Enabling the business-based Internet of Things and Services

(FP7 257852)

D4.3 Coverage and scope definition of a semantic knowledge model

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 control page

Document file: D4.3 Coverage and scope definition of a semantic knowledge model
Document version: 1.0
Document owner: Jan Hreno (TUK)

Work package: WP4 – Semantic Knowledge Infrastructure
Task: T4.5 – Knowledge creation analysis
Deliverable type: P

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
0.0	Karol Furdik (IS)	2011-04-10	Initial TOC
0.1	Martin Knechtel (SAP), Riccardo Tomasi (ISMB) and Mauricio Caceres (ISMB)	2011-05-09	Reorganization and first draft of section 3, updated subsections 3.3, 3.4 and 3.5
0.2	Mauricio Caceres (ISMB)	2011-05-09	Updates in subsections on requirements of ebbits use cases
0.3	Peter Kostelnik (TUK)	2011-05-16	Chapters 4, 5, and 6
0.4	Jan Hreno (TUK), Peter Kostelnik (TUK)	2011-05-21	Updates in chapters 4, 5, and 6
0.5	Karol Furdik (IS)	2011-05-23	Introduction, Executive summary
0.6			
0.7			
1.0	Jan Hreno, Peter Kostelnik, Karol Furdik	2011-05-29	Final corrections
		2011-05-31	Final version submitted to the European Commission

Internal review history:

Reviewed by	Date	Summary of comments
Ferry Pramudianto (FIT)	2011-05-27	Result of internal review
Matts Ahlsen (CNet)	2011-05-27	Result of internal review

Index:

1. Executive summary 4
2. Introduction 5
2.1 Purpose, context and scope of this deliverable 5
2.2 Background 5
3. Semantic aspects and requirements of ebbits use cases 6
3.1 Review of current use cases in ebbits 6
3.2 Requirements for semantic annotation of resources 8
3.3 Requirements for accessing semantic information 11
3.4 Examples of annotation and access of semantic resources 15
3.4.1 Examples in the Automotive Manufacturing domain 15
3.4.2 Examples in the Food Traceability domain 16
4. Survey of semantic knowledge models usage within the ebbits project 19
5. HYDRA semantic use cases and ontologies 21
5.1 HYDRA ontologies 21
5.2 Using the semantics in HYDRA 22
5.2.1 HYDRA enabling device 23
5.2.2 Semantic device discovery 23
5.2.3 Extending the device semantic description 24
5.2.4 Application context-awareness 25
5.2.5 Semantic devices 25
6. Proposed HYDRA ontology extensions for ebbits use cases 27
6.1 Ebbits specific semantic use cases 27
6.1.1 Business decision models for enterprises 27
6.1.2 Multi-sensory data fusion and context-awareness 27
6.1.3 Event management and service orchestration 28
6.1.4 Resource annotation and knowledge retrieval 28
6.2 Extension of HYDRA ontologies for ebbits 29
6.2.1 Reusing the HYDRA ontologies 29
6.2.2 Ebbits specific ontologies 29
7. Conclusion 30
8. References 31

1. Executive summary

This deliverable concludes the T4.5 task of the project and can be seen as a starting point towards the implementation of semantic structures required by ebbits pilot applications.

A detailed analysis of specified ebbits use cases with respect to the requirements on underlying semantic knowledge model is provided in chapter 3. The requirements, which are organised in tables of Volere style, are further supported by a set of examples that indicate the means for OWL annotation and access of envisioned semantic resources.

The proposed semantic infrastructure, which should cover enterprise, production, and device levels of enterprise structure, will be designed and implemented within several ebbits workpackages. The mapping of required semantic information types to the respective workpackages is provided in chapter 4.

The overall semantic knowledge model of ebbits will be based on the HYDRA ontology, which is described in chapter 5. Finally, proposed updates and extensions of the HYDRA ontology, as they were invoked by ebbits use cases, are summarised in section 6.

2. Introduction

2.1 Purpose, context and scope of this deliverable

The purpose of this deliverable is to specify the envisioned semantic knowledge model with regard to the capabilities required by ebbits use cases. The specification is focused on a provision of all the information that can be needed for further implementation of semantic infrastructure in various ebbits workpackages. It namely includes the analysis of initial use case descriptions and extraction of requirements that are relevant for the development of semantic structures. The specified set of requirements, properly prioritised and categorised, defines a scope of the resulting semantic knowledge model and can then serve as a validation framework for the developers / modellers during the implementation.

The coverage of ebbits semantic infrastructure is determined by the investigations of available technologies such as semantic stores, reasoning engines, and knowledge representation formalisms, which were provided in previous deliverables of WP 4. The HYDRA ontology[1] was selected as the main semantic resource on which the ebbits knowledge model will be developed. The directions and proposed means of HYDRA ontology extensions towards the ebbits use cases, as well as a distribution of development works between particular workpackages, is provided in the next sections of this deliverable.

2.2 Background

This deliverable is an outcome of the Task 4.5, which is specified in the ebbits Description of Work as follows:

**"*This task will analyse the required scope and coverage of the semantic model, specifically for the use cases in ebbits. Semantic interoperability of devices and information systems' resources needs a common defined terminology. One way to provide this is to adhere to standard interaction protocols and data formats. In fields, where such a standardisation does not exist since interaction mechanisms and architecture is in its innovative structure not yet covered by existing standards, a shared semantic model helps out.*

Most often, an ontology is used to store a formal representation of a shared conceptualisation. ebbits needs a semantic model in order to allow for semantic interoperability. In this task we will analyse the required scope and size of the semantic model in order to prepare its creation systematically. Based on the of knowledge representation formalism analysis carried out, IS will propose coverage and scope definition of the semantic knowledge model, inputs and comments will be provided by TUK. ISMB will contribute to the definition of solutions enabling semantic interoperability between physical devices and information systems."**

[1] The consortium of the HYDRA project decided to change the trademark name of the produced middleware and the related ontologies to the "LinkSmart" middleware / ontology. The details of this decision can be found at http://www.hydramiddleware.eu. Since in this deliverable we are referring to the ontology as it was originally designed in the HYDRA project, we are using the "older" name here.

3. Semantic aspects and requirements of ebbits use cases

The ebbits platform will allow different stakeholders in an enterprise domain to cooperate by sharing and correlating their own information with others in order to provide traceability, reasoning and context awareness capabilities in production environments such as manufacturing and agriculture. To do so, a proper modelling of the information, stakeholders and elements involved has to be performed, taking into account the knowledge representation and reasoning adopted in ebbits.

Deliverable D4.1 introduced a state of the art on semantic stores and reasoning engines, giving an opening overview of their possible implementation in the ebbits. In addition, deliverable D4.2 presented in detail the state of the art of knowledge representation formalisms, i.e. the standards used to describe semantic relationships in triple store repositories.

This section in particular, will focus on how the proposed use cases for the ebbits platform (introduced in deliverables D2.1 and D3.1) can be analysed in order to get some requirements for the proper knowledge modelling and annotation of semantic resources.

3.1 Review of current use cases in ebbits

Deliverables *D2.1 Scenarios for Usage of the ebbits Platform* and *D3.1 Enterprise Use Cases* introduced some preliminary use scenarios for the ebbits interoperability platform in two different application areas: Automotive Manufacturing and Food Traceability. Also in deliverable *D2.4 Initial Requirements Report* a first iteration on the user requirements is presented in the above application areas, and the possible utilisation of triple stores in relevance with the user requirements is discussed in table 8 of the same deliverable.

Such use cases can be grouped according to the specific domain they are proposed, in which some subdomains have been already identified:

- **Use cases in Automotive Manufacturing**
  - **Usage stories (Appendix 1 of D2.1)**
    - **6.2.1.1 Monitoring and storing data**
    - **6.2.1.2 Checking the status of the manufacturing plant**
    - **6.2.1.3 Sending alert to Mario**
    - **6.3.1.1 Production optimisation**
  - **5.2.1 Energy reduction management process (Appendix 1 of D3.1)**
    - **5.2.1.1 Initial assessment of energy consumption**
    - **5.2.1.2 Develop a strategy for energy reduction**
    - **5.2.1.3 Decision about improvement of the device**
    - **5.2.1.4 Intervention on the device**
    - **5.2.1.5 Monitoring of the performed modification**
    - **5.2.1.6 Decision about additional improvement**
  - **5.2.2 Plant production shut down and restart (Appendix 1 of D3.1)**
    - **5.2.2.1 Shut down of the plant production**
    - **5.2.2.2 Restart of the plant production**
  - **5.2.3 Automatic energy reduction (Appendix 1 of D3.1)**
    - **5.2.3.1 Decision about the energy reduction**
  - **5.2.4 Retrieve production information (Appendix 1 of D3.1)**
    - **5.2.4.1 Filling in a form about an accident**
    - **5.2.4.2 Collect the production data**
- **Use cases in Food Traceability**

Usage stories (Appendix 2 of D2.1)
- **8.2.1.1 Information about delivered feed**
- **8.2.1.2 Calculation production and delivery plan**
- **8.2.1.3 Feedstuff delivery**
- **8.2.1.4 Invoke information about delivered feed**
- **8.2.2.1 Implementation and activation of RFID tag**
- **8.2.2.2 Storage information about animals**
- **8.2.2.3 Monitoring of feeding and identification of deviations in behaviour of animals**
- **8.2.2.4 Special production**
- **8.2.2.5 Prediction of delivery to slaughterhouse**
- **8.2.3.1 Animal transportation to slaughterhouse**
- **8.2.4.1 Receiving information about the meat**
- **8.2.4.2 Identification of potentially infected meat packages**

6.2.1 Field area (Appendix 2 of D3.1)
- **6.2.1.1 Analyse nutrients in soil (optional)**
- **6.2.1.2 Analysis of the history of the land use**
- **6.2.1.3 Decision about the type of crops**
- **6.2.1.4 Decision about fertilization**
- **6.2.1.5 Decision about harvesting time**

6.2.3 Farm production (Appendix 2 of D3.1)
- **6.2.3.1 Check conditions for production**
- **6.2.3.2 Decision about insemination**
- **6.2.3.3 Check heat**
- **6.2.3.4 Replace a sow**
- **6.2.3.5 Vaccinate**
- **6.2.3.6 Wean sows**
- **6.2.3.7 Medical intervention**
- **6.2.3.8 Selling pigs to slaughterhouse**

6.2.4 Slaughterhouse (Appendix 2 of D3.1)
- **6.2.4.1 Specification of payment to the farm**
- **6.2.5.1 Order specification**

- **General use cases (domain/sector independent) (Section 4 of D3.1)**
  - **Order goods or services**
  - **Send Invoice**
  - **Record data to the accounting system**
  - **Send Payment**
  - **Request for information (RFI)**

These use cases will be used for drafting some semantic aspects: requirements for annotation and access to semantic information (triple stores) in the ebbits platform. The following subsections present these requirements in a Volere fashion.

3.2 Requirements for semantic annotation of resources

No	Summary	Rationale	Source	Fit Criteria	Priority	Requirement Type	Components	User Satisfaction	User Dissatisfaction
0	All stakeholders should be annotated with unique Id, type, name and relevant info.	It is important to recognize who is interacting with the system, and which privileges and restrictions has.	ALL USE CASES	One or multiple directories of stakeholders.	1	Functional		Neutral	Very high
AUTOMOTIVE MANUFACTURING DOMAIN									
A1	Monitored/sensed data should be contextualized (timestamp, geotag, type, etc).	It is important to know when and where data was sensed/monitored.	6.2.1.1 Monitoring and sorting data 6.3.1.1 Production optimisation	Semantic store with knowledge model for sensor readings.	2	Functional		Low	High
A2	Monitored/sensed data should be annotated (semantically) in local server/repo/store.	Information relationships should be available as soon as data enters the ebbits system.	6.2.1.1 Monitoring and sorting data 6.3.1.1 Production optimisation	Data acquisition, annotation and storing policy.	2	Functional		Low	High
A3	Alerts should be contextualized (timestamp, geotag, type, message, warning level, etc).	Generated messages and alerts need to be traceable and provide rich information about the event detected.	6.2.1.3 Sending alert to Mario	Alerting and messaging policy.	2	Functional		Low	High
A4	Devices should be annotated with id, type, name, location, and current/historical data (status, work in progress, consumables levels, quality record, energy consumption, energy profile, planned/unplanned intervention/maintenance, fault info, etc).	Another added value that ebbits could introduce in enterprise domains is efficiency tracking, which requires a monitoring and log of several metrics in devices/tools/machinery and resources in general.	5.2.1.1 Initial assessment of energy consumption 5.2.1.4 Intervention on the device 5.2.4.1 Filling in a form about an accident	Semantic store with knowledge model for devices.	2	Functional		High	Low

A5	Production should be annotated or modelled in order to calculate OEE and get number of orders/elements/products requested/delivered/in-progress/faulty.	Real-time traceability of produced goods/services is achieved by properly annotating their status and metrics during the manufacturing process.	5.2.3.1 Decision about energy reduction 5.2.4.2 Collect the production data	Semantic store with knowledge model for production.	1	Functional		High	Low
A6	Logistic should be annotated or modelled in order to get information about element and consumables (present, in transit from supplier, ordered, etc).	The ebbits platform could provide also a system for efficient management of consumables and logistic aspects needed in (not only) manufacturing domains.	5.2.4.2 Collect the production data	Semantic store with knowledge model for logistic.	2	Functional		Low	Neutral
FOOD TRACEABLITY DOMAIN									
F1	Feedstuff should be annotated with origin, genetics, treatment, storage conditions and transport/delivery info (batch number, silo id, amount, timestamp).	A detailed annotation of feedstuff is required to the reasoning processes devised.	8.2.1.1 Information about delivered feed 8.2.1.3 Feedstuff delivery	Semantic store with knowledge model for feedstuff.	1	Functional		High	Very high
F2	Animals should be annotated with RFID tag, weight, genetics, birth date, and current/historical (timestamped) data (growth/weight, location/movements, consumed feed, water, weaning, insemination, heat during pregnancy, born piglets, anomalies, vaccines/treatment/medication).	Proper identification of animals and logging the most relevant information about their lives is vital for the traceability and quality control proposed in ebbits.	8.2.2.1 Implementation and activation of RFID tag 8.2.2.3 Monitoring of feeding and identification of deviations in behaviour of animals 8.2.2.5 Prediction of delivery to slaughterhouse 8.2.3.1 Animal transportation to slaughterhouse 6.2.3.2 Decision about insemination & 6.2.3.3 Check heat 6.2.3.5 Vaccinate 6.2.3.6 Wean piglets 6.2.3.7 Medical intervention 6.2.3.8 Selling pigs to slaughterhouse	Semantic store with knowledge model for animals. Logging policy.	1	Functional		High	Very high
F3	Meat packages should be annotated with ID of animal (for trace).	Meat traceability is one of the main added values of the ebbits platform in the agricultural domain.	8.2.4.2 Identification of potentially infected meat packages	Semantic store with knowledge model for animals.	2	Functional		Low	High

F4	Farm's soil/fields should be annotated with location, laboratory analysis info (date, sample field source, lab id/name, results, etc), current/historical data (types of crops, grain maturity, soil nutrients, quality of products grown, etc).	Different reasoning applications devised in ebbits for tracking the soil efficiency require a detailed annotation and logging of farms' soil.	6.2.1.1 Analyse nutrients in soil 6.2.1.2 Analysis of the history of the land use 6.2.1.5 Decision about harvesting time	Semantic store with knowledge model for farm soil/fields.	2	Functional		Low	Neutral
F5	Farm's repository should store information about harvesting equipment, man power, etc.	The ebbits platform would provide also some functionalities for the management of resources needed for harvesting, thus they need to be included in the knowledge model.	6.2.1.5 Decision about harvesting time	Semantic store with knowledge model for farm resources.	2	Functional		Low	Neutral
F6	Sow farm production should be annotated or modelled in order to allow tracking the number of piglets, pigs at fertile age, pigs ready to slaughter, maximum capacity, etc.	By proper reasoning and processing, the ebbits platform can exploit the knowledge in the network and extract aggregated information required in real time.	6.2.3.1 Check conditions for production	Reasoning algorithms for production tracking.	1	Functional		Neutral	Neutral
F7	Halves of slaughtered pigs should be annotated with id, id of slaughtered pig, weight, fat thickness, date of slaughter, price, etc.	Traceability of meat requires proper tracking of pigs since birth to stores, thus the information after its slaughter is very relevant.	6.2.4.1 Specification of payment to the farm	Semantic store with knowledge model for halves of slaughtered pigs.	1	Functional		High	Neutral
ENTERPRISE DOMAIN									
E1	Order should be annotated with type/ID of good/service, amount, price, dates(issue, expiry, delivery, etc).	Ebbits platform can be exploited also for generic enterprise processes, like account management.	Order goods	Semantic store with knowledge model for orders.	3	Functional		Neutral	Neutral
E2	Invoices should be annotated with supplier's info (name, id, bank account, contacts, etc), goods or services info (type/id, amount, price, dates, etc).	Ebbits platform can be exploited also for generic enterprise processes, like account management.	Send invoice & Record data to the accounting system	Semantic store with knowledge model for invoices.	3	Functional		Neutral	Neutral
E3	Payment reports should be annotated with responsible id/name, bank, order number, status, etc.	Ebbits platform can be exploited also for generic enterprise processes, like account management.	Send payment	Semantic store with knowledge model for payments.	3	Functional		Neutral	Neutral

E4	Generic Information should be annotated with requester, sender, content (price, capacity, dates), etc.	Information exchanged between stakeholders could be exploited for some reasoning, thus it is convenient to model it semantically.	Request for information	Knowledge model for information exchange.	3	Functional		Neutral	Neutral

3.3 Requirements for accessing semantic information

No	Summary	Rationale	Source	Fit Criteria	Priority	Requirement Type	Components	User Satisfaction	User Dissatisfaction
0	Stakeholders should be stored in local catalogues or external directories (advisory company, chamber of commerce, etc) and accessed by the different subsystems inside and outside ebbits.	In order to apply access control lists/policies, all stakeholders must be identified.	ALL USE CASES	Stakeholder directories.	1	Functional		Neutral	Very high
AUTOMOTIVE MANUFACTURING DOMAIN									
A1	Manufacturing Monitor System should have write access to local server/repo/store.	The reasoning processes devised in ebbits require access to knowledge/information found in local stores.	6.2.1.1 Monitoring and sorting data.	Access rules/policy granted for MMS.	1	Functional		Low	High
A2	Manufacturing System for Analysis should have read/write access to local server/repo/store.	Analysis/reasoning is based on local monitored data, and reports are sent back to local server/repo/store.	6.2.1.1 Monitoring and sorting data.	Access rules/policy granted for MMS.	1	Functional		Low	High
A3	Reports should have a list of allowed readers/subscribers.	Aggregated data, reports, alerts, etc, should be available only to stakeholders interested in them.	6.2.1.2 Checking the status of the manufacturing plant.	Access rules/policy for generation/reading of reports.	2	Functional		High	Very high

A4	ebbits platform should have a list of alerts and subscribers.	The different monitored processes in ebbits should generate alerts and send them to the interested subsystems or stakeholders.	6.2.1.3 Sending alert to Mario	Directory of alerts/events.	2	Functional		Neutral	High
A5	Manufacturing Monitor System must have read access to internal and external environment data.	The reasoning processes devised by ebbits for the manufacturing domain require environmental monitoring.	6.3.1.1 Production optimisation	Internal/external sensors collected and annotated in respective repositories.	3	Functional		Low	Low
A6	Maintenance Manager and operators should have access to devices' and production info (proper ACL have to be implemented)	Some of the added values that ebbits will provide to managers in the manufacturing domain require a continuous tracking of the production processes, metrics and modifications introduced.	5.2.1.1 Initial assessment of energy consumption 5.2.1.5 Monitoring of the performed modification 5.2.1.6 Decision about additional improvement 5.2.3.1 Decision about the energy reduction 5.2.4.1 Filling in a form about an accident 5.2.4.2 Collect the production data	Stakeholders access policies for readings, devices and production.	1	Functional		High	Low
FOOD TRACEABLITY DOMAIN									
F1	Farm's Management System should have access (through secure connection) to Feed Provider Resources Monitoring System.	The food traceability scenario requires an exchange of information between all the enterprises involved in the production chain.	8.2.1.1 Information about delivered feed 8.2.1.2 Calculation production and delivery plan 8.2.1.4 Invoke information about delivered feed	Authenticated/secure access to Feed Provider RMS.	2	Functional		Neutral	Low
F2	Feed Provider should transfer delivery information about sent feedstuff.	The traceability relays on the successful exchange of information about the monitored processes linked to the tracked product.	8.2.1.3 Feedstuff delivery	Interface/procedure for request/receive/exchange of relevant information about deliveries.	1	Functional		High	Neutral
F3	Farm's Local server/repo should be accessible by RFID tag readers and National servers/repos/stores.	Information about animals is stored in farm local servers and used to retrieve information when reading RFID tags or when requested by National/European authorities.	8.2.2.2 Storage information about animals	Access policies to local server.	1	Functional		Low	High

F4	Farm's Monitoring System should have access to local server/repo/store.	Monitoring system keep track of several process in the farm and needs the metadata stored in local server.	8.2.2.3 Monitoring of feeding and identification of deviations in behaviour of animals 8.2.2.5 Prediction of delivery to slaughterhouse	Access policies to local server.	1	Functional		Low	Neutral
F5	Farm's Management Application Server should access local monitoring system servers/repos/stores for generation of reports.	Management/accounting systems perform their tasks based on the information stored/provided by local monitoring systems.	8.2.3.1 Animal transportation to slaughterhouse	Access policies to local monitoring system.	2	Functional		Low	Neutral
F6	Consumer should have access to meat reports.	Relevant reports about the produced meat since the piglet born will be used by ebbits in order to enhance the information provided to consumers about the meat they are buying.	8.2.4.1 Receiving information about the meat	Access policies to production reports.	2	Functional		High	Neutral
F7	Controller should have access to (online) queries to meat packages IDs.	Quality and health control authorities can rely on ebbits in order to track the distribution of meat packages when they discover some anomaly.	8.2.4.2 Identification of potentially infected meat packages	Access policies to production reports.	1	Functional		Low	Very high
F8	Farm's Management System should have access to field info repository.	Optimisation of the resources, like fields in an agricultural domain can be achieved by analyzing and processing information logged by their respective monitoring systems.	6.2.1.2 Analysis of the history of the land use	Access policies to historical info by Farm's MS.	2	Functional		Neutral	Low
F9	Farm's Management System should have access to external information (crop price, fertilizers price, consumables price, weather, etc).	Consumables information can be exploited through ebbits in order to manage efficiently the farm's production processes.	6.2.1.3 Decision about the type of crops 6.2.1.4 Decision about fertilization 6.2.1.5 Decision about harvesting time	Access to external/providers information.	3	Functional		Low	Low
F10	Sow Farm Management System should have access to production/animal repository.	The knowledge obtained by tracking all production processes in the farm will allow managers to optimise them through a single platform.	6.2.3.1 Check conditions for production 6.2.3.2 Decision about insemination 6.2.3.3 Check heat 6.2.3.5 Vaccinate 6.2.3.6 Wean piglets 6.2.3.7 Medical intervention 6.2.3.8 Selling pigs to slaughterhouse	Access policies to local server/repo/store.	2	Functional		Low	neutral

F11	Slaughterhouse Management System and Retail Management System should have access to both production (read) and slaughter (write) repositories.	Accounting though ebbits will be simplified thanks to the knowledge acquired by multiple systems in the food production chain.	6.2.4.1 Specification of payment to the farm 6.2.5.1 Order specification	Access policies to production and slaughter servers/repos/stores.	3	Functional		Low	Low
ENTERPRISE DOMAIN									
E1	Accounting Management System should store orders, and have access to Supplier Management System (to send/receive orders/acks/invoices).	The ebbits paradigm can be exploited also for improving the efficiency of accounting task.	Order goods Send invoice	Access policies to Supplier MS.	3	Functional		Neutral	Neutral
E2	Accounting Management System should have access to bank's management system for sending/receiving payment orders/confirms.	The ebbits paradigm can be exploited also for improving the efficiency of accounting task.	Send payment	Access policies to Banks' MS.	3	Functional		Neutral	Neutral
E3	Manager should have access to directory of stakeholders to interact with them (send/receive info).	The ebbits paradigm can be exploited also for improving the efficiency of accounting task.	Request for information	Access policies to Stakeholders directory.	3	Functional		Neutral	Neutral

3.4 Examples of annotation and access of semantic resources

In this section some examples of semantic annotation are presented in order to give a general idea of how the above requirements could be translated in semantic models of different resources.

3.4.1 Examples in the Automotive Manufacturing domain

Example RDF for a device/tool/machinery used in production

```
@prefix : <http://www.example.org/sample.rdfs#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

:Device rdf:type rdfs:Class;
 rdfs:subClassOf :Machine.

:hasId rdf:type rdf:Property;
rdfs:domain :Device;
rdfs:range xsd:integer.

:hasType rdf:type rdf:Property;
rdfs:domain :DeviceTypes;
rdfs:range :DeviceTypes.

:hasName rdf:type rdf:Property;
rdfs:domain :Device;
rdfs:range xsd:string.

:hasLocation rdf:type rdf:Property;
rdfs:domain :Device;
rdfs:range : GeoTag.

:hasLog rdf:type :DeviceStatus;
rdfs:domain :Device;
rdfs:range :DeviceStatus.


:log1 rdf:type : DeviceStatus ;
 :hasTimestamp "2011/06/01 13:24:56"^^xsd:datetime;
 :hasStatus "Off"^^xsd:string;
 :hasWorkInProgress :Piece;
 :hasConsumablesLevels "0.5"^^xsd:float;
 :hasQualityRecord "Good"^^xsd:string;
 :hasEnergyConsumption "Good"^^xsd:string;
 :hasEnergyProfile "SaveMode"^^xsd:string;
 :

:

:logN rdf:type :DeviceStatus;
 :
```

Example RDF for a piece in production

```
@prefix : <http://www.example.org/sample.rdfs#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

:Piece rdf:type rdfs:Class;
 rdfs:subClassOf :Product .

:hasId rdf:type rdf:Property;
rdfs:domain :Piece;
rdfs:range xsd:integer .

:hasType rdf:type rdf:Property;
rdfs:domain :PieceTypes;
rdfs:range :PieceTypes .

:hasStatus rdf:type rdf:Property;
rdfs:domain :Piece;
rdfs:range :GeoTag .

:hasLog rdf:type :ProductionStatus;
rdfs:domain :Piece;
rdfs:range : ProductionStatus .


:log1 rdf:type : ProductionStatus;
 :hasTimestamp "2011/06/01 13:24:56"^^xsd:datetime ;
 :hasStatus "in-progress"^^xsd:string ;
 :hasDevice :Device ;
:

:

:logN rdf:type : ProductionStatus;
 :
```

Example of SPARQL Query ^[Number of faulty pieces in the last month] (from use case 5.2.4.2 - Collect the production data)

```
PREFIX : <http://www.example.org/sample.rdfs#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT DISTINCT ?s WHERE {
        ?s :hasStatus ?a FILTER (?a = "faulty"^^xsd:string).
        ?m :Timestamp ?n FILTER ( ?n > "2011/05/01 00:00:00"^^xsd:datetime).
}
```

3.4.2 Examples in the Food Traceability domain

Example RDF for the pig farm

```
@prefix : <http://www.example.org/sample.rdfs#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

:Pig rdf:type rdfs:Class;
 rdfs:subClassOf :mammal .
```

```
:hasId rdf:type rdf:Property;
rdfs:domain :Pig;
rdfs:range xsd:integer .

:hasMother rdf:type rdf:Property;
rdfs:domain :Pig;
rdfs:range :Pig .

:hasWeight rdf:type rdf:Property;
rdfs:domain :Pig;
rdfs:range xsd:integer .

:motherSince rdf:type rdf:Property;
rdfs:domain :Pig;
rdfs:range xsd:integer .


:pig1 rdf:type :Pig ;
 :hasId "1"^^xsd:integer ;
 :hasMother :pig10;
 :hasWeight "5"^^xsd:integer .

:pig2 rdf:type :Pig ;
 :hasId "2"^^xsd:integer ;
 :hasMother :pig10 ;
 :hasWeight "8"^^xsd:integer .

:pig3 rdf:type :Pig ;
 :hasId "3"^^xsd:integer ;
 :hasMother :pig11 ;
 :hasWeight "8"^^xsd:integer .

:pig4 rdf:type :Pig ;
 :hasId "4"^^xsd:integer ;
 :hasMother :pig11 ;
 :hasWeight "10"^^xsd:integer .

:pig5 rdf:type :Pig ;
 :hasId "5"^^xsd:integer ;
 :hasMother :pig11 ;
 :hasWeight "5"^^xsd:integer .

:pig10 rdf:type :Pig ;
 :hasId "10"^^xsd:integer;
 :motherSince "35"^^xsd:integer .
 :

:pig11 rdf:type :Pig ;
 :hasId "11"^^xsd:integer ;
 :motherSince "25"^^xsd:integer .
```

Example of SPARQL Query ^[Wean Piglets] (from use case 6.2.3.6 - Wean piglets)

```
PREFIX : <http://www.example.org/sample.rdfs#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT DISTINCT ?s WHERE {
        ?s :hasId ?o. ?s :hasMother ?m.
        ?s :isAlive ?a FILTER (?a = "1"^^xsd:boolean).
```

```
        ?m :isAlive ?b FILTER(?b = "1"^^xsd:boolean).
        ?s :hasWeight ?w FILTER ( ?w > 7).
        ?m :farrowedOn ?n FILTER ( ?n < "2011-02-15"^^xsd:date).

}
```

Example of SPARQL Query ^[Sell N Pigs] (from Use case 6.2.3.8 - Selling pigs to slaughterhouse)

```
PREFIX : <http://www.example.org/sample.rdfs#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT DISTINCT ?s  WHERE {
        ?s :hasId ?o.
        {?s :hasWeight ?w FILTER ( ?w > 100)}
                UNION
        {?s :nonPregnant ?p FILTER (?p > 4)}
}
LIMIT 4
```

Example of SPARQL Query ^[Sell all Pigs]

```
PREFIX : <http://www.example.org/sample.rdfs#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT DISTINCT ?s ?w WHERE {
        ?s :hasId ?o.
        {?s :hasWeight ?w FILTER ( ?w > 100)}
                UNION
        {?s :nonPregnant ?np FILTER (?np > 4)}
}
```

Example of SPARQL Query ^[Living Pigs]

```
PREFIX : <http://www.example.org/sample.rdfs#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT DISTINCT ?s  WHERE {
        ?s :isAlive "1"^^xsd:boolean.
 }
```

4. Survey of semantic knowledge models usage within the ebbits project

The ebbits project is trying to provide semantic resolution to the Internet of Things (IoT) integration in enterprise business. It will try to focus on the selected domain and try to generalize results from these to prepare a reusable solution. From the semantic knowledge models point of view, this bottom up approach would require to prepare domain models for several levels of data and interactions from IoT up to inter-enterprise communication and planning. However, several existing formal descriptions of these are already available (ISA 88, ISA 95, FIPA ontologies, SWSDL, or several upper ontologies). Particular levels of enterprise structure are investigated in various tasks distributed among respective ebbits workpackages. Different data structures and relations are described for every level. These levels should better interact using the ebbits solution. Here we summarise, where we expect semantic knowledge models to become handy in particular workpackages.

WP2 Requirements Engineering *and Validation*

Generic and specific requirements elicitation is the main output of this workpackage. Performing evolutionary knowledge models related requirements refinement will be the starting point and continuous measure for both semantic infrastructure and knowledge models creation and population.

WP3 Enterprise Frameworks for Life-cycle Management

Here the Enterprise and Management processes are going to be described. Selected domains (Energy efficiency in manufacturing execution and Food traceability) will be used to better generalize the industrial processes. Semantic business decision models will be developed for executing orchestrated business services. Ownership of the data, has to be prevailed using regulatory frameworks.

WP4 Semantic modelling and enhancement of semantic stores.

The aim of this workpackage is to provide an optimised Semantic Store capable of storing all necessary knowledge models. The semantic store has to provide tools for semantic resolution over the knowledge. The semantic store has to be accessible from within the ebbits platform in a uniform way.

WP5 Centralised and Distributed Intelligence

Within this workpackage the ebbits project will provide a service-oriented architecture, a context management service framework, and define data structures, taxonomies, and ontologies for intelligent service components. Multi-sensory fusion rules, standardisation of sensor service interfaces will be provided also.

WP6 Mainstream Business Systems

This workpackage will provide information mapping for interoperability and integration of enterprise systems. Original information and metadata has to be available in an appropriate and controllable way in a distributed ebbits setup. Business rules need to be represented and executed. Query answering on business rules has to be available. The aim will be the identifying and describing generic standards for enterprise terminology in business processes and exchange between enterprise systems.

WP7 Event Management and Service Orchestration

Event management structure will be developed in this workpackage. Event and data structures, taxonomies, ontologies for the service components will be defined to support service orchestration. A service ontology with high-level concepts will be used in both development and run-time processes. Data management

structure should provide a model driven architecture for application development and deployment. A rule based service orchestration should allow the static or dynamic assembly of services and their execution.

WP8 Physical World Sensors and Networks

Network Management subset will be responsible for physical communication between devices, persons and external repositories. Syntactic interoperability between heterogeneous objects will be assured by mean of web services. Security management will be taken into an account. This level will be hidden from the semantic level by mapping all of very specific communication and security interaction behind well defined semantically enhanced web services.

WP9 Platform Integration and Deployment

Software Development Kit will be the main SW tool released within the project. Knowledge model infrastructure (the storage, the reasoning and querying capabilities) will be integrated, deployed and tested within this workpackage.

WP10 End-to-end Business Applications

Within this workpackage, ontologies will be populated for real-world examples. Domain specific business vocabularies and context rules will be defined. Testing of the platform will be performed.

WP11 Demonstration

Demonstration activities will be used for practical testing of particular models.

WP12 Dissemination and exploitation

Contribution of project results to relevant international standardisation bodies is crucial to keep results of the project reusable after the project. Knowledge models will be based on existing standards to ensure easier integration and interoperability with the rest of the world.

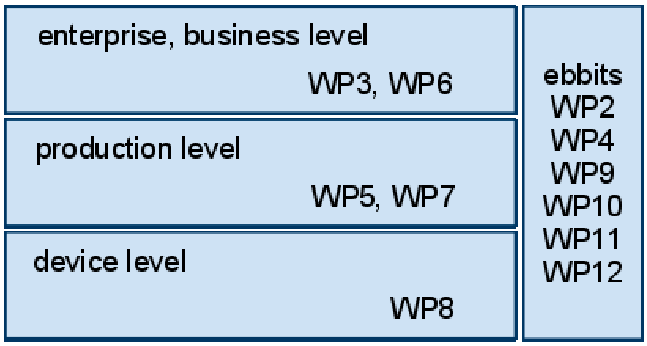

Figure 1: Simplified ebbits-enhanced enterprise

Overall position of knowledge models in different workpackages is depicted in a simplified enterprise schema on Figure 1. The aim is to use the ebbits middleware to enable better interaction of different levels in the scheme.

5. HYDRA semantic use cases and ontologies

One of the goals of the HYDRA project was to develop a middleware solution for networked embedded systems in ambient environments. The main output of the project is the middleware, which enables to connect various heterogeneous devices providing different services and with different capabilities. HYDRA is based on a Service-oriented Architecture (SOA) and provides an interoperable access to data, information and knowledge across heterogeneous platforms, including web services. It combines the use of ontologies with semantic web services, supporting thus true ambient intelligence for ubiquitous networked devices. Based on the combination of the ontologies with the semantic web services, HYDRA introduces the Semantic Model Driven Architecture (SeMDA) which aims to facilitate application development and to promote semantic interoperability for services and devices. The SeMDA of HYDRA includes a set of ontologies, and provides the set of tools, which can be used both in application design time and runtime. The ebbits project will take the HYDRA prototype solution further and will employ the HYDRA middleware in two application domains – manufacturing and agriculture. 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, actuators as services), people and workflows (operator and maintenance crews), optimisation using high level business rules (energy or cost performance criteria). The key requirement for the business rule execution is that the ebbits platform will be able to recognise and respond to the physical world events. Information acquired from events in the physical world, generated by various devices, create a basis for decision making at several levels of the ebbits architecture, including data fusion, situation pattern recognition, complex event processing, analysis of historical data, etc.

Another general characteristics of these applications is that they need to work with a large amount of information related to devices generating events or providing services, for further processing by event and service orchestration, decision or business rules. In some cases it must be possible to use this information to analyse also historical data generated by particular events. The decision making process will be supported by a rich semantic model enabling flexible knowledge representation of all the relevant events, roles, services and processes.

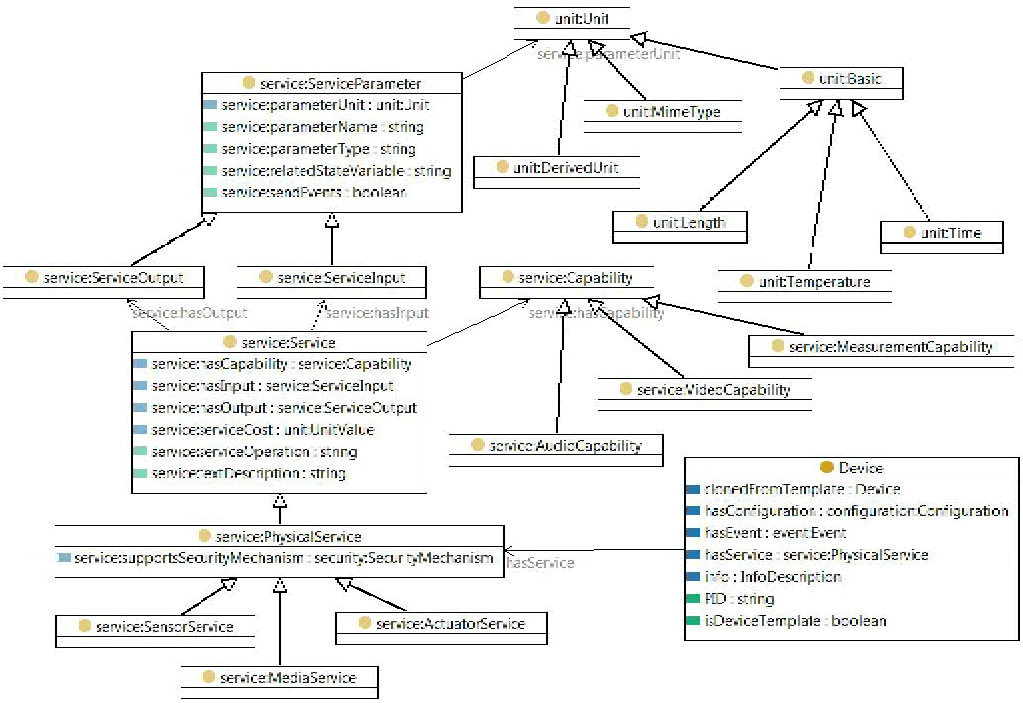

Figure 2: Illustration part of the HYDRA service ontology.

5.1 HYDRA ontologies

The most of models used in HYDRA are created as OWL-Lite [OWL, 2009] ontologies, in some cases the OWL-DL. With respect to characteristics of the domain, the careful modelling strategy was used. The

development of ontologies was strictly following the user and application requirements to keep them simple. The ontologies in HYDRA are used for both static information storage and also complex query answering purposes. This section will briefly introduce the core models used.

The HYDRA device ontology represents the concepts describing device related information, which can be used in both design and runtime. The basic ontology is composed of several partial models representing specific device information. The initial device ontology structure was extended from the FIPA device ontology specification (FIPA, 2002) and the initial device taxonomy was adopted and extended from AMIGO project vocabularies for device descriptions (AMIGO, 2006).

The core ontology contains the taxonomy of various device types and the basic device description that includes a model and manufacturer information.

Device services are modelled in the terms of operation names, inputs and outputs (see Figure 2 for illustration). The services are also organised into the taxonomy. The services are the basic executable and functionality units in HYDRA. To enrich the service description, to the model of service can be annotated several additional information, such as various capabilities, quality of service or security properties. The model of services used in HYDRA was inspired by OWL-S ontology (OWL-S, 2004). As the OWL-S was too exhaustive for the project purposes, the more suitable approach was to create simple and customised ontology for service description.

Device capabilities represent the hardware properties, software description and energy profiles. The mentioned information profiles are modelled as the static structures, where only one profile of each type can be attached to the device.

Discovery models contain the models of all discovery information provided by low-level communication protocols. Discovery model is mandatory and is attached to each device. The purpose of device discovery information ability to resolve the suitable device semantic model when new device enters the HYDRA network and is initially described only by low-level discovery information depending on communication protocol used.

Semantic device model represent the logical aggregates of composed devices to provide more advanced application related functionality. Semantic devices are modelled as the set of semantic services specified by preconditions, which have to be satisfied for semantic device to be executable. The preconditions specify the static or dynamic requirements for devices embedded in semantic device.

Application models contain the set of ontologies dedicated to various application domains. Each application model specifies the domain entities and relations in order to achieve the context aware application behaviour.

Quality of Service (QoS) model contains descriptions of various aspects of service quality. The high level properties such as taxonomy of service functional capabilities (e.g. plays video or measures temperature) are modelled. The QoS ontology contains also the specification of the lower level service properties, such as response time, availability or reliability. QoS ontology also contains the taxonomy of various units (such as temperature, time, pressure, currency, etc.).

Device malfunctions represent various types of errors and failures, which may occur when using the device at runtime. For each malfunction there is a set of possible remedies in the form of text description.

Security properties specify the various security properties, such as protocols, algorithms or objectives, which may be attached to devices or services. To describe the security properties, the third party NRL ontology (NRL, 2007) was integrated and annotated to device ontology.

Configuration model supports the device creation using DDK tools. For each created device, the information about the configuration and implementation files used by particular IDE are stored. These files serve as the templates of code or IDE project files and can be reused when new similar devices are created. Another purpose of configuration models is the support for automatic device code generation (e.g. selecting the suitable device implementations) for the device development.

5.2 Using the semantics in HYDRA

SeMDA of HYDRA provides the set of tools helping the application developer to use any wireless or wired device easy. All devices in HYDRA application are accessible in uniform way – as the semantic web services.

In order to achieve this, developer has to prepare all devices, which have to be used in the application, with help of the SeMDA tools. For each device there is created the semantic description, which can be used for device discovery, calling device services satisfying various requirements (such as suitable quality of service) or context-aware application behaviour. This functionality is ensured by SeMDA, thus the development process is simplified and the underlying implementation is transparent to the developer. This section will briefly introduce the basic scenarios of using Semantic Web technologies in application design and the runtime.

5.2.1 HYDRA enabling device

To achieve the semantic interoperability, each device in HYDRA application is exposed as the semantic web service. This approach enables to discover and call the device services in a common way, taking advantage of the semantic description. In application design time, each device has to be prepared for usage in HYDRA. This process is called the HYDRA enabling of the device. Developer can HYDRA-enable a new device using so-called device development kit (DDK). The new device is annotated to the suitable class in the device taxonomy (e.g. mobile device) and the basic description, such as device model name and number, manufacturer information, energy consumption profile or device discovery information is added. As the particular devices have the different connection and communication capabilities, the service calls have to be transformed into web service calls. For each service, the developer has to add the custom implementation. Each service is also annotated to the suitable service taxonomy class.

Devices use various low-level communication protocols, such as ZigBee or BlueTooth. Each device, depending on the supported communication protocol, has to be physically discovered by one of the discovery managers. When device is physically discovered in the network, each communication protocol provides the set of low-level device information. In order to be able to automatically discover the specific device in the future, the low-level discovery information is stored.

The role of semantics

The whole process of device HYDRA enabling is guided by the ontology, which contains the basic device, service taxonomies, basic device information and the models of energy consumption and low-level discovery information. Developer browses the taxonomies provided by the ontology when selecting the suitable device or service class. Basic information and energy consumption are entered into forms automatically generated from the ontology. Once the device is prepared, the new ontolology instances are automatically generated and the ontology is extended by the new device basic model. Ontology contains one instance for each specific device model. The device ontology representation is used in runtime for device discovery, searching for specific services or devices and for retrieving all information required for the service calls. Creating new device introduces only basic device semantic representation, which can be later further extended. Each device ontology instance represents the specific device model and serves as the static information template. In runtime, when new device enters in the HYDRA network, the best matching template is identified by the semantic discovery process, cloned and tied to the physical device. The values of the runtime instance can change as the device changes its state variables (e.g. measured values of sensor). When physical device leaves the network, assigned device runtime instance is removed from the ontology.

5.2.2 Semantic device discovery

When a new device enters the HYDRA network, it is discovered using one of the low-level discovery managers dedicated to various low-level communication protocols such as Bluetooth or ZigBee. In the most cases, the low-level discovery retrieves only weak information dependent on the particular protocol capabilities. In runtime, this information is used to identify the corresponding semantic device model in the ontology containing full description of device, its services and other relevant information assigned to device model in the design time. The newly discovered device has to be semantically resolved against all templates in the ontology. The semantic resolution is performed by comparing the actual low-level discovery information to discovery information assigned to device ontology templates in HYDRA enabling process.

Semantic device resolution can finish with the success, when there is just one best match. In this case, the run-rime instance of the identified ontology device template is cloned and tied with the physical device using the HYDRA specific persistent identifier. The full device information is retrieved from the ontology and passed to the main HYDRA application component called device application catalogue (DAC). DAC is responsible for mediating actual information of all devices presented in the HYDRA network and for

execution of the services. Using the device ontology representation, DAC creates the proxy transforming the low-level protocol calls to the web service calls using the device implementation created in HYDRA enabling process. When semantic resolution fails, that means, that there is no suitable semantic model for a device. In this case, DAC is populated with the generic device information and the physical device has no semantic support.

The role of semantics

Each low-level communication protocol represents the device discovery information in very different way. Sometimes, all available device information include only device model name and number, sometimes the various manufacturer information. In case of more sophisticated protocols, such as BlueTooth or UPnP, there can be also available the list of services or other extending information. For each low-level discovery information there exists the model in the ontology. The low-level discovery information is translated into the SPARQL (SPARQL, 2007) query and the solution of semantic device resolution is transformed into the graph matching problem. In many cases, the execution of query retrieves the more matching candidates, which have to be further investigated by heuristically comparing possible additional information. The possible additional information for each communication protocol are modelled in the ontology, so in the implementation of comparison procedure there is no need to hard-code the particular comparison cases. As the discovery information provided by communication protocols may be very different, it would be quite difficult to design the suitable fixed data structures supporting such a information variability. The flexibility of semantic representation and uniform information access through the graph matching is here the big advantage.

5.2.3 Extending the device semantic description

Most of sophisticated applications working with various devices would require to search for the devices satisfying the several requirements. For example, the application working with many sensor devices would require to compute average energy consumption of the applicances attached to the specific type of robots in the particular hall. Another application would need to track the water consumption of concrete type of animals during the focused time of the day. In another case it would be required to execute the service with the specific functionality while saving the energy consumption.

Semantic description of device models created in the HYDRA enabling process represent only the basic information necessary for device functionality. These information can be further extended using the Eclipse based IDE, which serves as the ontology and annotation editor. Based on most frequent application functionality and internal HYDRA requirements, the HYDRA ontology was extended by the models of hardware, software and energy profiles, quality of service properties and security properties. Device ontology was also extended by properties used to annotate the extended information to device models. As in the most cases the requirements were to search for services having several properties, the domain of annotation properties are mostly the classes from the service taxonomy. The hardware, software and energy consumption information are modelled as the static structures, there can be one hardware, software or energy profile per device. There can be multiple annotations of quality of service and security properties. Using extended semantic descriptions, the devices and services have the full semantic support and are findable in various ways.

The role of semantics

One of the biggest advantages of semantic modelling was the absolute simplification of extension process in ontology editor. The whole process of knowledge extension is guided by the ontology. Developer may browse related extension information. In case of static information, such as hardware or energy profiles, the input forms are generated automatically. In case of dynamic properties, which could be added using annotations, the developer can browse the structure of device or service and ontology editor automatically offers the relevant information, which can be added to the particular ontology entity. For example, for service inputs/outputs there can be offered relevant units depending on the device type (for example, for output parameter of temperature sensors, the editor would most likely offer the units of Celsius of Fahrenheit). All depends on the model in the ontology. If ontologies are extended with the new domain annotated to the device or service models, the tool automatically offers the new possibilities without any further implementation efforts. The dynamic extension of semantic device properties creates the basis for using the semantic web services in HYDRA. The fact, that the several application components, or in some cases the different HYDRA applications itself, use the same domain vocabulary, leads to the semantic interoperability at the semantic level.

5.2.4 Application context-awareness

When designing the application, in many cases it is required to create the application domain model. For example, in the case of the home automation application (home automation was one of the HYDRA usage scenarios), it is helpful to specify, which locations (e.g. rooms) will application have, which persons use the application, which devices belong to specific locations or are owned by concrete persons. This way it is possible to describe the application domain, but also to create the background knowledge for the reasoning in the context. When working with the application domain model, sometimes it is required to specify, which devices have to be used for certain operations (e.g. switch on the lamp besides Peter's bed in bedroom), so it is required to address the device directly. In other cases, it is required to infer, which devices belong to the specific context (e.g. get average temperature of thermometers in the living room).

The role of semantics

In HYDRA, the application domain models are integrated into ontologies including the properties for annotating devices to the context entities (e.g. rooms or persons). When developer creates the application, he or she can select, which devices will be used for context computations. These devices can be annotated using ontology editor IDE to the relevant context entities. Then, in application logic implementation, it is possible to call pre-implemented and parameterised ontology search services. The parameters of the query are formulated in specific notation developed for the purposes of simplification of the query mechanism. The query is formulated using the IDE, where developer can simply select, which parameters are searched and which parameters should be retrieved for further processing. The IDE translates the required parameters into the SPARQL query, which is executed against the ontology. The search methods then retrieve all devices matching parameters to be satisfied. For each device, the result contains a set of properties to be retrieved. In more complicated cases, developer can directly formulate the SPARQL query. Using this approach, developer has to be aware of the ontology structure and vocabulary. Anyway, with the support of inference mechanism, it is possible to achieve reasonably smart application behaviour already when using really simple context models.

5.2.5 Semantic devices

Each physical HYDRA device provides a set of specific services, which can be directly used by the application developer. For example, the thermometer may provide the device specific services, such as *getTemperature* or *setTemperature*. The idea behind the semantic devices is to enhance the application development by providing the application specific services. For example if there are more thermometers in the room, an application may provide e.g. *getAverageRoomTemperature* or *holdTemperatureAtSspecifieLevel* services. The concept of semantic devices brings the idea of specifying the application specific behaviour achieved as the composition of several HYDRA devices organised into the complex units (Kostelnik et.al., 2008). Simply said, semantic devices are logical aggregates of devices. Semantic devices can include both basic, but also other semantic devices. For example, the heating system semantic device may include embedded thermometer devices and provide the semantic services – the behaviour composed of all embedded services. Each semantic device is defined by the set of semantic services. Each semantic service is composed by the set of requirements in terms of preconditions. The preconditions are used in the runtime to generate the candidates matching the specified requirements (e.g. all thermometers measuring in degrees of Celsius located in the hall).

In design time, developer has to define and implement the semantic device services using DDK tool. In runtime, each time the new device enters the application, the semantic devices are rediscovered and the required devices satisfying defined preconditions are automatically tied with the semantic devices.

The implementation of semantic device is realised as the combination of statically defined devices and the orchestration behaviour. The static definition is used only in the case, when semantic service has to work exactly with some specific devices. But this specification does not entail any limitation for using also orchestrated devices. For example, the developer may decide to create a specific temperature alert device using just some selected thermometers in the room, which have to be specified (thermometers are specified as the concrete devices – static mapping).

The more complex semantic devices may be also used as the decision units providing a specific functionality in terms of effectiveness by some specified criteria. For example, application may use two semantic devices capable of controlling the light in the room. One semantic device controls the lamps, another controls the blinds. These two devices may be composed into more complex semantic device, which would be capable

for example to save the energy. Using the specific information, the device will be able to decide, how to perform the light control. Using more information about devices, e.g. various kinds of energy profiles, semantic devices can be used as standalone units implemented to perform the operations while satisfying the specified goals (e.g. energy saving). The application development can be radically simplified by reusing the existing semantic devices adjusted for the specific environment.

The role of semantics

When developer creates the new semantic device in the DDK tool, he or she has the picture of the target device functionality. The ontology support is useful when selecting the proper devices or services, which should be embedded in the semantic services. Ontology driven design simplifies the whole development process, as the ontology editor IDE provides the wide set of specific listings and views filtering the devices by the specified requirements (e.g. quality of service). For all devices having the suitable functionality, there is immediately accessible list of full signatures of available services. The role of developer is reduced to the selection of proper devices and to implementation of pure logic of semantic service.

In the runtime, the presence of devices in the HYDRA network may change. When devices enter or leave the HYDRA network, the ontology is continually queried and all affected semantic devices are rediscovered. Each change may cause, that some of available semantic devices are disabled, some may be enabled for usage. For more, semantic devices have to ensure the real-time orchestration of embedded devices. Each time when semantic services are executed, the ontology has to infer the actually presented devices matching the specified preconditions.

6. Proposed HYDRA ontology extensions for ebbits use cases

As ebbits is based on the HYDRA middleware, all the semantic use cases provided by HYDRA will be used, with respect to the extension of the ontologies for the ebbits purposes. This section aims to outline the use cases and the overview of expected extensions of the HYDRA ontologies in the context of ebbits. The proposed extensions will refer to the particular ebbits workpackages and the related parts of the HYDRA ontologies.

6.1 Ebbits specific semantic use cases

6.1.1 Business decision models for enterprises

In ebbits there are several level of generality for the service orchestration task. At the lower level, it will be needed to use the composition and orchestration of low level physical services and events. This problem was partly solved also in the HYDRA middleware, but will have to be extended and implemented in ebbits. This level of abstraction will be focused in subsection 3.1.3.

The specific case of semantic usage in ebbits is the higher level abstraction of the processes, which have to be specified for the co-operation of enterprises at the business level. This case is solved by workpackages 3 and 6.

WP3 will deliver the models describing the management, business and industrial processes and their parts in a general way. The processes have to be modelled in the way appropriate to be used in the decision models and the orchestration of higher level processes.

WP6 builds on the models delivered within the WP3 and focuses on the higher level processes specific for the enterprises. The main focus here is to identify and describe standards for enterprise terminology in business processes and exchange between enterprise systems. Generic models provided by WP3 have to be concretely defined with respect to the specific domain of enterprises.

The higher level of processes was not part of the HYDRA ontologies and will have to be created.

6.1.2 Multi-sensory data fusion and context-awareness

Important part of the semantic usage is the context modelling, generalisation of services & events to be used in multi-sensory fusion and context-aware decisions. This problem was focused within the HYDRA middleware only partly. HYDRA used the rule systems to handle the context-aware behaviour of the applications. The context rules took the advantages of the semantic model in the terms of using the semantic information related to the context entities of the interest and extending the knowledge about the context entities. The problem of multi-sensory data fusion was not solved within the HYDRA.

The context-awareness and multi-sensory data fusion is the responsibility of WP5. Depending on the decisions of technologies employed to handle these issues, the corresponding semantic models will be created to support the infrastructure with the semantic inference and the extension of information to improve the decision-making processes.

In this case, it is assumed that the semantic extension of context entities, such as devices, services, events, or persons, will be provided. This leads to the creation of context-awareness supporting properties of the relevant context entities. Another issue is also to create the domain models supporting the context. In ebbits, comparing to HYDRA, it will be needed to create the domain models of higher level and domain specific models of context entities. HYDRA was focused only to device and service models. Ebbits will have to focus also the application domain elements, such as events, persons or system states. This models will have to be prepared as generic, to be ready for extensions and specialisation for concrete application domains.

The similar situation is expected for the problem of multi-sensory data fusion. In this case, it is expected that the support of semantics will be realised as an extension of service and event models to provide the inference and extending information useful for the data-fusion algorithms.

6.1.3 Event management and service orchestration

The workpackage WP7 focuses on the event management and the rule based service orchestration allowing the static or dynamic assembly of services and their execution.

The event management in HYDRA did not rely on the semantic support of events, even though the HYDRA ontologies contained the models of events. In the HYDRA ontologies, the event models had only the information value for the developer, but were not actively used in the software event management. In ebbits, the event management task is crucial and will need the full semantic support in the automatic processing of events. The event models from HYDRA ontologies had to be abandoned and the new model of events was created to satisfy the basic assumptions for semantic support of event management. The prototype of semantic model of events was delivered in D7.2 Event and data structures taxonomies and ontologies.

The problem of service orchestration was focused in HYDRA, but was solved mostly on the level of application logic. Ebbits aims to deliver a more generic and automatic handling of composed/orchestrated services. A solution of this problem will have to be supported also semantically, with an extension of the models of services by the properties needed to be able to ensure the service discovery and investigation of service inputs, outputs and other functional properties. This extension should lead to the possibility of composing/orchestrating the services in a semi-automatic or fully automatic way, depending on the implementation of orchestrating engine in WP7.

6.1.4 Resource annotation and knowledge retrieval

In HYDRA, the only resources used in the semantic support were the devices and services. For more, the HYDRA application usually contained only few devices comparing to the situation, which is expected in ebbits. In HYDRA, it was enough to hold the all models in the ontology, because thanks to the number of used resource entities, the knowledge retrieval realised as the querying of the semantic model was still fast enough to be suitable.

In the ebbits, it is expected to handle the hundreds of resources including devices, but also the application specific entities, such as persons, parts of an industrial or business processes or animals. Taking into account this situation, the approach to be selected must be different to the HYDRA. To ensure the suitable response times of the knowledge information retrieval, it would be mostly suitable to use the semantic repositories for holding only the semantic information, not the physical data. This leads to the approach, where the application entities will have to be semantically annotated, but physically stored in the more efficient way. In this case, the retrieval of the knowledge will have to be realised using the semantic annotations between the semantic information and the related application entity.

This will most probably lead to the hybrid knowledge retrieval of information joining the semantic knowledge and the other type of data depending on the character of the storage, where the data about the particular entity is contained. Of course, in many cases it would be needed only to retrieve the semantic knowledge about the particular entity using the annotation information without need of merging the information.

The solution to this problem is responsibility of WP4. The responsibility of WP4 should also be the semantic modelling, where other workpackages deliver the models needed for semantic support specific for the particular WP, but this models are formally created as ontologies within the WP4, to hold the process of ontology creation on the one place.

6.2 Extension of HYDRA ontologies for ebbits

6.2.1 Reusing the HYDRA ontologies

The HYDRA device ontology represents the core device taxonomy including the sub-classed models of devices of certain type. The core can be fully reused and continually extended and improved for the device types to be used in ebbits. The ontology properties of devices in the taxonomy should be improved to be specific for concrete device types making the reasoning process more efficient.

The same situation holds for the models of services. The service taxonomy can be further improved for the ebbits use cases. For more, for the purposes of the service composition and orchestration, the HYDRA service ontology has to be extended with the properties efficiently describing the service functional properties and the models of inputs and outputs with association to the particular domain models. This extension should lead to the possibility of semi-automatic or automatic service composed processes. This extensions would also enable the more straightforward device or service discovery following the specific application or composition requirements, mainly for the purpose of automatic service orchestration.

Device capabilities, such as software, hardware or energy (or other resources) consumption information modelled in HYDRA ontologies can be reused as the basis of the ontology model, but have to be further extended for the specific ebbits requirements. As ebbits aims also to address the energy (or generally resource) consumption handling scenarios, the resource consumption models have to be further extended or redesigned, depending on the implementation of this scenarios.

Device discovery ontologies used in HYDRA were used for identification of the proper semantic model for the devices entering the system. When discovery was successful, the physical device was tied to the semantic model and the device had the full semantic support providing the additional information used in several scenarios, such as context-awareness resolution. The HYDRA semantic discovery process often used only pure low-level information describing the particular device models and the success of semantic discovery was often quite poor. In ebbits, the discovery models and discovery process must be further improved to increase the success of the semantic device discovery.

Quality of service and security ontologies used in HYDRA can be reused as they are, with the several extensions for the specific ebbits purposes.

One of the crucial part of ebbits is the event management. In HYDRA, the ontology for events was used only for information purposes. In ebbits, the event models have to be redesigned and aligned to the requirements of ebbits event management. The prototype of event ontology created following the basic assumptions of event management needs was designed in the deliverable D7.2 Event and data structures taxonomies and ontologies.

6.2.2 Ebbits specific ontologies

In HYDRA, there was not addressed the problem of managing the higher-level processes and events, such as business rules, business/enterprise/industrial processes and their compositions. Ebbits must deliver the semantic support for higher-level processes and events. The extension would include the general ontologies describing the business rules, decision models and their specification to the enterprise terminology. The models of processes must also take into account the information enabling the semantic support for business rule execution and inter-enterprise information exchange. This models have to be further extended and specified for the several addressed domains and applications.

Domain modelling in HYDRA was limited to the description of devices and services. Application domain models were created just experimentally for the proof of concept implementations. In ebbits, it will be needed to provide semantic domain modelling support to enable the specific views of devices, services and events for various applications. For more, the ebbits introduces the models of higher-level business rules and processes, which has to be extended and specified for particular concrete problem and application domains. The higher-level processes include also various context entities different from devices and services, such as persons, system states or animals. The domain models of context entities must be created in the general way and further specified for the various context entities contained in the higher-level processes. The particular domain models have to be further specified for the purposes of concrete applications.

7. Conclusion

TO DO: IS+TUK
Summary (e.g. a list/table) of updates on the LinkSmart/Hydra ontology.

8. References

(AMIGO, 2006) IST Amigo Project (2006). Amigo middleware core: Prototype implementation and documentation, deliverable 3.2. Technical report, IST-2004-004182.

(FIPA 2002) FIPA Device Ontology Specification, Foundation for intelligent physical agents, 2002.

(Kostelnik et al. 2008) Kostelnik, P., Sarnovsky, M., Hreno, J., Ahlsen, M., Rosengren, P., Kool, P., Axling, M.: Semantic Devices for Ambient Environment Middleware. In: EURO TrustAmi 2008, Internet of Things and Services, Sophia-Antipolis, France 18-19 September 2008.

(NRL, 2007) Naval Research Lab. Nrl security ontology http://chacs.nrl.navy.mil/projects/4SEA/ontology.html, 2007.

(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/

(SPARQL, 2007) E. Prud'hommeaux, A. Seaborne, SPARQL Query Language for RDF, W3C Proposed Recommendation, 2007.