliverable, IST project 2005-034891, 2009

[Jacobsen et al, 2009] Hans-Arno Jacobsen, Vinod Muthusamy, and Guoli Li: The PADRES event processing network: Uniform querying of past and future events. it - Information Technology, 51(5)250-260, 5 2009.

[Jacobsen et al, 2010] Hans-Arno Jacobsen, Alex Cheung, Guoli Li, Balasubramaneyam Maniymaran, Vinod Muthusamy, and Reza Sherafat Kazemzadeh: The PADRES Publish/Subscribe System., IGI Global, 2010.

[Kaiser et al, 2005] Jörg Kaiser, Cristiano Brudna, Carlos Mitidieri: COSMIC: A real-time event-based middleware for the CAN-bus. Journal of Systems and Software Volume 77, Issue 1, July 2005, Pages: 27-36

[Kim, 2008] Kim, J-H, et.al.Building a Service-Oriented Ontology for Wireless Sensor Networks, ICIS '08 Proceedings of the 7th IEEE/ACIS International Conference on Computer and Information Science, 2008

[MMI, 2009] MMI Sensors Ontology: A Community Development Project., Marine Metadata Interoperability, 2008.

[ODVA, 2011] ODVA, http://www.odva.org/Home/tabid/53/lng/en-US/language/en-US/Default.aspx [OEE, 2011] OEE, http://www.oee.com/

[OntoSensor, 2006]Goodwin C., Russomanno, D.J., An Ontology-Based Sensor Network Prototype Environment, Fifth International Conference on Information Processing in Sensor Networks (Poster), IEEE, Nashville , 2006

[OntoSensor, 2005] Russomanno, D.J., Kothari, C., Thomas, O., Building a Sensor Ontology: A Practical Approach Leveraging ISO and OGC Models, The 2005 International Conference on Artificial Intelligence, Las Vegas , NV, 2005

[OWL, 2009] W3C OWL Working Group, OWL 2 Web Ontology Language Document Overview, W3C Recommendation, 2009, available at: http://www.w3.org/TR/owl2-overview/

[OWL-S, 2004] Martin, D., et.al., OWL-S: Semantic Markup for Web Services, W3C Member Submission, 2004, available at: http://www.w3.org/Submission/OWL-S/

[Padres, 2011] PADRES: A Reliable Publish/Subscribe Middleware: http://padres.msrg.toronto.edu/Padres/, retrieved 02/2011

[R2O, 2006] Barrasa, J., Gómez-Pérez, A., Upgrading relational legacy data to the semantic web, In Proc. of 15th international conference on World Wide Web Conference (WWW 2006), Edinburgh, United Kingdom, 2006.

[SensorML, 2007] Botts, M. et. al., OpenGIS® Sensor Model Language (SensorML) Implementation Specification , OpenGIS Implementation Specification OGC 07-000 , 2007

[Sensei, 2009a] Wei, W., Barnaghi, P., Semantic annotation and reasoning for sensor data. In EuroSSC'09: Proceedings of the 4th European conference on Smart sensing and context, 2009.

[Sensei, 2009b] Barnaghi, P., Meissner, S., Presser, M., Sense and sensability: Semantic data modelling for sensor networks. In Proceedings of the ICT Mobile Summit 2009, 2009.

[SEP, 2010] Smart Energy Profile 2.0 Application Protocol Specification, ZigBee Alliance and HomePlug Powerline Alliance liaison , April 2010

[SquirrelRDF, 2007] Seaborne, A., et. al., SQL-RDF, Hewlett-Packard Development Company, 2007, available at: http://www.w3.org/2007/03/RdfRDB/papers/seaborne.html

[SSN, 2010] Lefort, L., Henson, C., et. al., Incubator Report, W3C Semantic Sensor Network Incubator Group, 2010, available at: http://www.w3.org/2005/Incubator/ssn/wiki/Incubator_Report

[SUMO, 2001] Niles, I. Pease, A., Origins of the Standard Upper Merged Ontology: A Proposal for the IEEE Standard Upper Ontology, In Working Notes of the IJCAI-2001 Workshop on the IEEE Standard Upper Ontology, Seattle, WA, 2001.

[TNM, 2009] Larsen, C. Event and Alarm-data Exchange Specification - Revision 0.4. March 2009.

[Virtuoso, 2007] Blakeley, C., RDF Views of SQL Data (Declarative SQL Schema to RDF Mapping), OpenLink Software, 2007.

[WSMO, 2005] Lausen, H., et.al.,. Web Service Modeling Ontology (WSMO). W3C Member Submission, 2005, available at: http://www.w3.org/Submission/WSMO/.

[ZigBee, 2007] ZigBee Cluster Library Specification, ZigBee Alliance, October 2007

[ZigBee, 2010] ZigBee Home Automation Public Application Profile , ZigBee Alliance, February 2010