Contract No. IST 2005-034891

Hydra

Networked Embedded System middleware for
Heterogeneous physical devices in a distributed architecture

D6.3 Semantic Web Services Design Document

Integrated Project
SO 2.5.3 Embedded systems

Project start date: 1st July 2006 **Duration: 48 months**

Published by the Hydra Consortium **2008-08-25 - version 1.1**
Coordinating Partner: C International Ltd.

Project co-funded by the European Commission
within the Sixth Framework Programme (2002 -2006)

Dissemination Level: Public

Document file: D6.3 Semantic Web services Design Document v1 01.doc

Work package: WP6

Tasks: T – 6.5

Document owner: CNet

Document history:

Version	Author(s)	Date	Changes made
0.1	Matts Ahlsén, Peter Rosengren, Peeter Kool	2007-08-31	ToC and initial structure
0.13	Matts Ahlsén	2007-10-19	Structure revised
0.15	CNet/ FIT /SIT	2007-11-09	Ch 4, 5
0.16	Mathias Axling	2007-11-12	Ch 4 update, service annotation and grounding
0.17	Julian Schütte, Matts Ahlsen	2007-11-25	Ch 4.4 Security issues
0.19	Peter Kostelnik, Peter Butka	2007-12-07	Update of ch 4, SWS basics for Hydra
0.20	Alexander Schneider	2007-12-12	Design of SWS
0.25	Matts Ahlsén, Peter Rosengren, Peeter Kool, Mathias Axling	2007-12-15	
0.30	Matts Ahlsén, Peter Rosengren	2008-01-15	Re-structuring of document
0.35	Matts Ahlsén, Peter Rosengren	2008-01-16	Formatting and numbering of document. Adding appendix.
0.40	Matts Ahlsén, Peter Rosengren, Peeter Kool	2008-01-18	Revised chapter 4
0.50	Matts Ahlsén, Peter Rosengren, Peeter Kool	2008-01-21	Revised service grounding aspects in chapter 1 and 4.
0.60	Matts Ahlsén, Peter Rosengren, Peeter Kool	2008-01-22	Revised scenario in chapter 4
0.70	Matts Ahlsén, Mathias Axling, Peter Rosengren, Peeter Kool	2008-01-23	Revised chapter 4, added UPnP discovery example.
0.80	Matts Ahlsén, Peter Rosengren, Peeter Kool	2008-01-24	Added service description examples.
0.90	Matts Ahlsén, Mathias Axling, Peter Rosengren, Peeter Kool	2008-01-25	Added service grounding examples.
0.95	Matts Ahlsén, Mathias Axling, Peter Rosengren, Peeter Kool	2008-01-25	Caching principles discussed
0.96	Matts Ahlsén, Peter Rosengren	2008-01-28	Revision after internal WP6 review. Version for Hydra Internal Review
0.98	Matts Ahlsén, Peter Rosengren	2008-02-04	Revision after internal project review
1.0	Matts Ahlsén, Peter	2008-02-05	Final version submitted to the

	Rosengren		European Commission
1.1	Matts Ahlsén	2008-08-25	Dissemination level set to Public

Internal review history:

Reviewed by	Date	Comments
Pablo Antolin	2008-01-31	Approved with comments
Daniel Thiemert	2008-02-01	Approved with comments

Index:

Figures

Tables

1. Introduction

1.1 Background

The Hydra project aims to research, develop, and validate middleware for networked embedded systems that allows developers to develop cost-effective, high-performance ambient intelligence applications for heterogeneous physical devices.

The first objective is to develop middleware based on a Service-oriented Architecture, to which the underlying communication layer is transparent. The middleware will include support for distributed as well as centralised architectures, security and trust, reflective properties and model-driven development of applications.

The Hydra middleware will be deployable on both new and existing networks of distributed wireless and wired devices, which operate with limited resources in terms of computing power, energy and memory usage. It will allow for secure, trustworthy, and fault tolerant applications through the use of novel distributed security and social trust components.

The embedded and mobile Service-oriented Architecture will provide interoperable access to data, information and knowledge across heterogeneous platforms, including web services, and support true ambient intelligence for ubiquitous networked devices.

The second objective of the Hydra project is to develop an Integrated Development Environment (IDE). The IDE will be used by developers to develop innovative semantic model driven applications with embedded ambient intelligence using the Hydra middleware.

1.2 Purpose, context and scope of this deliverable

Hydra aims to interconnect devices, people, terminals, buildings, etc. The Service-Oriented Architecture and its related standards provide interoperability at a syntactic level. However, in Hydra we also aim at providing interoperability at a *semantic level*. The objective of WP6 is to extend this syntactic interoperability to the application level, i.e., in terms of semantic interoperability. This is done by combining the use of ontologies with semantic web services.

In order to cope with the huge variety of capabilities of the devices to be integrated in Hydra, the middleware layer should provide adaptations to whatever interface the devices offer. To achieve this, Hydra aims to be able to describe the capabilities of the devices (ontologies) in such way that an automatic agent can understand these capabilities and use them. Once the semantics describing the model of the other device has been found, then the device capabilities could be accessed. This is done using semantic web service technologies.

This document (D6.3) describes Semantic Web Services Design principles of Hydra. It complements deliverable D6.2 *MDA Design Document* which describes the semantic model driven architecture of Hydra.

Chapter 1 gives an introduction to semantic web services and the contributions Hydra makes. Chapter 2 provides and executive summary. Chapter 3 gives and overview and analyses different existing standards in this area, while chapter 4 focuses on describing the approach and design decisions Hydra makes with regards to semantic web services for devices in light of our current understanding of the requirements. Finally in chapter 5 we discuss our intended future work. There are also two appendices attached to the document – A complete summary for the requirements with regards to semantic web services for devices, and the specification of relevant software managers.

1.3 Hydra Innovations and Contributions

Hydra's technological innovations in Semantic Web Service Design will be achieved in the following areas:

- Ontology-based Device and Service Descriptions

- Semantic Discovery and Advertising of Networked Devices and their Services
- Lightweight orchestration and composition of Device Services
- Ontology-driven Invocation and Execution of Device Services
- Secure Semantic Web Services for Devices
- Automatic generation of SWS Device Proxies
- Caching principles

The following highlighted extract from table 5 in the DOW section 4.5 "Technologies to be used, researched and developed" summaries the intended contributions from WP6 with respect to SoA and semantic web services for devices.

WP 6 SoA and MDA middleware

Technology area	Use of existing technologies	New technologies to be developed	New technologies to be researched
Embedded and mobile service-oriented architectures for AmI	The Hydra middleware will be based on mature web service technologies such as SOA, SOAP, WSDL, BPEL etc. to the furthest extend possible Embedded web services will be built using standard WS technologies including: • Web services stack • Fast evaluation of WS • Semantic stack	Technologies for bringing semantic web service technology down to device level to provide semantic interoperability between devices. •	New technologies for integration of WS with the device level will be researched. This will include: • Automatic generation of web services device proxies. • Caching principles
Semantic Model-Driven Architecture for AmI	The model driven architecture will be build with standard web service technologies including domain model meta descriptors such as IFC and HL7 classes Ontology frameworks will be based on standards such as OWL Horizontal standards such as WS-Coordination and WS-Transaction will be considered	New technologies for maintaining and accessing distributed domain meta models will be developed Semantic cooperative instantiation of devices, personas and services will be developed	Technologies for Automatic Device classification Technologies for Semantic-cooperative reasoning. New techniques based on combination UML and OWL for automatic construction and maintenance of ontologies will be researched. Research of principles and technologies for Intelligent Rules Processing to allow for configuration of device behaviour.
Semantics and knowledge management	Prototype semantic approaches will be used, e.g., inspired by OWL-S or SWS based on the	New technologies to provide interoperability at the semantic level will be developed	

	Semantic Web, to support properties such as discovery, context awareness, self-* properties Standard Knowledge Management (KM) techniques for knowledge capture, indexing and re-use will be deployed where needed and applicable	including profiling knowledge repository technologies for preference engineering	

Table 1: WP6 SWS contribution objectives

1.4 Service-Oriented Architecture

The Service Oriented Architecture (SoA) represents an architectural style where the primary concept is the use of loosely coupled, implementation-neutral services supporting a business process as building blocks. Service consumers use the service by means of its published interface-based service description without dependence on implementation, location or technology. The process building of combining and sequencing services to provide more complex services is known as orchestration.

A SoA solution is built of a set of services orchestrated by clients or middleware to realize an end-to-end (business) process. The openness of the architectural style also allows for ad-hoc service consumers and flexible and dynamically re-configurable processes. The World Wide Web Consortium (W3C) defines SoA as "A set of components which can be invoked, and whose interface descriptions can be published and discovered". No universally agreed definition is available, but the term is generally considered to imply that application functionality is provided and consumed as sets of services which can be published, discovered and accessed and are loosely coupled as well as implementation and technology neutral.

SoA encourages loose coupling among the interacting software systems. A service is used only via the published service description and the service consumer does not address a specific implementation or deployed instance of the service. Changes to the implementation do not affect the service consumer and the service consumer can change the instance of the service that is used (changing location or implementation of the service, e.g. when two service providers offer the same service) without modifying the client application.

By abstracting the service from the implementation, the developer will not need to consider which technique was used to implement the service. Parallel implementations of the service may be available, and the actual version used is transparent to the consumer.

The use of standardized protocols for publishing, discovering and accessing services allows the service to be provided on any platform that can implement these protocols. In orchestrating a SoA solution, services that are (internally) implemented with different languages, architectural styles and on platforms from different vendors, can be used together transparently.

Any technology that can be used to implement loosely coupled, implementation independent services could be used to realize SOA. However, most discussions and actual implementations of SoA use Web Service technologies as the way of publishing, discovering and accessing a service. Web Service technologies include SOAP and XML for exchanging messages containing structured and typed information to access services, to publish and describe a service and UDDI for dynamically finding and invoking web services. On top of these now well-established protocols, a host of new protocols have been developed to support orchestration of services and describe the semantics of services e.g. OWL-S builds on OWL to define a core set of mark-up language constructs for describing the properties and capabilities of Web services, WS-Coordination provides a method of defining and supporting workflows and business processes. WS-Coordination is an extensible framework for providing protocols that coordinate the actions of individual web services in distributed applications to provide a business process defined in BPEL. WSRF (Web Services Resource Framework) defines an open framework for modelling and accessing stateful resources using Web services.

The principles of SoA and loosely coupled, flexible and dynamically configured systems harmonizes well with Hydra objectives (e.g. with the use of an organisational memory) and the abundance of proposed standards and technologies will be evaluated and tested for applications in Hydra.

All of the software components comprising Hydra will be integrated in a Service Oriented Architecture (SoA), which will provide, among other things, interoperability. The Hydra middleware thus also becomes the link between web services and devices. Interoperability, which here is taken as the capability of components of Hydra to talk to each other no matter which is the technology used to implement them or their physical location, is achieved by means of the usage of many specifications around the web services world, including XML, SOAP, WSDL, XML Schema, WS-Security, WS-Addressing and several others. The key point that makes these standards interoperable is that they are platform-agnostic, but more important, that most industry player support most of them.

To summarise, the main purpose of the Service-oriented Architecture in Hydra is to provide interoperability between devices at a *syntactic level*.

1.5 Semantic Web Overview

Tim Berners-Lee defined the scope as 'Expressing Meaning' and 'Knowledge Representation' and the basic building blocks of the Semantic Web with 'Ontologies' and 'Agents' which would bring an 'Evolution of Knowledge' [3]. Though the vision of Berners-Lee has not been fulfilled a lot of research went into the two building blocks he mentioned namely Ontologies and Agents.

Web services have also seen a tremendous amount of research in various areas like service description, service provision and service discovery. A lot of standardization efforts are going on at the moment trying to harmonize the access to and usage of web services like the WS-* standards from W3C. At the beginning web services were replacing proprietary middleware to invoke methods across networked machines e.g. CORBA or RMI and do so with a standardize protocol like SOAP (Simple Object Access Protocol) [1].

It was used in EAI (Enterprise Application Integration) and besides the SOAP protocol a language for service description named WSDL (Web Service Description Language) and a registry for looking up available web services called UDDI (Universal Description, Discovery and Integration) were developed. The basic architecture of the components is shown in Figure 1 [op. cit.]:

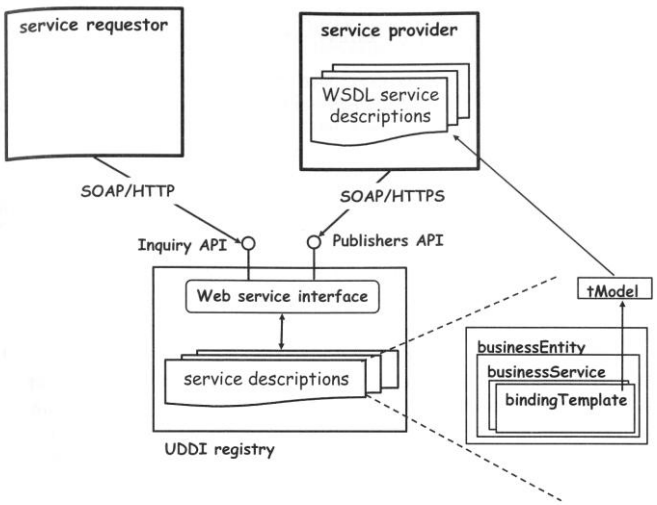

Figure 1: Basic web service architecture

The basic web service approach has one notable limitation in that the service descriptions are not machine-understandable. The format is standardized (WSDL) but it lacks a well-defined semantics.

In the area of the Semantic Web a lot of research work was done on ontologies and their usage as content markup languages. In the beginning there was OIL [6] and DAML [7] which eventually evolved into OWL (Web Ontology Language) [20] and OWL-S [18]. McIllraith et al. first described the usage of ontologies together with web services to create semantic web services [21]. Semantic web services are web services which are semantically described so that the semantic markup of service descriptions becomes machine understandable. This helps in the automatic service discovery to find a web service that offers a specific functionality and also matches a certain set of properties, in automatic service execution and automatic service composition in that one can specify a set of properties and an agent could consume one particular semantic web service or automatically combine several available web services to execute the needed functionality.

While a lot of work has been done in all single areas it is still problematic to combine those new technologies to form a working system though. This has a lot to do with standards that are created from different working groups but that are somehow addressing the same area but are incompatible to each other like the WSMO [16] vs. the OWL-S approach. Therefore it is impossible to identify one standard that one could follow and be sure that this is the "right" approach. Moreover projects like WSMO are still being actively developed and lack therefore support for the complete process. WSMO for example has specified a so called web service execution environment (WSEX) but this is still far from being ready for production systems.

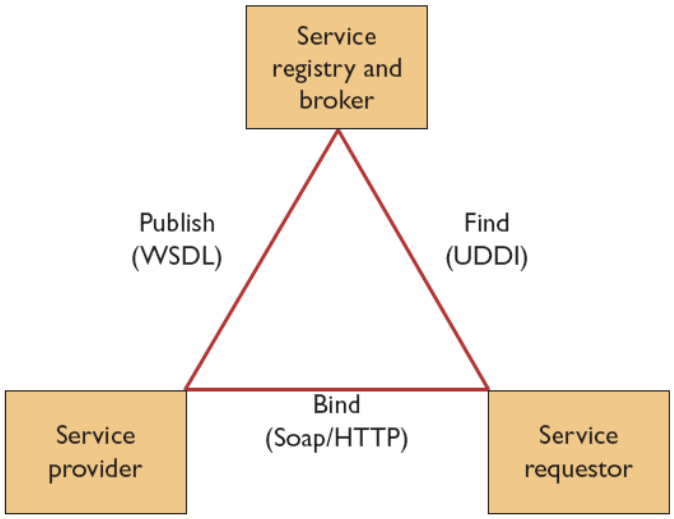

Figure 2: Basic service oriented architecture [8]

The migration path from the standard web service infrastructure to semantically enhanced web services is also still in flux. Figure 2 shows a basic web service architecture in which the steps Publish and Find needs to be semantically enriched.

1.6 Semantic Web Services for Devices

The main contribution of Hydra to the Semantic Web is to bring semantic web technologies down to the device level, i.e. each device can act as a semantic web service accessible by other devices, users and software applications. This work is carried out as part of Task 6.5 "SoA and Semantic Web Services for Devices", and this deliverable complements the deliverable D6.2 "MDA Design Document" [9].

1.6.1 Ontology-based Device and Service Descriptions

Putting semantic web services on devices is a question of merging two different perspectives – a device-oriented perspective which arises from technologies like UPnP (Universal Plug and Play) with a service-oriented perspective that stems from semantic web service technology. Hydra models

services separate from devices, by representing them in two related models represented by the Device Ontology (DO) and the Service Ontology (SO).

As a tool for modelling Hydra services, it is possible to use both OWL-S and WSMO technology, which enable to solve all of the tasks that have to be solved by the second iteration of Hydra services. Both approaches provide an acceptable solution. OWL-S as the well known standard seems to be more mature in various aspects, whereas WSMO provides more complete conceptual model, but its specification and implementation is still incomplete and in development. In addition, since the first Hydra iteration is characterised by the tasks of service discovery, explicit composition and invocation, both standards seem to be overly complex for the needs of Hydra.

When searching for simple and practical solution, OWL-S or WSMO approach should be used mainly in cases, when there is the need for modelling of such a complex issues as:

- reasoning with the service preconditions and effects or
- service orchestration with ability of searching the services in the work-flow on the fly

For the first prototype of Hydra services we will use the SAWSDL standard [30] annotated to the custom service model. The development of service ontology must take into account the future extension of Hydra requirements on the services. It should also be possible to completely substitute the custom Hydra service ontology with selected SWS standard, such as OWL-S or WSMO.

1.6.2 Semantic Discovery of Networked Devices and Services

An important aspect of all ambient intelligence applications is for users, applications and devices to quickly and easily discover devices that are available in there vicinity. The first issue is to discover the existence of a device that one can communicate with, the second issue is to discover what type of services the device offers and thirdly to discover how to access and execute these services.

One of the contributions from Hydra is to merge UPnP (Universal Plug and Play) discovery of networked devices with semantic services, allowing UPnP-enabled devices to act as semantic web services towards the network.

The UPnP architecture offers pervasive peer-to-peer network connectivity of PCs, intelligent appliances and wireless devices. The UPnP architecture is a distributed, open networking architecture that uses TCP/IP and HTTP. It enables seamless proximity networking in addition to data transfer between networked devices at home, in the office and everywhere in between.

It enables data communication between any two devices under the command of any control device in the network. The UPnP architecture supports zero-configuration, invisible networking and automatic discovery for a breadth of device categories from a wide range of vendors. Devices can dynamically join a network, obtain IP addresses, announce their names, convey their capabilities upon request, and learn about the presence and capabilities of other devices. DHCP and DNS servers are optional. A device can leave a network smoothly and automatically without leaving any unwanted state information behind.

The goal of service discovery task is to find a suitable service provided by specific device (or device type) in accordance to defined requirements. UDDI is an industry initiative that is becoming the de facto standard repository for web services. But it provides a weak discovery mechanism which does not allow the discovery of services based on the functionality the service provides. Therefore an OWL-S/UDDI matchmaker is needed that provides the capability to search for web services based on their functionality [24]. Such a matchmaker needs a mapping between concepts from the ontology (in this case OWL-S) and UDDI which is shown in Figure 3[23] [18].

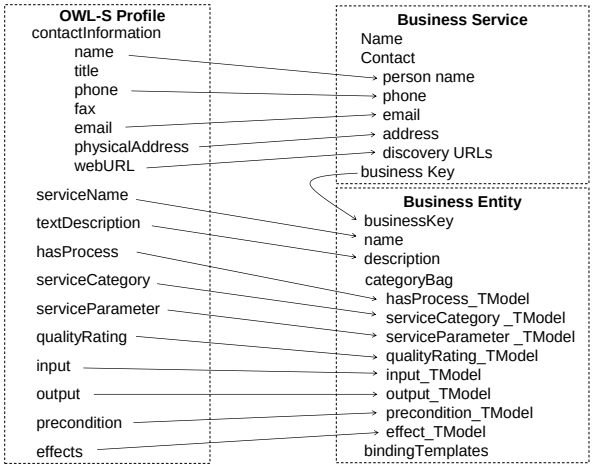

Figure 3: UDDI matching to OWL-S

There are existing tools and matchmakers supporting the service discovery for both OWL-S and WSMO standards (description of this tools is out of scope of this deliverable), which may be used for particular approach. The issue, which should be especially addressed, is the support of using the IOPEs (Input, Output, Preconditions and Effects) for service discovery. In the real applications, IOPEs are not used properly, because the reasoning with preconditions and effects in real-time discovery process is very time expensive. Usually, the potential preconditions and effects are skipped or pre-computed.

1.6.3 Lightweight Orchestration of Device Services

In a service oriented architecture (SOA) some of the key aspects are loose coupling of services, implementation neutrality, flexible configurability and coarse granularity. If a service designer has those goals in mind while defining the scope of one service it will usually be a rather low-level service from a functionality point of view. In order to create higher-level services one has to define in which order other services will be consumed and then find and execute them.

So within a SOA service discovery is only the starting point. To create useful applications on the SOA architecture style one has to orchestrate services to support workflows that were previously defined as well as creating composite services out of existing lower level.

In the first prototype of Hydra complex services were created by static service composition. The *orchestration* of a sequence of services can be done by several technologies. WS-BPEL is the extension of BPEL (Business Process Execution Language) to web services but is recognized to be rather complex and most probably not suited to the requirements of Hydra

Existing orchestration approaches appear to be overly complex and resource intensive for use in Hydra. Therefore we will research a more lightweight approach to be used. During this iteration we will make various experiments with different approaches and then in later iterations design such a custom orchestration language ("Device Orchestration Language Light", DOLL).

1.6.4 Ontology-driven Invocation of Services

Semantic Web Services frameworks like OWL-S and WSMO combine semantic descriptions of Web service capabilities, inputs, outputs and behavior with the syntactic interface descriptions in WSDL and XML Schema. The glue between the semantic and syntactic description layers is called *grounding [17]*.

Device Services are invoked using a *grounding model*, which specifies how to communicate with a particular service. The Hydra approach to service grounding and invocation is based on the combination of WSDL semantic annotation with grounding references in the ontology. The device ontology holds descriptions of devices and services. The discovery, mapping and reasoning done to find suitable services to accomplish a certain task is performed using the ontology. The WSDL

grounding for the semantic service is referenced in the ontology, so that the web service on any device that can be found in the ontology also can be called without using information from the device (although the device could provide its own WSDL).

Hydra is also allowing service grounding directly to UPnP[1].

1.6.5 Secure Semantic Web Services for Devices

Security is an important issue when it comes to web services - whether semantic or not. Although web services are re-usable and accessible components by design, not every service is thought to be used by everybody. Access to a service may rather depend on the identity of the requester, of certain attributes or even of the current context. As web services themselves do not support any kind of access control, such mechanisms have to be provided in addition to Hydra's services. Even other security requirements like confidentiality, non-repudiation, integrity or authenticity are not an integral part of web services and have to be specified by additional mechanisms.

1.6.6 Automatic Generation of SWS proxies for Devices

To enable the Hydra middleware and consequently the Hydra developer to view and use all devices and services in a heterogeneous fashion, automatic generation of semantic web service proxies for devices will be necessary. Some devices will provide their own semantic information with hooks into the service and device ontologies while the simpler, non-Hydra enabled devices will have to be identified in the device ontology and their semantic information added to the description in a semantic web service proxy. For developers, some additional support for domain concepts will also be generated by the SDK.

1.6.7 Caching Principles

While the semantic technologies in combination with the service oriented architecture of Hydra, enhance the functionality and usability of the Hydra middleware, the quality of service must also be attained to sufficient levels. This may pertain to the accessibility of both devices and device services. As many Hydra applications will be designed for networked and distributed environments, it is foreseen that caching techniques could be exploited on several levels in the Hydra architecture to improve accessibility and performance of device and service use.

Caching issues are not considered in this iteration of the Hydra development cycle and will therefore not be further discussed in this document.

[1] UPnP – Universal Plug and Play: http://www.upnp.org/

2. Executive Summary

This workpackage applies Service Oriented and Model Driven Architecture techniques to AmI systems. All of the devices and services comprising a Hydra network will be integrated in a *Service Oriented Architecture* (SoA), which will provide, among other things, interoperability. The Hydra middleware thus also becomes the link between web services and devices. Interoperability, which here is taken as the capability of components of Hydra to talk to each other no matter which is the technology used to implement them or their physical location, is achieved by means of the usage of many specifications in the context of the web services world, including XML, SOAP, WSDL, XML Schema, WS-Security, WS-Addressing and several others. To summarise, the main purpose of the Service-Oriented Architecture in Hydra is to provide interoperability between devices at a *syntactic level*.

Hydra aims to interconnect devices, people, terminals, buildings, etc. As mentioned above, the Service-Oriented Architecture and its related standards provide interoperability at a syntactic level. However, in Hydra we also aim at providing interoperability at a *semantic level*. Thus, the Hydra middleware must also model services offered by different devices from an applications point of view.

A main contribution of this workpackage to is to bring semantic web technologies down to the device level, i.e., each device can act as a semantic web service accessible by other devices, users and software application. This will be done in close cooperation with WP4 which are investigating techniques for embedding web services into devices. In this WP we are concerned with the design of semantic web services and automating the generation of web services code for devices based on meta data and ontology descriptions.

In order to cope with the huge variety of capabilities of the devices to be integrated in Hydra, two broad options can be considered: a) to force every device to be compliant to some set of more or less flexible interfaces, or b) to have Hydra middle layer provide adaptation to whatever interface the devices offer.

Since choice a) will probably not be applicable neither to the present nor to the future world, Hydra will opt for choice b), so it will try to be able to adapt to the variety of interfaces, information and operations that devices offer. And given the vast amount of devices, the only viable option to address this issue is to try to do it in some automatic way.

In order to achieve this, Hydra aims to be able to describe the capabilities of the devices (ontologies) in such way that an automatic agent can understand these capabilities and use them. Once the semantics describing the model of a device has been found, then its device capabilities could be accessed.

Hydra's technological innovations in semantic web services design will be in the following areas:

- Ontology-based Device and Service Descriptions
- Semantic Discovery and Advertising of Networked Devices and their Services
- Lightweight orchestration and composition of Device Services
- Ontology-driven Invocation and Execution of Device Services
- Secure Semantic Web Services for Devices
- Automatic generation of SWS device proxies
- Caching principles

3. Standards for Semantic Web Services

In this chapter we describe, analyze and compare the most important standards that exist for Semantic Web Services.

3.1 OWL Web Ontology Language for Services (OWL-S)

OWL-S is the OWL ontology for semantic description of the Web Services [29]. The structure of the OWL-S consists of a service profile for service discovering, a process model which supports composition of services, and a service grounding, which associate profile and process concepts with the underlying service interfaces.

Service profile has functional and nonfunctional properties. Functional properties describe the inputs, outputs, preconditions and effects of the service (IOPEs). The nonfunctional properties describe the semi-structured information intended for human users for service discovery, e.g. service name, description and parameters which incorporates further requirements on the service capabilities (e.g. security, quality-of-service and geographical scope).

Service model specifies how to interoperate with the service. The service is viewed as a process which defines the functional properties of the service (IOPEs), together with details of its constituent processes (if the service is a composite service). The service model functional properties can be shared with the service profile.

OWL-S distinguishes between atomic, simple, and composite processes. OWL-S atomic processes can be invoked, have no sub-processes, and are executed in a single step from the requester's point of view. The simple processes are used as elements of abstraction, they are viewed as executed in a single step, but they are not invocable. Composite processes consist of the simple processes and define their work-flows using control constructs, such a sequence, split, if-then-else or iterate.

Service grounding enables the execution of the Web Service by binding the abstract concepts of the OWL-S profile and process model to concrete messages and protocols. Although different message specifications are supported by OWL-S, the widely accepted WSDL is preferred.

3.2 Web Service Modelling Ontology (WSMO)

The Web Service Modeling Ontology (WSMO) [33] is a conceptual model for describing semantic Web Services. WSMO consists of four major components: ontologies, goals, Web Services and mediators.

Ontologies provide the formal semantics to the information used by all other components. WSMO specifies the following constituents as part of the description of ontology: concepts, relations, functions, axioms, and instances of concepts and relations, as well as non-functional properties, imported ontologies, and used mediators. The latter allows the interconnection of different ontologies by using mediators that solve terminology mismatches.

Goals specify objectives that a client might have when consulting a Web Service, i.e. functionalities that a Web Service should provide from the user perspective. In WSMO a goal is characterized by a set of *non-functional properties*, *imported ontologies*, *used mediators*, the *requested capability* and the *requested interface (see the Web Services description)*.

A *Web Service* description in WSMO consists of five sub-components: non-functional properties, imported ontologies, used mediators, a capability and interfaces.

The *capability* of a Web Service defines its functionality in terms of preconditions, postconditions, assumptions and effects. A capability (therefore a Web Service) may be linked to certain goals that are solved by the Web Service via *me*diators. Preconditions, assumptions, postconditions and effects are expressed through a set of axioms and a set of shared all-quantified variables.

The *interface* of a Web Service provides further information on how the functionality of the Web Service is achieved. It describes the behavior of the service from the client's point of view (service

choreography) and how the overall functionality of the service is achieved in terms of cooperation with the other services (service orchestration).

A choreography description consists of the states represented by the ontology, and the if-then rules that specify (guarded) transitions between states. The ontology that represents the states provides the vocabulary of the transition rules and contains the set of instances that change their values from one state to the other. The concepts of an ontology used for representing a state may have specified the grounding mechanism which binds service description to the concrete message specification (e.g. WSDL).

Like for the choreography, an orchestration description consists of the states and guarded transitions. In extension to the choreography, in an orchestration can also appear transition rules that have as postcondition the invocation of a mediator that links the orchestration with the choreography of a required Web Service.

Mediators describe elements that aim to overcome structural, semantic or conceptual mismatches that appear between the different components that build up a WSMO description. Currently the specification covers four different types of mediators:

- *OOMediators* - import the target ontology into the source ontology by resolving all the representation mismatches between the source and the target;
- *GGMediators* - connect goals that are in a relation of refinement allowing the definition of sub-goal hierarchies and resolve mismatches between those;
- *WGMediators* - links a goal to a Web Service via its choreography interface meaning that the Web Service fulfills the goal; or links a Web Service to a goal via its orchestration interface meaning that the Web Service needs this goal to be resolved in order to fulfill the functionality;
- *WWMediators* - connect several Web Services for collaboration.

WSMO is formalized using the Web Service Modeling Language (WSML, [32]) which is based on description logic, first-order logic and logic programming formalisms. WSML consists of a number of variants based on these different logical formalisms, namely:

- *WSML-Core* is based on the intersection of Description logic and Horn logic;
- *WSML-DL* extends WSML-Core to an expressive Description logic and offers similar expressiveness to OWL-DL;
- *WSML-Flight* is an extension in the direction of Logic programming and incorporates a rule language, while still allowing efficient reasoning;
- *WSML-Rule* extends WSML-Flight to Logic programming language, which does not require rule safety and allows to use function symbols; and
- *WSML-Full* unifies all WSML variants under a common first-order umbrella with non-monotonic extensions.

3.3 Web Service Semantics (WSDL-S)

WSDL-S is a small set of proposed extensions to Web Service Description Language (WSDL) by which semantic annotations may be associated with WSDL elements.

WSDL-S defines URI reference mechanisms to the interface, operation and message WSDL constructs to point to the semantic annotations defined in the externalized domain models. WSDL-S defines following extensibility elements and attributes:

- *modelReference* element - allows for one-to-one associations of WSDL input and output type schema elements to the concepts in a semantic model;
- *schemaMapping* attribute - allows for many-to-many associations of WSDL input and output complex type schema elements to the concepts in a semantic model. It can point to a transformation (for example XSLT), from XML data to the external ontological data in RDF/OWL or in WSML;

- *precondition* and *effect* elements - are used on WSDL interface operations to specify conditions that must hold before and after the operation is invoked. The conditions can be specified directly as a expression with format defined by the semantic language or by reference to the semantic model;
- *category* element - provides a pointer to some taxonomy category. It can be used on a WSDL interface and is intended to be used for taxonomy-based discovery.

3.4 Semantic Annotations for WSDL and XML Schema (SAWSDL)

Semantic Annotations for WSDL and XML Schema is a W3C recommendation [30] that defines how to add semantic annotations to Web Service Description Language (WSDL) and to XML Schema. It defines the extension attributes that can be applied to elements in both WSDL and XML Schema in order to annotate WSDL interfaces, operations and their input and output messages. SAWSDL is the successor of WSDL-S often considered to be the first step towards standardization in the area of Semantic Web Services.

Semantic annotations in WSDL and XML Schema are used for these purposes:

- associating WSDL interfaces with some taxonomical categories to help semantic Web service discovery,
- describing the purpose or applicability of WSDL operations to help discovery or composition,
- linking and mapping inputs, outputs and faults of WSDL operations to semantic concepts to help facilitate mediation and service discovery and composition.

According to the Semantic Annotations for WSDL Working Group (http://www.w3.org/2002/ws/sawsdl/), the key design principles for SAWSDL are:

- The specification enables semantic annotations for Web services using and building on the existing extensibility framework of WSDL.
- It is agnostic to semantic representation languages.
- It enables semantic annotations for Web services not only for discovering Web services but also for invoking them.

SAWSDL can be split in two extension components: (1) An extension attribute, named *modelReference*, to specify the association between a WSDL component and a concept in some semantic model and (2) two extension attributes, named *liftingSchemaMapping* and *loweringSchemaMapping*, that are added to XML Schema element declarations and type definitions for specifying mappings between semantic data and XML. These two components are described in the following text.

Model reference is the first major part of SAWSDL represented by the attribute called *modelReference*. The value of the attribute is a list of URIs that reference concepts in an external semantic model. SAWSDL defines how model references can be used on WSDL interfaces, operations, faults, and on XML Schema element declarations or type definitions.

On a WSDL interface, a model reference can provide a classification of the interface, for example by pointing into a products and services taxonomy. Model references on a WSDL operation define what the operation does. This can be done with a direct reference to a verb concept or to a logical axiom or by specifying the operation's preconditions and effects. On a WSDL fault, model references define what kind of failure the fault means, so that the fault can be handled more appropriately by the client. Model references on XML Schema element declarations and type definitions define the semantics of the inputs or outputs of WSDL operations.

In general, model references can have many uses, and indeed, SAWSDL does not limit the applicability of the attribute.

Schema mappings transform between XML data described with XML Schema and semantic data described by a semantic model. Mappings can be used for example to support invocation of a Web service from a client that works natively with semantic data. SAWSDL defines two extension

attributes: *liftingSchemaMapping* and *lowerSchemaMapping*. These attributes are used to point from a schema element declaration or type definition to a mapping that specifies (in any suitable mapping language, e.g. XSLT) how data is transformed from XML to the semantic level (*lifting*) or back (*lowering*).

3.5 Comparison and relation of SWS standards

SWS standards can, according to the modelling approach, generally be divided into two categories: (1) standards which cover the semantic description of web services in their own specification (OWL-S, WSMO) and the standards which add the semantic information using the annotations to external knowledge sources (WSDL-S/SAWSDL approaches). This chapter describes the basic differences between OWL-S and WSMO. As the both approaches primarily use the WSDL as the base for service grounding, the relationship of these standards to WSDL-S/SAWSDL will be described (as the SAWSDL is the WSLD-S successor, description will be focused mainly on SAWSDL).

3.5.1 Comparison of OWL-S and WSMO

WSMO and OWL-S are the two major efforts whose purpose is to specify semantic information for Web services in order to enable automatic service discovery, composition and execution. However, there are substantial differences between these approaches [26].

OWL-S uses four different ontologies to describe the service. The upper Service ontology refers to: Profile, Service Model and Grounding ontologies. WSMO is based on the Web Service Modelling Framework (WSMF), which divides the service description into four components: Ontologies, Web Services, Goals and Mediators. The first significant difference between these approaches is that OWL-S does not exactly separate what a user requires from what the service provides. The OWL-S Profile is used both for service advertising and for discovery. In WSMO, a Goal defines, what the user needs and the Web Service specifies what the service provides using its capabilities.

The non-functional properties in an OWL-S Profile (such as human-readable service name, description, etc.) are not explicitly based on standard meta-data specifications. WSMO tends to use the common vocabularies such as the Dublin Core element set. Moreover, the WSMO enables to use the non-functional properties in any element, in OWL-S is this restricted only to the service Profile.

In the OWL-S Service Model, there is no explicit difference between the orchestration and the choreography and there is only one Service Model for each service, thus there is only one way to interact with the service. In WSMO, the choreography and orchestration are defined in the interface of the Web Service. WSMO also enables the use of multiple interfaces for each service.

OWL-S supports the logical expressions defined in the SWRL[2] and KIF[3] languages, but the interaction between the inputs and outputs specified as OWL classes, and the logical expressions, are not clear enough. Various kinds of mediation is required to create the relation between various heterogeneous resources. OWL-S does not explicitly handle the issue of mediation, it is considered to be handled by the underlying Web Service infrastructure. WSMO explicitly defines the mediation in the conceptual model.

To summarize, OWL-S and WSMO are very similar, although with some differences in the approach they take to achieve their goals. OWL-S seems to be more mature in certain aspects, including the definition of process model and grounding specifications. However, WSMO provides a more complete conceptual model as it addresses aspects such as goals and mediators, but it has to further define some open aspects to be completely usable in real applications.

3.5.2 Relation of OWL-S and WSMO to SAWSDL

SAWSDL provides a standard means by which WSDL documents can be related to semantic descriptions, such as those provided by OWL-S and WSMO. As a standard, SAWSDL provides a

[2] Semantic Web Rule Language: http://www.w3.org/Submission/SWRL/
[3] Knowledge Interchange Format: http://www-ksl.stanford.edu/knowledge-sharing/kif/

common ground for the various ongoing efforts toward SWS frameworks. In both cases, the utilization of SAWSDL description into WSMO and OWL-S is still more on the level of suggestions than definitions, which require the further investigation.

The guidelines and suggestions regarding the use of OWL-S in conjunction with SAWSDL from the SAWSDL perspective is provided by [19]. It is explained what OWL-S constructs are appropriate for use with the various SAWSDL annotations. These explanations are provided with a view to supporting WSDL users and WSDL tool vendors in achieving the objectives that are associated with SAWSDL. The recommendations can be briefly summarized as follows:

- The WSDL operation can refer to an OWL-S atomic or composite process.
- The WSDL interface should refer to an instance of an OWL-S profile class (i.e., Profile or a subclass of Profile). If a particular instance is not available, a profile class can serve as the referent.
- The WSDL fault should refer to a conditional effect of an OWL-S process – the process that corresponds to the operation for which the fault is declared.
- Model references in XML Schema should refer to OWL constructs, and can do so independently of OWL-S.
- WSDL input and output elements should be used to relate those elements to inputs and outputs of an OWL-S process – the process that corresponds to the operation for which the input or output element is declared.
- Schema mapping (lifting and lowering) annotations can refer to XSLT scripts.

The approach showing WSMO grounding using SAWSDL, linking from WSDL components to WSMO is described in [16]. Grounding using SAWSDL is described in the terms of WSMO choreography, which specifies when certain data can be sent or received and the grounding specifies how exactly it can be sent or received. To ground a WSMO choreography, it is needed to put model references on the element declarations that are inputs or outputs of WSDL operations in very similar way as in the case OWL-S:

- An element that is an input message to a WSDL operation should contain a model reference to an "in" or "shared" concept in a WSMO choreography, and an output message element should have a model reference to an "out" or "shared" concept.
- Additionally, the liftingSchemaMapping and loweringSchemaMapping attributes should refer to the data grounding transformations (e.g. XSLT scripts). Any SAWSDL schema mapping annotations can be used for pointing to data grounding.
- The WSDL service element should refer to a WSMO web service

Using SAWSDL for WSMO grounding brings both benefits and drawbacks over the WSMO-based grounding, therefore both approaches can be seen as the alternatives. The main possible benefits and drawbacks can be briefly summarized as follows:

- The SAWSDL-based grounding improves the relation of WSMO to Web services standards, making it easier for Web service users to take advantage of semantic descriptions.
- SAWSDL provides schema mapping annotations to attach lifting and lowering transformations which are not yet specified in the WSMO-based grounding.

SAWSDL-based grounding links are in a WSDL description; however a WSMO semantic execution environment is primarily based on WSMO, where the grounding links are readily available whenever the execution environment needs them. With SAWSDL-based grounding, the grounding information needs to be looked up by looking through all the known WSDL descriptions.

For both, OWL-S and WSMO standards, the SAWSDL-based grounding improves the relation of OWL-S and WSMO to Web services standards, making it easier for Web service users to take the advantage of semantic descriptions. The SAWSDL-based grounding also allows the partial understanding of the semantic description. For instance, the links from XML Schema element

declarations to ontology concepts may be not only useful for discovery or execution of services, but can also be used by human-oriented tools to enhance the manipulation of the schema with semantic information available from the ontology.

4. Hydra approach to Semantic Web Services for Devices

In this chapter we will explain our approach to achieve semantic interoperability between devices and their services. We will start by giving a short overview of the ideas behind the semantic MDA of Hydra, and then briefly describe an example scenario. After that we will then discuss the approach we are taking with regards to:

- Device and Service Descriptions
- Discovery and Advertising of Networked Devices and their Services
- Orchestration, composition and choreography of Device Services
- Invocation and Execution of Device Services
- Security
- Automatic generation of SWS device proxies

For each of these areas we will discuss different alternatives and explain the choices and decisions we have made.

4.1 Overview of the Semantic MDA of Hydra

The semantic model-driven architecture of Hydra (SeMDA) is based on the application of ontologies and semantic web technologies to support the design of device-oriented networked applications and is also intended as a run-time resource in the execution of device services. The SeMDA is explained in detail in deliverable D6.2 "MDA Design Document", here we will only discuss the aspects relevant for Semantic Web Services.

The basic idea behind the Hydra Semantic MDA is to differentiate between the physical devices and the application´s view of the device. We introduce the concept of *Semantic Devices*. The physical devices offers a set of services, a lamp might offer "on/off" and "dimming" as two services while a pump might offer "increase flow" and "get water temperature" as two services.

The services offered by the physical devices have been designed independently of particular applications in which the device might be used. A semantic device on the other hand represents what the particular application would like to have. For instance, when we are designing the lighting system for a building it would be more appropriate to model the application as working with a logical lighting system that provides services like "working light", "presentation light", and "comfort light" rather than working with a set of independent lamps that can be turned on/off. These logical devices might in fact consist of aggregates of physical devices, and use different devices to deliver the service depending on the situation. The service "Working light" might be achieved during daytime by pulling up the blind (if it is down) and during evening by turning of a lamp (blind and lamp being Hydra devices). We call these logical aggregates of devices and their services for *Semantic Devices*.

Semantic Devices should be seen as a programming concept. The application programmer designs and programs his application using semantic devices. Figure 4 below illustrates the concept. The semantic device "Heating System" consist of three physical devices: a pump that circulates the water, a thermometer that delivers the temperature and a light that flashes when something is wrong.

The developer will only have to use the services offered by the semantic device "Heating System", for instances "Keep temperature:20 degrees Celsius" and "Set warning level:17 degrees Celsius", and does not need to know the underlying implementation of this particular heating system.

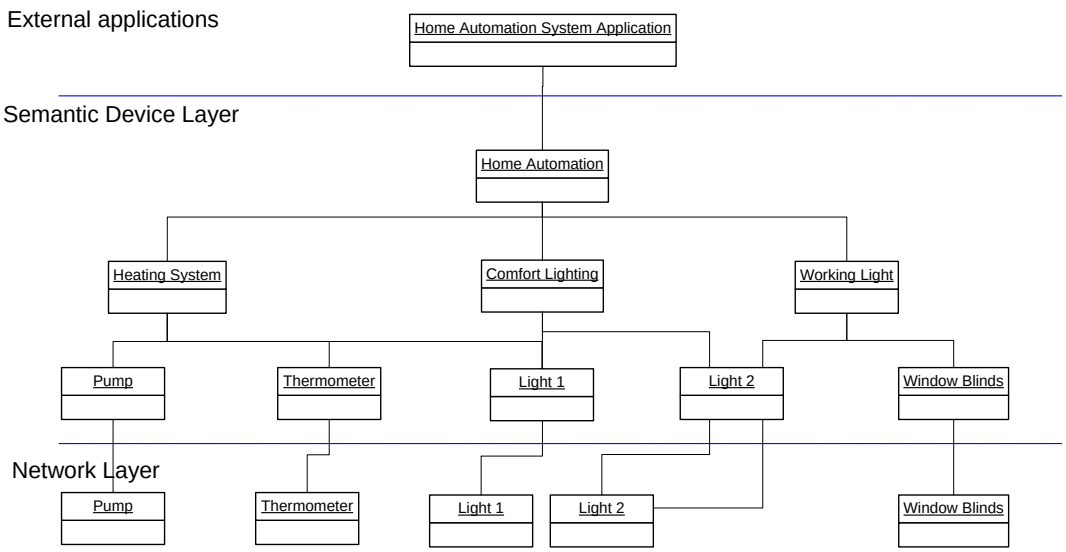

Figure 4: Semantic Devices provide a high-level programming interface.

The Semantic Device concept is flexible and will support both static mappings as well as dynamic mappings to physical devices.

Static mappings can be both 1-to-1 from a semantic device to a physical device or mappings that allow composition.

- An example of a 1-to-1 mapping would be a "semantic pump" that is exposed with all its services to the programmer.
- An example of a composed mapping is a semantic heating system that is mapped to three different underlying devices – a pump, a thermometer and a digital lamp.

Static mappings will require knowledge about which devices exists in the runtime environment, for instance the heating system mentioned above will require the existence of the three underlying devices – pump, thermometer and lamp – in a building.

Dynamic mappings will allow semantic devices to be instantiated at runtime. Consider the heating system above. We might define it as consisting of the following devices/services:

- A device that can circulate the water and increase its temperature
- A device that can measure and deliver temperature
- A device that can create an alarm/alert signal if temperature is out of range.

When such a device is entered into the runtime environment it will use service discovery to instantiate itself and it will query the physical devices it discovers as to which can provide the services/functions the semantic device requires. In this example the semantic device most probably starts by finding a circulation pump.

But then it might find two different thermometers which both claims they can measure temperature. The semantic device could then query about which of the thermometers can deliver the temperature in Celsius, with what resolution and how often. In this case it might only be one of the thermometers that meet the requirements. Finally the semantic device could search the network if there is a physical device that can be used to generate an alarm if the temperature drops below a threshold or increases to much. By some reasoning the semantic device can deduct that by flashing the lamp repeatedly it can generate an alarm signal, so the lamp is included as part of the semantic heating system.

The basic idea behind semantic devices is to hide all the underlying complexity of the mapping to, discovery of and access to physical devices. The programmer just uses it as a normal object in his

application focusing on solving the application's problems rather then the intrinsic of the physical devices.

To achieve our vision of a Semantic Model Driven Architecture we have chosen to base our approach on ontologies and related semantic technologies. In Hydra there are three major ontologies used - Device Ontology, Security Ontology and Software Components Ontology.

The Hydra Device Ontology presents the basic high level concepts describing basic device related information, which will be used in both development and run-time process. The device ontology is divided into four interconnected modules:

- Basic device information and taxonomy
- Device malfunctions
- Device capabilities and state machine
- Device services

The content and structure of the Device Ontology as well as the others ontologies are described in more detail in deliverable D6.2 "MDA Design Document".

To summarise, there are two uses of the semantic MDA in Hydra. Firstly, it is relevant at design-time, and it will support both device developers as well as application developers. Secondly, at run-time any Hydra application is driven from the semantic MDA.

To realize our vision of the semantic MDA we not only need ontologies to represent information regarding devices and services, but we also need solutions for exchanging and using metadata regarding devices and services in runtime, i.e. we need to achieve *semantic interoperability* between devices and applications. The purpose of this chapter is to describe how we intend to solve these issues.

4.2 Example Scenario (Sending SMS)

Putting semantic web services on devices is a question of merging two different perspectives – a device-oriented perspective which arises from technologies like UPnP (Universal Plug and Play) with a service-oriented perspective that stems from semantic web service technology.

An example environment involving two Smartphones and a service for sending SMS messages will be used to illustrate the use of device and service ontologies, grounding and service descriptions. The devices both have a service for sending SMS messages that differ in syntax (different WSDL descriptions), however, the services are semantically the same and should be represented with one semantic service in the service ontology with two different groundings. This will illustrate the use of different devices to implement the same service as well as switching between implementations of a service based on availability.

This scenario resulted in some issues that have to be resolved:

- How do we represent that there are two groundings for the same semantic service. Will there be two services in the service ontology or one with two groundings?
- How do the device service WSDLs (which can be retrieved from the device) reference the semantic service description in the service ontology?
- For devices that have a fixed set of services, how do we reference the services that are implemented?
- For devices that can implement any service (Smartphones, PDAs, personal computers), do we represent this in the device ontology? How do we identify if a service provided by such a device is in the service ontology)?
- What type of annotations is needed?

- What is the most basic / primitive form of service description we will handle? e.g., a name/id in the service ontology and a link to a WSDL exists? (i.e., no semantics except identification and possible ISA relations).

4.3 Ontology-based Device and Service Descriptions

4.3.1 Requirements

ID	Description	Rationale
114	Semantic enabling of device web services	Middleware should be able to attach semantic descriptions to device web services based on device ontology.
389	Service browsing in device ontology	It must be possible to view services as central building blocks, thus an application developer should be able to browse the device ontology from a service perspective, in addition to a device perspective.

Table 2: Requirements on Device and Service Descriptions

Hydra models services separate from devices, by representing them in two related models represented by the Device Ontology (DO) and the Service Ontology (SO). Certain devices are obviously permanently bound to the services they provide (thermometer device – get temperature) whereas others, e.g. of the SmartPDA device type, are not.

The Hydra Device Ontology is currently developed using the OWL language, see deliverable D6.2 "MDA Design Document".

4.3.2 Implications of different SWS framework approaches on Hydra

4.3.2.1 OWL-S

Using the OWL-S approach to existing OWL ontology is quite straightforward. The OWL-S standard is built as the extension of OWL language, thus there is no formal language compatibility problem. OWL-S, from its definition, is capable to link required concepts to existing OWL classes describing the taxonomy structures, inputs, outputs, preconditions and effects (IOPEs) of service and service capabilities modeled in existing device ontology. Actual suggested service model described in D6.2 "MDA Design Document" deliverable is explicitly inspired by OWL-S approach and can be directly substituted by OWL-S ontology describing the services linked to main device concept.

Unfortunately it quickly gets complicated to describe even simple services. The reason for this is of course that OWL-S has been designed with complex business processes in mind and not specifically for devices with rather simple functions, like turning lights on and off.

Below we show how our example SMS service would be described in OWL-S. The example below expresses that if the phone number is OK the SMS is sent and if it is not OK then "notify error" is called. As can be seen from the example this is rather complicated to express using OWL-S.

.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
xml:base="http://hydra.cnet.se/SMSService/SMSService_ProcessModel.owl#"
xmlns:dataflow="http://jamsci.servehttp.com/owlsedit/Dataflow.owl#"
xmlns:drs="http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/drsonto
040112.owl#"
xmlns:owl="http://jamsci.servehttp.com/owlsedit/owl.rdf#"
xmlns:process="http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/owl
s11/Process.owl#"
xmlns:profile="http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/owl
s11/Profile.owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

```
xmlns:rdfs="http://jamsci.servehttp.com/owlsedit/rdf-schema.rdf#"
xmlns:service="http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/owl
s11/Service.owl#"
xmlns:swrl="http://jamsci.servehttp.com/owlsedit/swrl.owl#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
<owl:Ontology rdf:about="">
<owl:versionInfo>Version 1.0</owl:versionInfo>
<rdfs:comment>Add Ontology Comment</rdfs:comment>
        <owl:imports rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns"/>
        <owl:imports rdf:resource="http://jamsci.servehttp.com/owlsedit/owl.rdf"/>
        <owl:imports rdf:resource="http://jamsci.servehttp.com/owlsedit/rdf-
        schema.rdf"/>
        <owl:imports
        rdf:resource="http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlsedit
        FYP/owls11/Service.owl"/>
        <owl:imports
        rdf:resource="http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlsedit
        FYP/owls11/Profile.owl"/>
        <owl:imports
        rdf:resource="http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlsedit
        FYP/owls11/Process.owl"/>
</owl:Ontology>
<process:ProcessModel rdf:ID="SMSService_ProcessModel">
        <process:hasProcess rdf:resource="#SMSService_Composite"/>
        <process:describes
        rdf:resource="http://hydra.cnet.se/SMSService/SMSService_Service#SMSService_S
        ervice"/>
</process:ProcessModel>
<process:CompositeProcess rdf:ID="SMSService_Composite">
        <process:composedOf>
                <dataflow:TagBind>
                        <dataflow:tagBound>
                        <dataflow:TagNames>
                        <dataflow:tagName
                        rdf:datatype="http://jamsci.servehttp.com/owlsedit/XMLSchema.x
                        sd#string">condition_0</dataflow:tagName>
                        </dataflow:TagNames>
                        </dataflow:tagBound>
                </dataflow:TagBind>
        <process:Sequence rdf:ID="CheckNumber_SEQ_Head">
                <process:components rdf:parseType="Collection">
                        <dataflow:Call>
                                <dataflow:callee
                        rdf:resource="#SMSPortType_CheckNumber"/>
                                <dataflow:tag
                                rdf:datatype="http://jamsci.servehttp.com/owlsedit/XMLS
                                chema.xsd#string">condition_0</dataflow:tag>
                        </dataflow:Call>
                </process:components>
                <dataflow:flow/>
        </process:Sequence>
        <process:If-Then-Else rdf:ID="IsValidNumber">
                <process:ifCondition>
                        <drs:bound_vars rdf:parseType="Collection">
                        <swrl:Variable drs:name="condition_0" rdf:ID="condition_0">
                                <drs:declare
                                rdf:resource="#SMSPortType_CheckNumber_CheckNumberResul
                                t_OUT"/>
                        </swrl:Variable>
                        </drs:bound_vars>
                        <drs:Atomic_Formula>
                                <rdf:subject rdf:resource="#condition_0"/>
                                <rdf:predicate
                                rdf:resource="http://jamsci.servehttp.com/owlsedit/awol
                                .rdf#equalTo"/>
                                <rdf:object>
                                <drs:value
                                rdf:type="http://www.w3.org/2001/XMLSchema#boolean">tru
                                e</drs:value>
                                </rdf:object>
                        </drs:Atomic_Formula>
                </process:ifCondition>
                <process:then>
                        <process:Sequence rdf:ID="SendSMS_SEQ_Head">
                        <process:components rdf:parseType="Collection">
                        <dataflow:Call>
```

```
                        <dataflow:callee rdf:resource="#SMSPortType_SendSMS"/>
                        </dataflow:Call>
                        </process:components>
                        <dataflow:flow/>
                        </process:Sequence>
                </process:then>
                <process:else>
                        <process:Sequence rdf:ID="NotifyError_SEQ_Head">
                                <process:components rdf:parseType="Collection">
                                        <dataflow:Call>
                                        <dataflow:callee
                                rdf:resource="#SMSPortType_NotifyError"/>
                                        </dataflow:Call>
                                </process:components>
                                <dataflow:flow/>
                        </process:Sequence>
                </process:else>
        </process:If-Then-Else>
        </process:composedOf>
</process:CompositeProcess>
<process:Input rdf:ID="SMSPortType_SendSMS_number_IN">
        <process:parameterName>SMSPortType_SendSMS_number_IN</process:parameterName>
        <process:parameterType
        rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</process:Input>
<process:Input rdf:ID="SMSPortType_SendSMS_message_IN">
        <process:parameterName>SMSPortType_SendSMS_message_IN</process:parameterName>
        <process:parameterType
        rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</process:Input>
<process:Output rdf:ID="SMSPortType_SendSMS_SendSMSResult_OUT">
        <process:parameterName>SMSPortType_SendSMS_SendSMSResult_OUT</process:paramet
        erName>
        <process:parameterType
        rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
</process:Output>
<process:AtomicProcess rdf:ID="SMSPortType_SendSMS">
        <process:hasInput rdf:resource="#SMSPortType_SendSMS_number_IN"/>
        <process:hasInput rdf:resource="#SMSPortType_SendSMS_message_IN"/>
        <process:hasResult>
                <process:Result>
                        <process:hasOutput
                        rdf:resource="#SMSPortType_SendSMS_SendSMSResult_OUT"/>
                </process:Result>
        </process:hasResult>
</process:AtomicProcess>
<process:Input rdf:ID="SMSPortType_CheckNumber_number_IN">
        <process:parameterName>SMSPortType_CheckNumber_number_IN</process:parameterNa
        me>
        <process:parameterType
        rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</process:Input>
<process:Output rdf:ID="SMSPortType_CheckNumber_CheckNumberResult_OUT">
        <process:parameterName>SMSPortType_CheckNumber_CheckNumberResult_OUT</process
        :parameterName>
        <process:parameterType
        rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
</process:Output>
<process:AtomicProcess rdf:ID="SMSPortType_CheckNumber">
<process:hasInput rdf:resource="#SMSPortType_CheckNumber_number_IN"/>
<process:hasResult>
        <process:Result>
                <process:hasOutput
                rdf:resource="#SMSPortType_CheckNumber_CheckNumberResult_OUT"/>
        </process:Result>
</process:hasResult>
</process:AtomicProcess>
<process:Input rdf:ID="SMSPortType_NotifyError_number_IN">
        <process:parameterName>SMSPortType_NotifyError_number_IN</process:parameterNa
        me>
        <process:parameterType
        rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</process:Input>
<process:Input rdf:ID="SMSPortType_NotifyError_message_IN">
        <process:parameterName>SMSPortType_NotifyError_message_IN</process:parameterN
        ame>
```

```
                <process:parameterType
                rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
        </process:Input>
        <process:Output rdf:ID="SMSPortType_NotifyError_NotifyErrorResult_OUT">
                <process:parameterName>SMSPortType_NotifyError_NotifyErrorResult_OUT</process
                :parameterName>
                <process:parameterType
                rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
        </process:Output>
        <process:AtomicProcess rdf:ID="SMSPortType_NotifyError">
                <process:hasInput rdf:resource="#SMSPortType_NotifyError_number_IN"/>
                <process:hasInput rdf:resource="#SMSPortType_NotifyError_message_IN"/>
                <process:hasResult>
                        <process:Result>
                                <process:hasOutput
                                rdf:resource="#SMSPortType_NotifyError_NotifyErrorResult_OUT"/
                                >
                        </process:Result>
                </process:hasResult>
        </process:AtomicProcess>
        </rdf:RDF>
```

Figure 5: Service description using OWL-S

4.3.2.2 WSMO

The example below shows a service description using WSMO's language WSML.

```
fVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"
namespace { _"http://www.example.org/ontologies/example#",
                        dc _"http://purl.org/dc/elements/1.1#",
                        foaf _"http://xmlns.com/foaf/0.1/",
                        wsml _"http://www.wsmo.org/wsml/wsml-syntax#",
                        loc _"http://www.wsmo.org/ontologies/location#",
                        oo _"http://example.org/ooMediator#" }

/*****************************
* WEBSERVICE
*****************************/
webService _"http://example.org/Germany/BirthRegistration"
        nfp
                dc#title hasValue "Birth registration service for Germany"
                dc#type hasValue _"http://www.wsmo.org/TR/d2/v1.2/#services"
                wsml#version hasValue "$Revision: 1.1 $"
        endnfp
        importsOntology { _"http://www.example.org/ontologies/example",
                        _"http://www.wsmo.org/ontologies/location" }

        capability _"http://example.org/Germany/BirthRegistration#cap1"
                sharedVariables ?child

                precondition
                        nonFunctionalProperties
                                dc#description hasValue "The input has to be boy or a girl
                                  with birthdate in the past and be born in Germany."
                        endNonFunctionalProperties
                        definedBy
                                ?child memberOf Child
                                and ?child[hasBirthdate hasValue ?brithdate]
                                and wsml#dateLessThan(?birthdate,wsml#currentDate())
                                and ?child[hasBirthplace hasValue ?location]
                                and ?location[locatedIn hasValue oo#de]
                                or (?child[hasParent hasValue ?parent]
                                and?parent[hasCitizenship hasValue oo#de] ) .

                assumption
                        nonFunctionalProperties
                                dc#description hasValue "The child is not dead"
                        endNonFunctionalProperties
                        definedBy
                                ?child memberOf Child
                                and naf ?child[hasObit hasValue ?x].

                effect
                        nonFunctionalProperties
                                dc#description hasValue "After the registration the
child
```

```
                                    is a German citizen"
                            endNonFunctionalProperties
                            definedBy
                                    ?child memberOf Child
                                    and ?child[hasCitizenship hasValue oo#de].

interface _"http://example.org/Germany/BirthRegistration#iface1"
          choreography _"http://example.org/tobedone"
          orchestration _"http://example.org/tobedone"
```

Figure 6: Service description using WSMO

Using the WSMO approach to an existing OWL model has several drawbacks. Even though the WSMO provides a really impressive conceptual model for service description, it uses the WSML language. WSML enables to refer to existing OWL category, IOPEs and capability concepts, as WSML requires only a URI for reference representation. It means, that WSML does not specify exactly, that the concepts referred by URI have to be WSML concepts.

But, describing service semantics using WSML language may cause several implementation problems mainly in the run-time caused mainly by WSML references to OWL concepts. As the WSMO reasoning engine requires the model, which is fully described in WSML, there are two main possibilities, how to adapt the references from WSML to OWL concepts:

- In run-time, the OWL ontology has to be transformed to a WSMO ontology to enable the reasoner to take into account all referred concepts. The mapping approach is described for example in [31]. As the content of device ontology containing referred concepts may change also in run-time, this approach does not seem to be suitable, because the transformation would have to be done after any change in device ontology content to actualize actual WSML model used.
- The second possibility is the use of WSMO mediators for transformation of referred OWL concepts to WSML at run-time. This approach seems to be more suitable, but, logically, can be more time expensive.

4.3.2.3 SAWSDL

As the SAWSDL does not provide the service semantics description itself, this approach would require development of a custom service model, which satisfies the Hydra specific requirements for services. This model – the service ontology – should be linked to the device ontology. SAWSDL semantic annotations should be linked to particular service ontology concepts.

Below is an example of a WSDL file for our example SMS service that has been annotated using SA-WSDL.

```
<?xml version="1.0" ?>
<definitions name="SMSService"
targetNamespace="http://Hydra.cnet.se/SMSService.wsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://Hydra.cnet.se/SMSService.wsdl"
xmlns:sawsdl="http://www.w3.org/ns/sawsdl">
<message name="SendSMSSoapIn">
        <part name="number" type="xsd:string"
        sawsdl:modelReference="http://hydra.cnet.se/serviceOntology/SMSService#Teleph
        oneNumber"
        sawsdl:liftingSchemaMapping="http://hydra.cnet.se/serviceOntology/SMSService/
        StringToTelephoneNumber.lifting.xslt"
        sawsdl:loweringSchemaMapping="http://hydra.cnet.se/serviceOntology/SMSService
        #TelephoneNumberToString.lowering"/>
        <part name="message" type="xsd:string" />
</message>
<message name="SendSMSSoapOut">
         <part name="SendSMSResult" type="xsd:boolean"
        sawsdl:modelReference="http://hydra.cnet.se/serviceOntology/SMSService#Operat
        ionSuccessful"/>
</message>
<message name="CheckNumberSoapIn">
```

```
        <part name="number" type="xsd:string"
        sawsdl:modelReference="http://hydra.cnet.se/serviceOntology/SMSService#Teleph
        oneNumber"
        sawsdl:liftingSchemaMapping="http://hydra.cnet.se/serviceOntology/SMSService/
        StringToTelephoneNumber.lifting.xslt"
        sawsdl:loweringSchemaMapping="http://hydra.cnet.se/serviceOntology/SMSService
        #TelephoneNumberToString.lowering"/>
</message>
<message name="CheckNumberSoapOut">
        <part name="CheckNumberResult" type="xsd:boolean"
        sawsdl:modelReference="http://hydra.cnet.se/serviceOntology/SMSService#Operat
        ionSuccessful"/>
</message>
<message name="NotifyErrorSoapIn">
        <part name="number" type="xsd:string"
        sawsdl:modelReference="http://hydra.cnet.se/serviceOntology/SMSService#Teleph
        oneNumber"
        sawsdl:liftingSchemaMapping="http://hydra.cnet.se/serviceOntology/SMSService/
        StringToTelephoneNumber.lifting.xslt"
        sawsdl:loweringSchemaMapping="http://hydra.cnet.se/serviceOntology/SMSService
        #TelephoneNumberToString.lowering"/>
        <part name="message" type="xsd:string" />
</message>
<message name="NotifyErrorSoapOut">
        <part name="NotifyErrorResult" type="xsd:boolean" />
</message>
<portType name="SMSPortType"
sawsdl:modelReference="http://hydra.cnet.se/serviceOntology/SMSService">
        <operation name="SendSMS">
                <sawsdl:attrExtensions
                sawsdl:modelReference="http://hydra.cnet.se/serviceOntology/SMSService
                #SendSMS"/>
                <input message="tns:SendSMSSoapIn" name="SendSMS" />
                <output message="tns:SendSMSSoapOut" name="SendSMSResponse" />
        </operation>
        <operation name="CheckNumber">
                <sawsdl:attrExtensions
                sawsdl:modelReference="http://hydra.cnet.se/serviceOntology/SMSService
                #ValidateTelephoneNumber"/>
                <input message="tns:CheckNumberSoapIn" name="CheckNumber" />
                <output message="tns:CheckNumberSoapOut" name="CheckNumberResponse" />
        </operation>
        <operation name="NotifyError">
                <sawsdl:attrExtensions
                sawsdl:modelReference="http://hydra.cnet.se/serviceOntology/SMSService
                #NotifyError"/>
                <input message="tns:NotifyErrorSoapIn" name="NotifyError" />
                <output message="tns:NotifyErrorSoapOut" name="NotifyErrorResponse" />
        </operation>
  </portType>
  <binding name="SMSSoapBinding" type="tns:SMSPortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="SendSMS" >
      <soap:operation soapAction="" />
      <input>
        <soap:body use="encoded" namespace="urn:SMS-SoapServices"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
      <output>
        <soap:body use="encoded" namespace="urn:SMS-SoapServices"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
    <operation name="CheckNumber">
      <soap:operation soapAction="" />
      <input>
        <soap:body use="encoded" namespace="urn:SMS-SoapServices"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
      <output>
        <soap:body use="encoded" namespace="urn:SMS-SoapServices"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
    <operation name="NotifyError">
      <soap:operation soapAction="" />
```

```
      <input>
        <soap:body use="encoded" namespace="urn:SMS-SoapServices"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
      <output>
        <soap:body use="encoded" namespace="urn:SMS-SoapServices"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
  </binding>
  <service name="SMSSoapService">
    <documentation></documentation>
    <port name="SMSSoapPort" binding="tns:SMSSoapBinding">
      <soap:address
      location="http://hydra.cnet.se/SMSService/SmsService.asmx"/>
    </port>
  </service>
</definitions>
```

Figure 7: Service description using SAWSDL

The example above is an annotated WSDL for the SMS service. The *modelReference* attribute is used to identify the port type "SMSPortType" as the "SMSService" from the service ontology. The operations "SendSMS", "CheckNumber" and "NotifyError" are described by the concepts "SendSMS", "ValidateTelephoneNumber" and "NotifyError" in the semantic model defined externally. The message part "number" is identified as a "TelephoneNumber" from the ontology, and a lowering and lifting schema mapping is defined to associate the string in the WSDL with transformations to and from the corresponding "TelephoneNumber" concept in the semantic model.

4.3.3 Hydra approach: SAWSDL combined with service ontology

For all above mentioned approaches, the developer of semantic markup of Hydra services must have the knowledge about existing Hydra ontologies. When creating the services semantic markup, in case of OWL-S and WSMO approach, the service description should be done manually. In the case of SAWSDL approach, the service models can be developed manually, but there is also the real possibility to support the generation of service descriptions semi-automatically, or fully automatically using the SAWSDL annotations. Moreover, device manufacturers may extend the device services WSDL with semantic annotations enabling for example semi or fully automatic processing of SAWSDL file in order to update the Hydra device or service ontology.

As the standard for modelling of Hydra services, it is possible to use both OWL-S and WSMO technology, which enable to solve all of the tasks that have to be solved by the first iteration of Hydra services. Both approaches provide an acceptable solution. OWL-S, as the well known standard seems to be more mature in various aspects. WSMO provides a more complete conceptual model, but its specification and implementation is still incomplete and in development. In addition, as the solution to character of initially defined tasks of service discovery, explicit composition and invocation, both standards seem to be overly complex for the needs of Hydra. When searching for simple and practical solution, OWL-S or WSMO approach should be used mainly in the case, when there is the need for modelling of such a complex issues as:

- reasoning with the service preconditions and effects or
- service orchestration with ability of searching the services in the work-flow on the fly

A simple and practical solution for the first prototype of Hydra services seems to be using SAWSDL to provide annotations referring to the custom service model. The custom Hydra service model can be developed as a simplified, but further extensible ontology inspired by the OWL-S or WSMO standards. The development of service ontology must take into account the future extension of Hydra requirements on the services. It should be also possible to completely substitute the custom Hydra service ontology with selected SWS standard, such as OWL-S or WSMO.

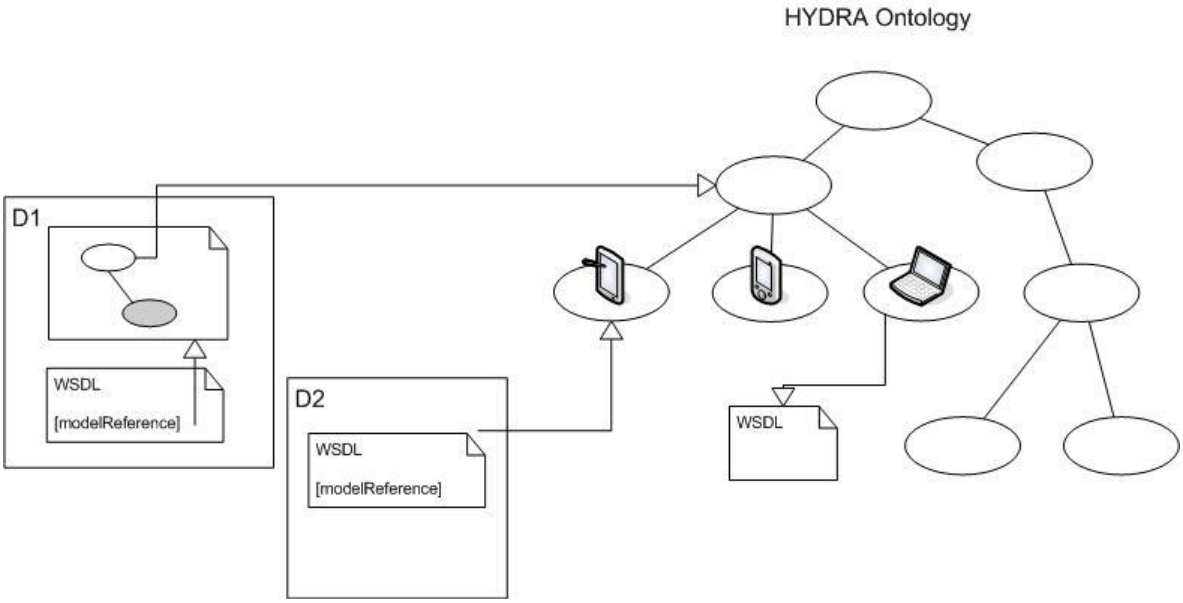

Figure 8: The Hydra approach in the first iteration is based on SAWSDL with references to the ontology

The Hydra service ontology needs to be specified from functional and structural view independently of underlying SWS modelling formalism. Service ontology has to represent the model of various device services. Models of these services have to be used as the basic information required to solve the tasks of service discovery, composition (it is assumed, that the composition in first service prototype will be static) and service invocation.

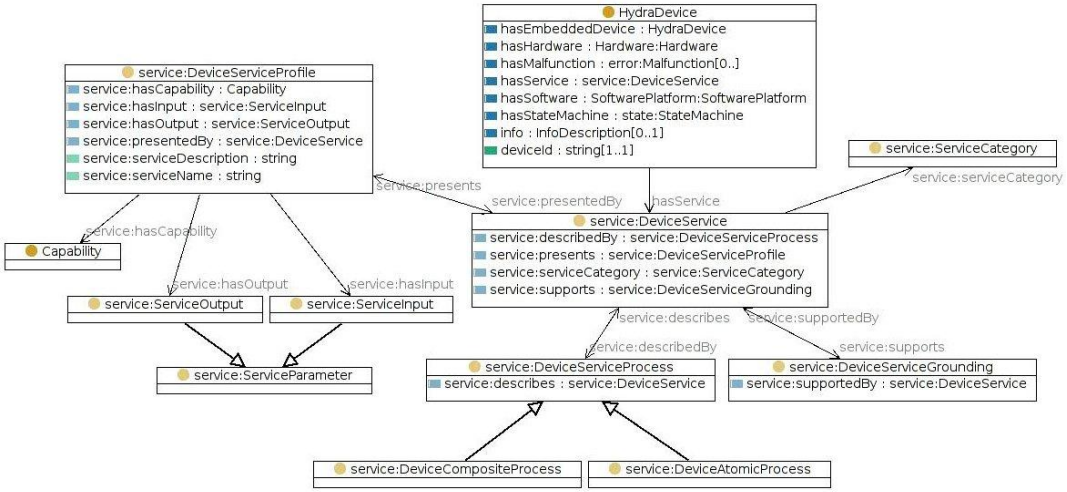

Figure 9: Service Ontology in Hydra

As the character of services provided by various devices can differ, the model has to take into account the various aspects. As mentioned in the SMS service example, the device services models should focus on two general cases:

- the services are permanently bound to device (e.g. every smart phone provides the SMS service)
- the services may, but do not have to be provided by the device (e.g. smart PDA may provide the SMS service, but does not have to or services, which are permanently bound to device, but are not implemented)

This setup implies more problems, which solutions should be supported by the service ontology. According to device ontology, devices are modeled as the hierarchy of various device types, in

addition, there are attached various properties and capabilities to each device (or each device type). The basic issues, which have to be addressed, can be outlined as follows:

- Various device models of the same type (e.g. smart PDA) may provide various services (e.g. some PDAs provide SMS service, some do not).
- Various devices (device types) may provide the services, which can be the same from the view of functionality and the input, output types (e.g. SMS service, which requires the phone number and message text as the inputs, sends the message and returns delivery report as the output is provided by the smart phone, but also by smart PDA device).
- Various devices (device types) or even the device itself may provide the services, which can be the same from the functionality view, but input and output types are different (e.g. the device provides two SMS services with various input variables).

There are many possible solutions to the described setup. The solutions should be based on the assumptions, what have to be modeled. The suggestion is that the service ontology should model the following issues:

- Service category representing the classification of services by their functionality or capabilities. The service ontology may also contain the taxonomy of possible service operations.
- Service inputs and outputs (and, possibly the preconditions and effects) represented by ontology concepts as the single (text, number, etc.) or complex (person name, address, etc.) type. These I/O properties have to be linked to the service grounding model (with respect to mediation requirements). Service grounding model should also contain the reference to the related real device WSDL (or, possibly SAWSDL) file enabling the service invocation.
- Service capabilities representing for example the functional device properties (e.g. device is able to send SMS), required capabilities (e.g. specific security requirements), etc.

The first modelling approach is to have service taxonomy, similar to device taxonomy, which will represent all possible services. The model of each service should contain references to service capabilities and I/O properties. Each specific device concept will refer to the set of related service concepts in taxonomy. Each service model should contain the specific information on the service grounding, which should describe the data mediation information and the link to the service WSDL file. In the case of new device, which provides the new, not yet modeled service, the service ontology has to be extended with the new concept representing the new service in taxonomy.

If the device joins the Hydra network, there is the expectation, that the device has implemented all of the modeled services and provides the WSDL files, which match the modeled services, operations and their input/output properties. In this basic case, a new ontology instance of device is created, for each service linked in the ontology, the new instance is created and filled with link to particular WSDL file.

If a device provides a set of permanently bound services, but not all services are implemented or if the device may provide any service (e.g. PDA or PC), additional information to create the resulting model of device services has to be provided. In this case, SAWSDL annotations are helpful. According to parsed SAWSDL information, the actual run-time device model can be created automatically. The required assumption is that SAWSDL file will contain all references to existing concepts in service ontology.

4.4 Semantic Discovery of Networked Devices and Services

One of the contributions from Hydra is to merge UPnP (Universal Plug and Play) discovery of networked devices with semantic services, allowing UPnP-enabled devices to act as semantic web services towards the network.

4.4.1 Requirements

ID	Description	Rationale
104	Automatic Discovery of Services	It should be possible to configure the middleware to discover available services that meets defined criteria.
129	Support for Semantic Web Standards for Device Communication	Middleware should support different semantic web standards, including OWL-S, WSMO, and selected parts of WS-*
158	There should be a hook-up-service	When the developer creates a new application/device he wants to have a broker that can supply him with all available services that match certain criteria.
159	Service brokers must be organized in a hierarchical way	With hierarchical brokers the system becomes more robust and scalable. Users do not want that everything acts up in case of a fire and a broker goes down. Additionally hierarchical brokers allow for having certain rules/services only within a sub-domain.
160	Search masks for device/service discovery	When the developer needs a service he wants to be able to define search criteria for discovery of services
164	Support for Service standards	Middleware should support widely used standards for service description, discovery, orchestration and execution.
196	Basic Service Registry	Services should register at a basic service/module of the middleware in order to provide a base for service orchestration
198	A service broker is responsible to provide services according to specific keywords	Service discovery should be enhanced by a service broker module/service as basic service of the middleware that enables the search for services according to specific keywords
207	Service selection by context	In order to select an appropriate service for a specific task, contextual information, like the spatial position, must be taken into account. Hydra must provide a method to specify a desired service by contextual parameters. For example, if a certain room in a building is specified in a search request for a service, only services are returned that are relevant in the current user's location and context.
209	Middleware has a service for providing information about the technical environment/infrastructure	In order for the services to query the available infrastructure the middleware should provide such a service
419	Backbone - Device services and resources announcement through the Gateway	Each device either Hydra-enabled or non-Hydra-enabled (through proxies) must announce its services and resources in the Backbone through its Gateway

Table 3: Requirements on Device and Service Discovery

4.4.2 Discovery issues

An important aspect of all ambient intelligence applications is for users, applications and devices to quickly and easily discover devices that are available in there vicinity. The first issue is to discover the existence of a device that one can communicate with, the second issue is to discover what type services the device offers and thirdly to discover how to access and execute these services.

4.4.3 Hydra Approach: Combining UPnP with semantics for discovery

The overall discovery approach in Hydra is based on the combination of UPnP for networked devices with semantic services, allowing UPnP-enabled devices to act as semantic web services towards the network. To this end, we will use a layered approach to discovery, handling discovery at three layers – *physical*, *network* and *semantic*.

4.4.3.1 A layered discovery architecture

At the lowest level the Hydra project is developing techniques for the discovery at the physical level. This will allow us to discover devices using communication protocols like Bluetooth, ZigBee, WiFi etc. This part of the work is carried out as part of workpackage 5 "Wireless Networks and Devices" in task 5.3 "Wireless Devices".

Once a device has been discovered at the physical level we need to discover it at a network level. This is done by creating a UPnP (Universal Plug and Play) wrapper to represent the device on the network. The UPnP wrapper then allows the device to be discovered at a network layer. The UPnP discovery process is further described in the next section.

The UPnP wrapper is part of the "Hydra Proxy" in Figure 10 below.

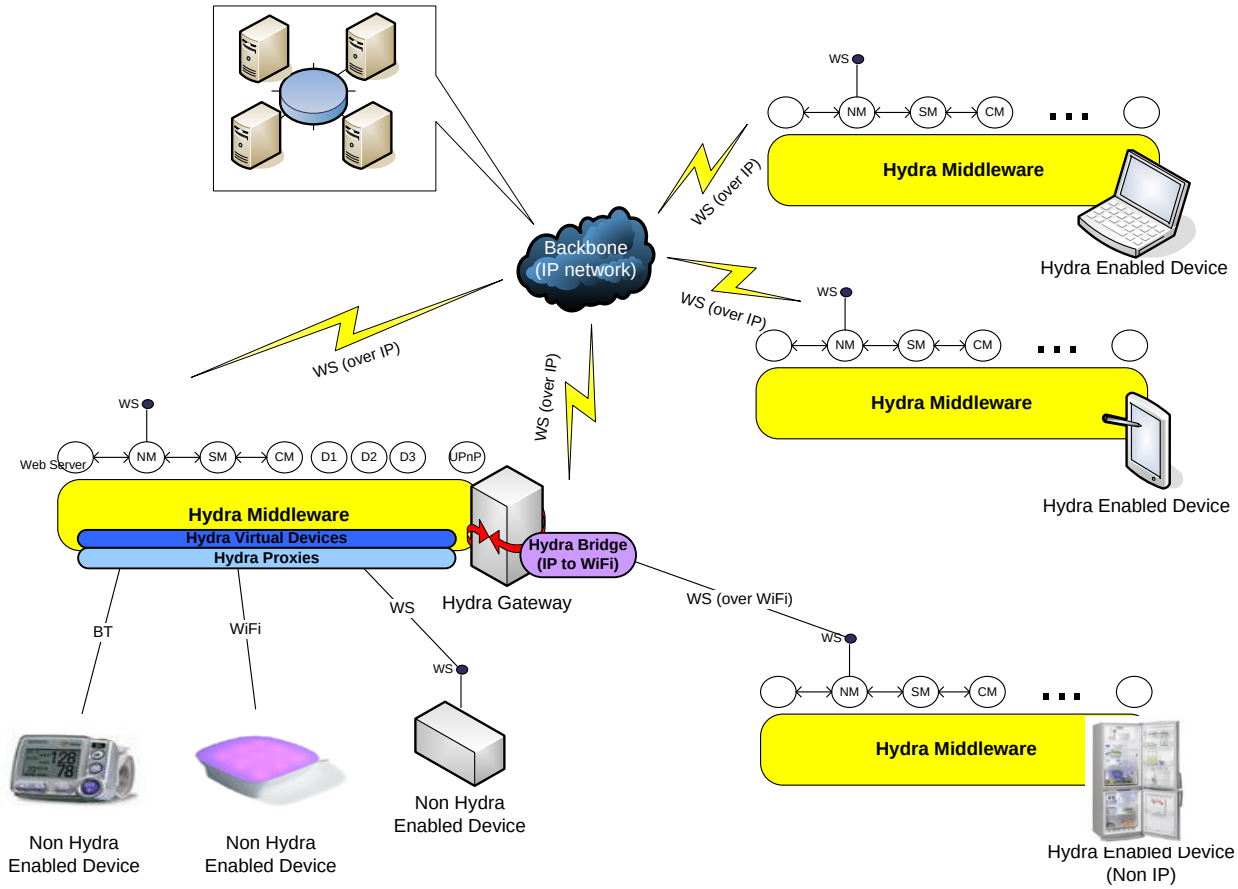

Figure 10: A Hydra network and its components

Once the device is discovered as part of the network, we then need to discover it from a semantic point of view, i.e., we need to relate the device to the Hydra Device Ontology so that we know what kind of device we have discovered, this is also further described below.

In WP 6 and this deliverable we are mainly concerned with the semantic discovery and how this is combined with the UPnP device discovery.

4.4.3.2 UPnP-based Device Discovery

The UPnP (Universal Plug and Play) architecture offers pervasive peer-to-peer network connectivity of PCs, intelligent appliances and wireless devices. The UPnP architecture is a distributed, open networking architecture that uses TCP/IP and HTTP. It enables seamless proximity networking in addition to data transfer between networked devices at home, in the office and everywhere in between.

It enables data communication between any two devices under the command of any control device in the network.

UPnP has a number of characteristics:

- *Media and device independence.* UPnP technology can run on any medium including phone lines, power lines, Ethernet, IR (IrDA), RF (WiFi, Bluetooth), and FireWire. No device drivers are used; common protocols are used instead.
- *Common base protocols*. Base protocol sets (Device Control Protocols, DCP) are used, on a per-device basis.
- *User interface (UI) Control*. UPnP architecture enables vendor control over device user interface and interaction using the web browser.
- *Operating system and programming language independence*. Any operating system and any programming language can be used to build UPnP products. UPnP does not specify or constrain the design of an API for applications running on control points. OS vendors may create APIs that suit their customer's needs. UPnP enables vendor control over device UI and interaction using the browser as well as conventional application programmatic control.
- *Internet-based technologies*. UPnP technology is built upon IP, TCP, UDP, HTTP, SOAP and XML, among others.
- *Programmatic control*. UPnP architecture also enables conventional application programmatic control.
- *Extensibility*. Each UPnP product can have value-added services layered on top of the basic device architecture by the individual manufacturers.

The UPnP architecture supports zero-configuration, invisible networking and automatic discovery for a breadth of device categories from a wide range of vendors. Devices can dynamically join a network, obtain IP addresses, announce their names, convey their capabilities upon request, and learn about the presence and capabilities of other devices. DHCP and DNS servers are optional. A device can leave a network smoothly and automatically without leaving any unwanted state information behind.

UPnP relies on standardised DCPs (Device Control Protocols) that define the interface to different devices in an UPnP network. UPnP uses two different XML structures to describe a device and its capabilities. First there is the device description which contains various metadata regarding the device such as its type, the manufacturer, model etc. An example is shown below:

```
<device>
        <deviceType>urn:schemas-upnp-org:device:waterPump:1</deviceType>
        <friendlyName>GrundfosPump</friendlyName>
        <manufacturer>Grundfos</manufacturer>
        <manufacturerURL>http://www.grundfos.com</manufacturerURL>
        <modelDescription>Pump</modelDescription>
        <modelName>Grundfos Magna</modelName>
        <modelNumber>X1</modelNumber>
        <UDN>uuid:dac824ab-bca1-4d5c-93c5-578a0c697ba1</UDN>
        <serviceList>
                <service>
                <serviceType>urn:schemas-upnp-
                org:service:grundfosPumpService:1</serviceType>
                <serviceId>urn:upnp-org:serviceId:grundfosPumpService</serviceId>
                <SCPDURL>_grundfosPumpService_scpd.xml</SCPDURL>
                <controlURL>_grundfosPumpService_control</controlURL>
                <eventSubURL>_grundfosPumpService_event</eventSubURL>
                </service>
```

```
        </serviceList>
</device>
```

Figure 11: UPnP Device description (pump).

Secondly, there is the SCPD (Service Control Point Description), which describes the capabilities of the device and how to invoke its different services:

```
<?xml version="1.0" encoding="utf-8"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
   <specVersion>
      <major>1</major>
      <minor>0</minor>
   </specVersion>
   <actionList>
      <action>
         <name>GetStatus</name>
         <argumentList>
            <argument>
               <name>ResultStatus</name>
               <direction>out</direction>
               <relatedStateVariable>Status</relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>SetTarget</name>
         <argumentList>
            <argument>
               <name>newTargetValue</name>
               <direction>in</direction>
               <relatedStateVariable>Target</relatedStateVariable>
            </argument>
         </argumentList>
      </action>
   </actionList>
   <serviceStateTable>
      <stateVariable sendEvents="yes">
         <name>Status</name>
         <dataType>boolean</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>Target</name>
         <dataType>boolean</dataType>
      </stateVariable>
   </serviceStateTable>
</scpd>
```

Figure 12: UPnP Service description

4.4.3.3 Service Discovery

The goal of service discovery task is to find a suitable service provided by specific device (or device type) in accordance to defined requirements. The requirements of desired service are usually specified by the following terms (represented as the inter-related ontology concepts):

- Service category: the specification of service position in the service taxonomy, which represents the classification of services by their capabilities or usage purposes. Generally, the category information tends to reduce the complexity of the discovery process.
- Service inputs and outputs: specification of concepts required on the inputs and the outputs of service represented by ontology concepts as the single (text, number, etc.) or complex types (person name, address, etc.).
- Service preconditions and effects: usually the rules expressing the constraints, which have to be satisfied to enable the service invocation and the effects which service invocation causes. Preconditions and effects usually refer the run-time values (and the value changes in the case of effects) in the ontology.
- Service capabilities: similarly, as in the case of category information, specification of service capabilities (for example, service is capable to send SMS, service is capable to play audio, etc.) should lead to reduced complexity of the discovery process. Capability information may

also define the special requirements on the service, such as capability of using required communication protocols or satisfying specified security requirements.

In the context of Hydra, the service discovery task defined this way can be used in various cases, for example:

- From a developer user point of view: to find the required service provided by specific device in the process of development of basic communication patterns, such as composed (or orchestrated) services, choreography interfaces or service user interfaces.
- From a system or application point of view: to find the required service provided by specific device when executing the complex process requiring the service orchestration.

There are existing tools and matchmakers supporting the service discovery for both OWL-S and WSMO standards (description of this tools is out of scope of this deliverable), which may be used for particular approach. The issue, which should be especially addressed, is the support of using the IOPEs for service discovery. In the real applications, IOPEs are not used properly, because the reasoning with preconditions and effects in real-time discovery process is very time expensive. Usually, the potential preconditions and effects are skipped or pre-computed.

As the SAWSDL approach does not explicitly support service discovery, there are two basic possibilities, which can be used in this case:

- The Service discovery process is realized by searching the SAWSDL according to provided semantic annotations.
- Using the annotations in SAWSDL file, the model of service is annotated in the Hydra service ontology and the discovery process is realized by matching the ontology concepts in accordance to specified requirements, similarly as in OWL-S/WSMO approach.

4.5 Lightweight Orchestration of Device Services

4.5.1 Requirements

ID	Description	Rationale
113	Composition (of services and devices)	In order to enhance or replace application level functions it will be useful to be able to compose services and devices from different providers and/or manufacturers into high level services/devices
129	Support for Semantic Web Standards for Device Communication	Middleware should support different semantic web standards, including OWL-S, WSMO, and selected parts of WS-*
157	Availability of combined services	A developer wants to easily access a higher level service which is in fact a combination of multiple services
164	Support for Service standards	Middleware should support widely used standards for service description, discovery, orchestration and execution.
196	Basic Service Registry	Services should register at a basic service/module of the middleware in order to provide a base for service orchestration
198	A service broker is responsible to provide services according to specific keywords	Service discovery should be enhanced by a service broker module/service as basic service of the middleware that enables the search for services according to specific keywords
211	There are components/services in the middleware that integrate	The integration of basic systems to subsystems should ease the configuration of higher level services. Higher level services could

	subsystems	then consist of a combination of basic systems
216	The middleware should have a graceful degradation service	Services should be organised in a cascade of services in order to allow an orchestration of services providing best possible services down to basic services automatically, according to their availability
290	Share service orchestration between users	Service orchestration definition should be shared between developer users, in order to allow a distribution of useful service orchestration to other developers
394	Stateful service orchestration	In order to specify service workflows we need to be able to keep state between the execution of the stateless services.

Table 4: Requirements on Orchestration

4.5.2 Orchestration in a Service-Oriented Architecture

In a service oriented architecture (SOA) some of the key aspects are loose coupling of services, implementation neutrality, flexible configurability and coarse granularity. If a service designer has those goals in mind while defining the scope of one service it will usually a rather low-level service from a functionality point of view. In order to create higher-level services one has to define in which order other services will be consumed and then find and execute them.

So within a SOA service discovery is only the starting point. To create useful applications on the SOA architecture style one has to orchestrate services to support workflows that were previously defined as well as creating composite services out of existing lower level services as can be seen in the SOA example in figure 4 (examples for orchestration at the bottom and composition at the top).

The *orchestration* of a sequence of services can be done by several technologies. WS-BPEL is the extension of BPEL (Business Process Execution Language) to web services but is recognized to be rather complex and most probably not suited to the requirements of Hydra. One example of using BPEL is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
   <process xmlns=http://docs.oasis-open.org/wsbpel/2.0/process/executable
   xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
   xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
   xmlns:ns1="http://docs.active-
   endpoints.com/activebpel/sample/wsdl/order/2006/09/order.wsdl" xmlns:ns1-
   1="http://docs.active-
   endpoints.com/activebpel/sample/wsdl/pick_start/2006/09/pick_start.wsdl"
   xmlns:ns1-2="http://docs.active-
   endpoints.com/activebpel/sample/wsdl/pick_start/2006/09/pick_start.wsdl"
   xmlns:ns2="http://docs.active-
   endpoints.com/activebpel/sample/wsdl/orderProcess/2006/09/orderProcess.wsdl"
   xmlns:ns3="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:ns4="http://docs.active-
   endpoints.com/activebpel/sample/wsdl/pick_start/2006/09/pick_start.wsdl"
   xmlns:ns5="urn:oasis:names:specification:ubl:schema:xsd:Order-1.0"
   xmlns:ns6="urn:oasis:names:specification:ubl:schema:xsd:CommonBasicComponents-1.0"
   xmlns:ns7="urn:oasis:names:specification:ubl:schema:xsd:CommonAggregateComponents-
   1.0" xmlns:ns8="http://schemas.active-
   endpoints.com/sample/orderTypes/2006/09/orderTypes.xsd"
   xmlns:ns9="urn:oasis:names:specification:ubl:schema:xsd:OrderResponseSimple-1.0"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="multi-start_receives"
   suppressJoinFailure="yes" targetNamespace="http://docs.active-
   endpoints.com/activebpel/sample/bpel/multi-start_receives/2006/09/multi-
   start_receives.bpel">
   <import importType="http://schemas.xmlsoap.org/wsdl/"
   location="project:/BPEL_Samples/Resources/WSDL/order.wsdl"
   namespace="http://docs.active-
   endpoints.com/activebpel/sample/wsdl/order/2006/09/order.wsdl"/>
   <import importType="http://schemas.xmlsoap.org/wsdl/"
   location="project:/BPEL_Samples/Resources/WSDL/orderProcess.wsdl"
```

```
namespace="http://docs.active-
endpoints.com/activebpel/sample/wsdl/orderProcess/2006/09/orderProcess.wsdl"/>
<import importType="http://schemas.xmlsoap.org/wsdl/"
location="project:/BPEL_Samples/Resources/WSDL/pick_start.wsdl"
namespace="http://docs.active-
endpoints.com/activebpel/sample/wsdl/pick_start/2006/09/pick_start.wsdl"/>
<partnerLinks>
     <partnerLink myRole="orderProcess" name="orderProcessPLT"
     partnerLinkType="ns2:orderProcessPLT"/>
   <partnerLink myRole="pick-startProcess" name="ublOrderPLT"
   partnerLinkType="ns1-1:ublOrderPLT"/>
</partnerLinks>
<variables>
   <variable messageType="ns1:orderMessage" name="orderMessage"/>
   <variable messageType="ns2:orderProcessResponse" name="orderProcessResponse"/>
   <variable messageType="ns1-1:ublOrderMessage" name="ublOrderMessage"/>
   <variable messageType="ns1-1:customOutputMessage" name="customOutputMessage"/>
   <variable messageType="ns1-1:ublOutputMessage" name="ublOutputMessage"/>
</variables>
<correlationSets>
   <correlationSet name="Order" properties="ns2:PONum ns2:CustID"/>
</correlationSets>
<flow>
   <sequence>
     <receive createInstance="yes" operation="receiveOrder"
     partnerLink="orderProcessPLT" portType="ns2:OrderPT" variable="orderMessage">
          <correlations>
             <correlation initiate="join" set="Order"/>
          </correlations>
    </receive>
     <assign name="CreateResponse">
          <copy>
              <from>concat('Received custom order with PO# ',
             $orderMessage.order/OrderHeader/PONo, ' from customer ',
             $orderMessage.order/OrderHeader/CustId)</from>
             <to part="response" variable="orderProcessResponse"/>
          </copy>
          <copy>
             <from part="order" variable="orderMessage">
                <query>OrderHeader/PONo</query>
             </from>
             <to part="PONum" variable="orderProcessResponse"/>
          </copy>
          <copy>
             <from part="order" variable="orderMessage">
                <query>OrderHeader/CustId</query>
             </from>
             <to part="CustID" variable="orderProcessResponse"/>
          </copy>
   </assign>
     <reply operation="receiveOrder" partnerLink="orderProcessPLT"
     portType="ns2:OrderPT" variable="orderProcessResponse">
          <correlations>
             <correlation initiate="no" set="Order"/>
          </correlations>
    </reply>
   </sequence>
   <sequence>
     <receive createInstance="yes" operation="receiveUBLOrder"
     partnerLink="ublOrderPLT" portType="ns1-1:ublPT" variable="ublOrderMessage">
          <correlations>
             <correlation initiate="join" set="Order"/>
          </correlations>
      </receive>
      <assign name="InitializeVariable">
          <copy>
             <from>
                <literal>
    <OrderResponseSimple
     xmlns="urn:oasis:names:specification:ubl:schema:xsd:OrderResponseSimple-1.0"
     xmlns:cac="urn:oasis:names:specification:ubl:schema:xsd:CommonAggregateCompon
     ents-1.0"
     xmlns:cbc="urn:oasis:names:specification:ubl:schema:xsd:CommonBasicComponents
     -1.0"
     xmlns:ccp="urn:oasis:names:specification:ubl:schema:xsd:CoreComponentParamete
     rs-1.0"
```

```
        xmlns:dsc="urn:oasis:names:specification:ubl:schema:xsd:DocumentStatusCode-
        1.0"
        xmlns:sdt="urn:oasis:names:specification:ubl:schema:xsd:SpecializedDatatypes-
        1.0"
        xmlns:udt="urn:oasis:names:specification:ubl:schema:xsd:UnspecializedDatatype
        s-1.0">
                <ID>id</ID>
                <cbc:IssueDate/>
                <AcceptedIndicator>true</AcceptedIndicator>
                <cac:OrderReference/>
                <cac:BuyerParty/>
                <cac:SellerParty/>
        </OrderResponseSimple>
                   </literal>
                </from>
                <to part="output" variable="ublOutputMessage"/>
            </copy>
          </assign>
          <assign name="CreateResponse">
             <copy>
                 <from>concat('Received UBL order with PO# ',
                $ublOrderMessage.order/ns5:BuyersID, ' from customer ',
                $ublOrderMessage.order/ns7:BuyerParty/ns7:Party/ns7:PartyName/ns6:Name
                )</from>
                <to part="PONum" variable="orderProcessResponse"/>
             </copy>
             <copy>
                <from>'OK'</from>
                <to part="output" variable="customOutputMessage"/>
             </copy>
             <copy>
                <from part="order" variable="ublOrderMessage">
                   <query>ns6:IssueDate</query>
                </from>
                <to part="output" variable="ublOutputMessage">
                   <query>ns6:IssueDate</query>
                </to>
             </copy>
          </assign>
          <reply operation="receiveUBLOrder" partnerLink="ublOrderPLT" portType="ns1-
          1:ublPT" variable="ublOutputMessage"/>
       </sequence>
    </flow>
        </process>
```

Figure 13: Orchestration using BPEL

OWL-S does not distinguish the choreography and composite processes (or orchestration). In OWL-S, each choreography interface is realized as the composite process using only the operations of the one single service by defining the required sequence of service operations. Each of these composite processes can be treated as the choreography interface and can be used as the building block when composing more complex orchestrating processes.

On the other side, WSMO has the strong support for clearly distinguished choreography interfaces and orchestration work-flows. Generally, the process of complex process composition is the similar as in the case of OWL-S. The difference is that the composite processes are built as the combination of choreography blocks and the orchestration processes.

4.5.3 Static or dynamic orchestration

Another distinction is between static and dynamic service composition. In static service composition the services that are composed to form a higher level service are known in advance during development time and the sequence of executions can be pre-determined. On the other hand in dynamic composition one defines during development time the goal and the functionality that needs to be carried out and the discovery of matching services and the sequence of the service consumptions is determined dynamically during runtime.

In this Hydra iteration we have chosen to support only the static composition approach because the dynamic approach only increases the complexity and is of no real value in the Hydra context.

Dynamic composition can be added later on without any problems into the middleware. But this does not mean that services cannot be replaced by other services with the same functionality during runtime. Therefore it will be possible that the developer specifies e.g. that he needs a service for transmitting data with a certain bandwidth and TCP/IP support and an agent will then query the available repositories and provide a list of available services!

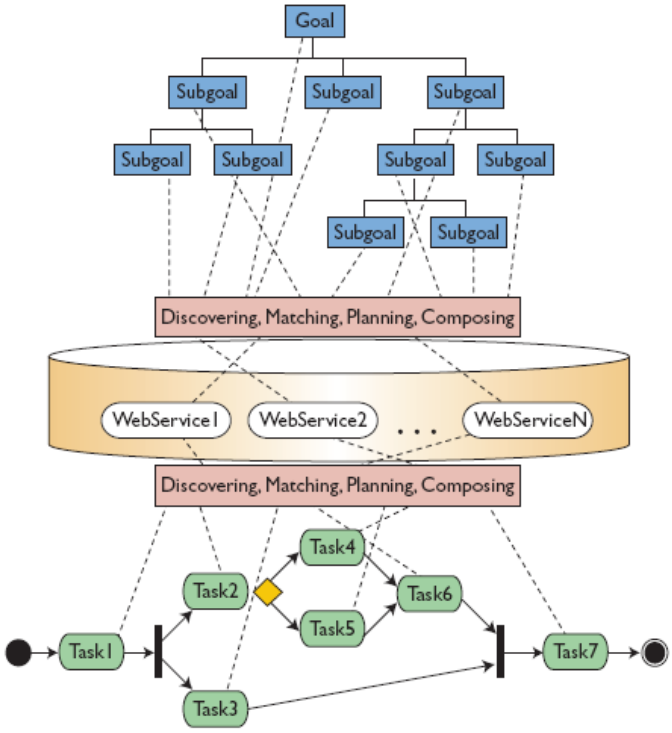

Figure 14: Orchestration example [8]

Another area that needs to be tackled is the transaction safety of executing a sequence of loosely coupled services. How does the middleware behave if a service is not available or delivers wrong data? How does one model how the application will react and will it be possible to rollback previously executed service invocations? For this there exists standards like WS-Transaction but it has to be seen if this is feasible within the Hydra context.

4.5.4 Hydra approach: Lightweight orchestration

As was said above in the first prototype of Hydra complex services are created by static service composition. Based on our experiments with existing orchestration approaches, as have been discussed above, we have concluded that they appear to be too complex and resource intensive to be used in Hydra. Therefore we will research if a more lightweight approach can be used. Such a custom orchestration language (Device Orchestration Language Light, DOLL), will be specified in the Hydra project, but it is not foreseen to be completed during this development iteration.

In the case of using the SAWSDL in Hydra, it would be suitable to create the models of services annotated from SAWSDL and use those processes as the building blocks for the composite work-flows. In all cases, if there is no need to take into account the complex processes requiring the on-line service discovery and planning, the composite processes can be specified in the development process and simply executed.

4.6 Ontology-driven Invocation of Services

4.6.1 Requirements

ID	Description	Rationale
111	Dynamic Web Service Binding	Middleware should be able to after device discovery and categorisation expose a new device as a web service that can be called without re-compilation.
129	Support for Semantic Web Standards for Device Communication	Middleware should support different semantic web standards, including OWL-S, WSMO, and selected parts of WS-*
164	Support for Service standards	Middleware should support widely used standards for service description, discovery, orchestration and execution.
180	Service mediating network connections according to different qualities	There should be a service which lists different network connections depending on specified properties (connection speed, encryption). Devices can then negotiate such connections with remote devices, without the need to take care about the networking details
197	Services define their communication needs in terms of needed QoS parameters	The services define their communication needs in terms of needed QoS parameters (needed bandwidth, needed quality...) without specifying the technical details. The middleware is free to choose the appropriate networking matching the specified needs

Table 5: Requirements on Invocation and Execution

4.6.2 Service Grounding

Semantic Web Services frameworks like OWL-S and WSMO combine semantic descriptions of Web service capabilities, inputs, outputs and behavior with the syntactic interface descriptions in WSDL and XML Schema. Services are invoked using the grounding model, which specifies how to communicate with the particular service. For the interoperability reasons with existing Web Services and infrastructures, both OWL-S and WSMO support the service grounding into WSDL. According to the grounding model the service is invoked using specified operations with related inputs, outputs and endpoints.

In the past years some approaches to service grounding have been developed. While each of them has quite different characteristics they share the extension of WSDL with semantic information. Below we discuss how service grounding is done in OWL-S, WSMO and SAWSDL.

4.6.2.1 OWL-S

Below is an example of a service grounding using OWL-S.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
xml:base="http://hydra.cnet.se/SMSService/SMSService_Grounding.owl#"
xmlns:grounding="http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/owls11/Grounding.owl#"
xmlns:owl="http://jamsci.servehttp.com/owlsedit/owl.rdf#"
xmlns:process="http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/owls11/Process.owl#"
xmlns:profile="http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/owls11/Profile.owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://jamsci.servehttp.com/owlsedit/rdf-schema.rdf#"
xmlns:service="http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/owls11/Service.owl#" xmlns:xsd="http://jamsci.servehttp.com/owlsedit/XMLSchema.xsd#">
```

```
<owl:Ontology rdf:about="">
        <owl:versionInfo>Version 1.0</owl:versionInfo>
        <rdfs:comment>Add Ontology Comment</rdfs:comment>
        <owl:imports rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns"/>
        <owl:imports rdf:resource="http://jamsci.servehttp.com/owlsedit/owl.rdf"/>
        <owl:imports rdf:resource="http://jamsci.servehttp.com/owlsedit/rdf-schema.rdf"/>
        <owl:imports
        rdf:resource="http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/owls
        11/Service.owl"/>
        <owl:imports
        rdf:resource="http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/owls
        11/Profile.owl"/>
        <owl:imports
        rdf:resource="http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/owls
        11/Process.owl"/>
        <owl:imports
        rdf:resource="http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/owls
        11/Grounding.owl"/>
</owl:Ontology>
<grounding:WsdlGrounding rdf:ID="SMSService_Grounding">
        <service:supportedBy
        rdf:resource="http://hydra.cnet.se/SMSService/SMSService_Service#SMSService_Service"/
        >
        <grounding:hasAtomicProcessGrounding
        rdf:resource="#WSDLGrounding_SMSService_SendSMS"/>
        <grounding:hasAtomicProcessGrounding
        rdf:resource="#WSDLGrounding_SMSService_CheckNumber"/>
        <grounding:hasAtomicProcessGrounding
        rdf:resource="#WSDLGrounding_SMSService_NotifyError"/>
</grounding:WsdlGrounding>
<grounding:WsdlAtomicProcessGrounding rdf:ID="WSDLGrounding_SMSService_SendSMS">
        <grounding:owlsProcess
        rdf:resource="http://hydra.cnet.se/SMSService/SMSService_ProcessModel#SMSPortType_Sen
        dSMS"/>
        <grounding:wsdlOperation>
                <xsd:uriReference rdf:value="http://Hydra.cnet.se/SMSService.wsdl#SendSMS"/>
        </grounding:wsdlOperation>
        <grounding:wsdlInputMessage>
                <xsd:uriReference
        rdf:value="http://Hydra.cnet.se/SMSService.wsdl#SendSMSSoapIn"/>
        </grounding:wsdlInputMessage>
        <grounding:wsdlInputMessageParts rdf:parseType="Collection">
                <grounding:WsdlMessageMap>
                        <grounding:owlsParameter
                        rdf:resource="http://hydra.cnet.se/SMSService/SMSService_ProcessModel#
                        SMSPortType_SendSMS_number_IN"/>
                        <grounding:wsdlMessagePart>
                                <xsd:uriReference
                        rdf:value="http://Hydra.cnet.se/SMSService.wsdl#number"/>
                        </grounding:wsdlMessagePart>
                </grounding:WsdlMessageMap>
                <grounding:WsdlMessageMap>
                        <grounding:owlsParameter
                        rdf:resource="http://hydra.cnet.se/SMSService/SMSService_ProcessModel#
                        SMSPortType_SendSMS_message_IN"/>
                        <grounding:wsdlMessagePart>
                                <xsd:uriReference
                        rdf:value="http://Hydra.cnet.se/SMSService.wsdl#message"/>
                        </grounding:wsdlMessagePart>
                </grounding:WsdlMessageMap>
        </grounding:wsdlInputMessageParts>
        <grounding:wsdlOutputMessage>
                <xsd:uriReference
        rdf:value="http://Hydra.cnet.se/SMSService.wsdl#SendSMSSoapOut"/>
        </grounding:wsdlOutputMessage>
        <grounding:wsdlOutputMessageParts rdf:parseType="Collection">
                <grounding:WsdlMessageMap>
                        <grounding:owlsParameter
                        rdf:resource="http://hydra.cnet.se/SMSService/SMSService_ProcessModel#
                        SMSPortType_SendSMS_SendSMSResult_OUT"/>
                <grounding:wsdlMessagePart>
                        <xsd:uriReference
                rdf:value="http://Hydra.cnet.se/SMSService.wsdl#SendSMSResult"/>
                </grounding:wsdlMessagePart>
                </grounding:WsdlMessageMap>
        </grounding:wsdlOutputMessageParts>
```

```
        <grounding:wsdlReference>
                <xsd:uriReference rdf:value="http://www.w3.org/TR/2001/NOTE-wsdl-20010315"/>
        </grounding:wsdlReference>
</grounding:WsdlAtomicProcessGrounding>
<grounding:WsdlAtomicProcessGrounding rdf:ID="WSDLGrounding_SMSService_CheckNumber">
        <grounding:owlsProcess
        rdf:resource="http://hydra.cnet.se/SMSService/SMSService_ProcessModel#SMSPortType_Che
        ckNumber"/>
        <grounding:wsdlOperation>
                <xsd:uriReference
        rdf:value="http://Hydra.cnet.se/SMSService.wsdl#CheckNumber"/>
        </grounding:wsdlOperation>
        <grounding:wsdlInputMessage>
                <xsd:uriReference
        rdf:value="http://Hydra.cnet.se/SMSService.wsdl#CheckNumberSoapIn"/>
        </grounding:wsdlInputMessage>
        <grounding:wsdlInputMessageParts rdf:parseType="Collection">
                <grounding:WsdlMessageMap>
                        <grounding:owlsParameter
                        rdf:resource="http://hydra.cnet.se/SMSService/SMSService_ProcessModel#
                        SMSPortType_CheckNumber_number_IN"/>
                <grounding:wsdlMessagePart>
                        <xsd:uriReference
                rdf:value="http://Hydra.cnet.se/SMSService.wsdl#number"/>
                </grounding:wsdlMessagePart>
                </grounding:WsdlMessageMap>
        </grounding:wsdlInputMessageParts>
        <grounding:wsdlOutputMessage>
                <xsd:uriReference
                rdf:value="http://Hydra.cnet.se/SMSService.wsdl#CheckNumberSoapOut"/>
        </grounding:wsdlOutputMessage>
        <grounding:wsdlOutputMessageParts rdf:parseType="Collection">
                <grounding:WsdlMessageMap>
                        <grounding:owlsParameter
                        rdf:resource="http://hydra.cnet.se/SMSService/SMSService_ProcessModel#
                        SMSPortType_CheckNumber_CheckNumberResult_OUT"/>
                        <grounding:wsdlMessagePart>
                                <xsd:uriReference
                        rdf:value="http://Hydra.cnet.se/SMSService.wsdl#CheckNumberResult"/>
                        </grounding:wsdlMessagePart>
                </grounding:WsdlMessageMap>
        </grounding:wsdlOutputMessageParts>
        <grounding:wsdlReference>
                <xsd:uriReference rdf:value="http://www.w3.org/TR/2001/NOTE-wsdl-20010315"/>
        </grounding:wsdlReference>
</grounding:WsdlAtomicProcessGrounding>
<grounding:WsdlAtomicProcessGrounding rdf:ID="WSDLGrounding_SMSService_NotifyError">
        <grounding:owlsProcess
        rdf:resource="http://hydra.cnet.se/SMSService/SMSService_ProcessModel#SMSPortType_Not
        ifyError"/>
        <grounding:wsdlOperation>
                <xsd:uriReference
                rdf:value="http://Hydra.cnet.se/SMSService.wsdl#NotifyError"/>
        </grounding:wsdlOperation>
        <grounding:wsdlInputMessage>
                <xsd:uriReference
                rdf:value="http://Hydra.cnet.se/SMSService.wsdl#NotifyErrorSoapIn"/>
        </grounding:wsdlInputMessage>
        <grounding:wsdlInputMessageParts rdf:parseType="Collection">
                <grounding:WsdlMessageMap>
                        <grounding:owlsParameter
                        rdf:resource="http://hydra.cnet.se/SMSService/SMSService_ProcessModel#
                        SMSPortType_NotifyError_number_IN"/>
                        <grounding:wsdlMessagePart>
                                <xsd:uriReference
                        rdf:value="http://Hydra.cnet.se/SMSService.wsdl#number"/>
                        </grounding:wsdlMessagePart>
                </grounding:WsdlMessageMap>
                <grounding:WsdlMessageMap>
                        <grounding:owlsParameter
                        rdf:resource="http://hydra.cnet.se/SMSService/SMSService_ProcessModel#
                        SMSPortType_NotifyError_message_IN"/>
                        <grounding:wsdlMessagePart>
                                <xsd:uriReference
                                rdf:value="http://Hydra.cnet.se/SMSService.wsdl#message"/>
                        </grounding:wsdlMessagePart>
```

```
                </grounding:WsdlMessageMap>
            </grounding:wsdlInputMessageParts>
            <grounding:wsdlOutputMessage>
                <xsd:uriReference
                rdf:value="http://Hydra.cnet.se/SMSService.wsdl#NotifyErrorSoapOut"/>
            </grounding:wsdlOutputMessage>
            <grounding:wsdlOutputMessageParts rdf:parseType="Collection">
                <grounding:WsdlMessageMap>
                    <grounding:owlsParameter
                    rdf:resource="http://hydra.cnet.se/SMSService/SMSService_ProcessModel#
                    SMSPortType_NotifyError_NotifyErrorResult_OUT"/>
                    <grounding:wsdlMessagePart>
                        <xsd:uriReference
                        rdf:value="http://Hydra.cnet.se/SMSService.wsdl#NotifyErrorRes
                        ult"/>
                    </grounding:wsdlMessagePart>
                </grounding:WsdlMessageMap>
            </grounding:wsdlOutputMessageParts>
            <grounding:wsdlReference>
                <xsd:uriReference rdf:value="http://www.w3.org/TR/2001/NOTE-wsdl-20010315"/>
            </grounding:wsdlReference>
        </grounding:WsdlAtomicProcessGrounding>
</rdf:RDF>
```

Figure 15: Service grounding using OWL-S

4.6.2.2 SAWSDL

WSDL provides several hooks into extending the description with semantic elements. One approach can be to include the semantic elements directly in the WSDL file and the other approach inserts only a link to an external file which contains the semantic description.

The newest member of this group is SAWSDL which is primarily based on WSDL-S. SAWSDL is agnostic to the underlying semantic web framework therefore it could be used with OWL-S, WSMO or even a non-ontology language like UML.

```
<?xml version="1.0" ?>
<definitions name="SMSService"
targetNamespace="http://Hydra.cnet.se/SMSService.wsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://Hydra.cnet.se/SMSService.wsdl"
xmlns:sawsdl="http://www.w3.org/ns/sawsdl">
  <message name="SendSMSSoapIn">
        <part name="number" type="xsd:string"
        sawsdl:modelReference="http://hydra.cnet.se/serviceOntology/SMSService#Teleph
        oneNumber"
        sawsdl:liftingSchemaMapping="http://hydra.cnet.se/serviceOntology/SMSService/
        StringToTelephoneNumber.lifting.xslt"
        sawsdl:loweringSchemaMapping="http://hydra.cnet.se/serviceOntology/SMSService
        #TelephoneNumberToString.lowering"/>
        <part name="message" type="xsd:string" />
  </message>
  <message name="SendSMSSoapOut">
        <part name="SendSMSResult" type="xsd:boolean"
        sawsdl:modelReference="http://hydra.cnet.se/serviceOntology/SMSService#Operat
        ionSuccessful"/>
  </message>
  <message name="CheckNumberSoapIn">
        <part name="number" type="xsd:string"
        sawsdl:modelReference="http://hydra.cnet.se/serviceOntology/SMSService#Teleph
        oneNumber"
        sawsdl:liftingSchemaMapping="http://hydra.cnet.se/serviceOntology/SMSService/
        StringToTelephoneNumber.lifting.xslt"
        sawsdl:loweringSchemaMapping="http://hydra.cnet.se/serviceOntology/SMSService
        #TelephoneNumberToString.lowering"/>
  </message>
  <message name="CheckNumberSoapOut">
        <part name="CheckNumberResult" type="xsd:boolean"
        sawsdl:modelReference="http://hydra.cnet.se/serviceOntology/SMSService#Operat
        ionSuccessful"/>
  </message>
  <message name="NotifyErrorSoapIn">
```

```
            <part name="number" type="xsd:string"
            sawsdl:modelReference="http://hydra.cnet.se/serviceOntology/SMSService#Teleph
            oneNumber"
            sawsdl:liftingSchemaMapping="http://hydra.cnet.se/serviceOntology/SMSService/
            StringToTelephoneNumber.lifting.xslt"
            sawsdl:loweringSchemaMapping="http://hydra.cnet.se/serviceOntology/SMSService
            #TelephoneNumberToString.lowering"/>
            <part name="message" type="xsd:string" />
      </message>
      <message name="NotifyErrorSoapOut">
        <part name="NotifyErrorResult" type="xsd:boolean" />
      </message>
    <portType name="SMSPortType"
    sawsdl:modelReference="http://hydra.cnet.se/serviceOntology/SMSService">
        <operation name="SendSMS">
          <sawsdl:attrExtensions
            sawsdl:modelReference="http://hydra.cnet.se/serviceOntology/SMSService#SendSM
            S"/>
          <input message="tns:SendSMSSoapIn" name="SendSMS" />
          <output message="tns:SendSMSSoapOut" name="SendSMSResponse" />
        </operation>
        <operation name="CheckNumber">
                  <sawsdl:attrExtensions
                  sawsdl:modelReference="http://hydra.cnet.se/serviceOntology/SMSService
                  #ValidateTelephoneNumber"/>
          <input message="tns:CheckNumberSoapIn" name="CheckNumber" />
           <output message="tns:CheckNumberSoapOut" name="CheckNumberResponse" />
        </operation>
        <operation name="NotifyError">
          <sawsdl:attrExtensions
            sawsdl:modelReference="http://hydra.cnet.se/serviceOntology/SMSService#Notify
            Error"/>
          <input message="tns:NotifyErrorSoapIn" name="NotifyError" />
          <output message="tns:NotifyErrorSoapOut" name="NotifyErrorResponse" />
        </operation>
      </portType>
      <binding name="SMSSoapBinding" type="tns:SMSPortType">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
        <operation name="SendSMS" >
          <soap:operation soapAction="" />
          <input>
            <soap:body use="encoded" namespace="urn:SMS-SoapServices"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
          </input>
          <output>
            <soap:body use="encoded" namespace="urn:SMS-SoapServices"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
          </output>
        </operation>
        <operation name="CheckNumber">
          <soap:operation soapAction="" />
          <input>
            <soap:body use="encoded" namespace="urn:SMS-SoapServices"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
          </input>
          <output>
            <soap:body use="encoded" namespace="urn:SMS-SoapServices"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
          </output>
        </operation>
        <operation name="NotifyError">
          <soap:operation soapAction="" />
          <input>
            <soap:body use="encoded" namespace="urn:SMS-SoapServices"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
          </input>
          <output>
            <soap:body use="encoded" namespace="urn:SMS-SoapServices"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
          </output>
        </operation>
      </binding>
      <service name="SMSSoapService">
        <documentation></documentation>
        <port name="SMSSoapPort" binding="tns:SMSSoapBinding">
          <soap:address
```

```
        location="http://hydra.cnet.se/SMSService/SmsService.asmx"/>
      </port>
    </service>
  </definitions>
```

Figure 16: Service Grounding using SAWSDL

SAWSDL is seen as very flexible because it is domain independent and independent on the mapping language and this flexibility is needed in a business context where one cannot expect to have uniform representations of data and interfaces between e.g. two companies. This flexibility raises also some criticism that additional conventions and guidelines need to be set up to ensure a proper execution of the supported operations. SAWSDL also is no fully fledged SWS framework in that it provides no sophisticated support for discovery and composition but only supports basic service discovery [25].

4.6.3 Data grounding

An important issue related to service grounding and invocation is the data grounding, which specifies how complex WSDL types should be transformed to ontology concepts and vice versa. This task often requires data mediation in cases when the WSDL complex type does not match the related ontology concept perfectly.

OWL-S usually uses the XSLT transformations as the solution to data mediation. Conceptual model of WSMO contains description of mediators, which are used also for purposes of data mediation between WSDL types and ontology concepts.

When using the SAWSDL approach, the Hydra service ontology would have to model the grounding information, which can be inspired by the OWL-S or WSMO approach. Data mediation can be realized by extending the ontology model of inputs and outputs according to *liftingSchemaMapping* and *loweringSchemaMapping* SAWSDL attributes.

4.6.4 Hydra approach

The Hydra approach to service grounding and invocation is based on the combination of WSDL semantic annotation with grounding references in the ontology. The device ontology holds descriptions of devices and services. The discovery, mapping and reasoning done to find suitable services to accomplish a certain task is performed using the ontology. The WSDL grounding for the semantic service is referenced in the ontology, so that the web service on any device that can be found in the ontology also can be called without using information from the device (although the device could provide its own WSDL).

4.6.4.1 SAWSDL grounding

SAWSDL provides a big advantage for the later use of Hydra especially in the business context and therefore we have chosen to use SAWSDL as the basis for the SWS grounding. This decision was strongly influenced by requirements #129 (Support for different SWS frameworks) and #164 (Support of widely used standards).

Regarding the annotation of WSDL files with semantic information SAWSDL only supports inline annotations. Therefore linking to an external file for the semantic description is not possible. But since WSDL files are usually automatically created using specific tools this is not a disadvantage but more an advantage. This approach allows us to define additional guidelines and conventions in models that can be checked during the automatic generation of the WSDL files. Those referenced models can also be used during development to provide the developer with hints on how to adhere to these conventions and therefore lower the defect rate and increase compatibility with other Hydra based services.

Another problem with SAWSDL is the need to map between the ontologies used and the XML data that is used for the SAWSDL description. This problem is also augmented by our automatic MDA approach to generating the WSDL files.

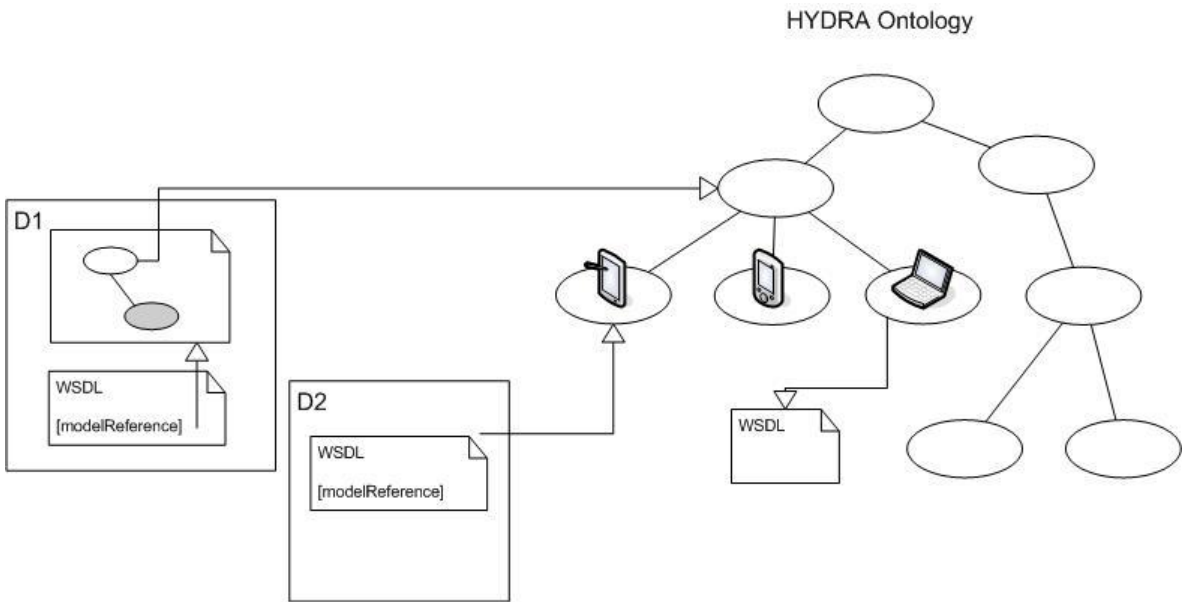

Figure 17: The Hydra approach to service grounding is based on SAWSDL with grounding references in the ontology

However, it may also be the case that a device manufacturer provides an "embedded" WDSL with semantic annotations using concepts from the Hydra Device Ontology, e.g. "D1 is an instance of a subclass of Device class X". When Hydra discovers this new Hydra compliant device, it can automatically incorporate this information in the Device Ontology and infer the capabilities of the device. Semantic annotations of web services in Hydra may thus be performed in both a centralized and a decentralized manner. For the latter decentralized approach we will use SAWSDL to semantically annotate the service descriptions.

4.6.4.2 UPnP grounding

We also allow service grounding directly using UPnP by extending the SCPD format to allow direct annotation on the device. This allows a device manufacturer to either provide a link to the Hydra Device Ontology for each action a device can perform or provide a device type identifier that can be looked up in the Hydra Device Ontology.

```
<?xml version="1.0" encoding="utf-8"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
   <specVersion>
      <major>1</major>
      <minor>0</minor>
   </specVersion>
   <actionList>
      <action hydraannotation="http://www.hydra.cnet.se">
         <name>GetStatus</name>
         <argumentList>
            <argument>
               <name>ResultStatus</name>
               <direction>out</direction>
               <relatedStateVariable>Status</relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action hydraannotation=" http://www.hydra.cnet.se">
         <name>SetTarget</name>
         <argumentList>
            <argument>
               <name>newTargetValue</name>
               <direction>in</direction>
               <relatedStateVariable>Target</relatedStateVariable>
```

```
            </argument>
         </argumentList>
      </action>
   </actionList>
   <serviceStateTable>
      <stateVariable sendEvents="yes">
         <name>Status</name>
         <dataType>boolean</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>Target</name>
         <dataType>boolean</dataType>
      </stateVariable>
   </serviceStateTable>
</scpd>
```

Figure 18: UPnP Service Grounding by annotating SCPD

4.7 Secure Semantic Web Services for Devices

4.7.1 Requirements

ID	Description	Rationale
129	Support for Semantic Web Standards for Device Communication	Middleware should support different semantic web standards, including OWL-S, WSMO, and selected parts of WS-*
164	Support for Service standards	Middleware should support widely used standards for service description, discovery, orchestration and execution.
239	Automatic service diagnostic for security relevant services	Security relevant services should provide a self-diagnostic services that provides an overview of all security-relevant features
358	Developer must be able to semantically define security requirements	If developers are to make devices that can co-operate through other protocols and security mechanisms, they have to be able to describe the inherent security requirements in a semantic interoperable language. It is not enough just to use a specific protocol's security as this does NOT tell WHY he uses it and WHAT he really needs for the application to proceed.

Table 6: Requirements on Security of Web Services for Devices

Security is an important issue when it comes to web services - whether semantic or not. Although web services are re-usable and accessible components by design, not every service is thought to be used by everybody. Access to a service may rather depend on the identity of the requester, of certain attributes or even of the current context. As web services themselves do not support any kind of access control, such mechanisms have to be provided in addition to Hydra's services. Even other security requirements like confidentiality, non-repudiation, integrity or authenticity are not an integral part of web services and have to be specified by additional mechanisms. Although there are a number of different mechanisms which could be used to secure communication between Hydra's managers, web service specific standards like WS-Policy and WS-Security will probably be used for core Hydra security. A discussion of possible solutions is found in deliverable D5.6 and will be amended by the contributions in D7.3.

Semantic web services provide some benefits supporting security. It is for instance possible to use semantic descriptions attached to web services in order to describe their security requirements and capabilities. Security requirements describe conditions set by the service which have to be met by the requester. For example a service could make the condition that every call has to be signed by a trusted third party. Security capabilities on the other hand describe functionality supported by a service, e.g. the ability to apply PGP encryption to data. However adding semantic descriptions to web services also introduces some new threats which do not exist in traditional web services. A new

attack vector is for instance the manipulation of security-relevant semantic annotations of a service: An attacker could make a service pretend to support strong security mechanisms which are in fact not available and thereby trick a requestor into assuming a higher protection level. The result could be that requesters choose the attacked service for critical operations as it appears to provide the best security mechanisms. However the service might in fact provide no security mechanisms at all and thereby leave all critical data prone to manipulations by the attacker.

In this section, we will outline some benefits for Hydra we expect from using semantic web services for security reasons. Further, we will give an overview of existing approaches in the area and discuss if and how they could be adopted to the Hydra architecture. Finally we will outline some challenges raising from the combination of security and semantic web services.

4.7.2 Benefits

In this subsection, we sketch how semantic web services could contribute to Hydra's security concept.

4.7.2.1 Using semantic annotations to express security requirements

Semantic web services differ from syntactic web services in semantic annotations that are attached to the service' description and allow for automatic discovery, composition and execution of services depending on their properties. These descriptions could also be used to express a service's security requirements. These could either be contained in the semantic description itself, e.g. in OWL-S or WSMO or could be described by external policies which are referenced from the semantic description of a service. The latter allows already existing security policies to be referenced from a service and thereby broadens the range of possible policy languages to be used. While the description of security requirements of traditional web services is limited to WS-specific languages like WS-Policy (and its extension WS-SecurityPolicy), semantic annotations can refer to any kind of external policy definition and thereby allow using potentially more appropriate policy languages. In case the security requirements are directly specified in the semantic service description, they could be referenced from external policies and be used in a rule's condition. An example would be using XACML policies to restrict the access to services, depending on properties of the service which are declared in its semantic annotation.

However, both possibilities require adapting policy enforcement mechanisms in order to take semantic descriptions into account. Just attaching security specifications to a service does not guarantee that callers obey them. Instead a policy enforcement point[4] (PEP) must ensure that all calls to a service are first checked against the security requirements specified by the service's semantic annotations. It is clear that these annotations must be understood by all PEPs. The same applies to the policy language used. Semantic descriptions cannot ensure interoperability if they refer to policy specifications which are not understood by all parties. Thus, a common policy representation has to be used and semantic descriptions of security properties have to refer to instances of a common security ontology.

4.7.2.2 Discovery according to security requirements

Enriching Hydra services by semantic descriptions of security capabilities and requirements can not only be used for enforcement during invocation of the services.

A further possibility is to use semantic security specifications even for discovering and selecting appropriate services at run time. As the security requirements of a requester are known in advance, it is possible to select only such services which match these requirements. Thus a requester can avoid orchestrating services which can not be used at a later point due to mismatching security

[4] A PEP is responsible for enforcing a security policy, i.e. for ensuring that it becomes effective. The PEP is an abstract concept and thus does not have to be a separate component; it can also be integrated into one or more of the existing services.

requirements. This helps reducing errors during runtime as well as it can reduce communication overhead.

As a service appears as a black box to the requester, semantic annotations can provide more information about how data is handled and thereby go beyond what can be described using traditional security policies which in most cases are limited to the communication between requester and service. For example a requester could require services not to store any data received by the request. In case a service uses semantic annotations to declare that it will fulfil this requirement, the requester might prefer this service to another for orchestration. A further advantage would be that it becomes possible to predict in advance the level of security to which all orchestrated services will be able comply.

4.7.2.3 Security as a wrapper to services

Semantic descriptions separate the description of security requirements and capabilities of a service from the actual implementation. "Security as a wrapper" thus means that the behaviour of a service with respect to security can be changed without modifying the service itself. It therefore eases modelling security aspects of a whole system at once and makes modifications of a component's code superfluous. For example in case a service should be used in a more critical environment where higher security requirements apply, the service's security requirements can be increased although its implementation might not be accessible. However "security as a wrapper" must not be taken as a replacement for careful implementation of security features, it is rather thought to be an additional concept to help decoupling the security possibilities (the implementation) of a system from its behaviour (described by policies).

4.7.2.4 Context-dependent security requirements

By providing machine-processable descriptions, semantic web services can provide information tailored to a user's requirements and needs. The user's context therefore plays a major role as it enables automatic reasoning about the requirements a user might have in a certain situation and the capabilities provided by a service. This applies not only to requirements which will result in a more personalised behaviour of the service but also to security requirements which may strongly depend on the current context, e.g. on properties of a device, on the current location of the user or on the current time.

As an example, imagine a service providing guidance in a public building. The information provided by the service depends on the user's current location as well as her expertise. In addition, access to the service should be limited to legitimate users which are inside the building. Thus information about the user's context (e.g. her location) is part of the access-control policy of the service. In this example, the service could provide semantic annotations stating that location data must be provided to the PEP in order to validate whether a user is allowed to use the service or not. What exactly "location" means could be derived from an ontology, resulting in different possibilities like GPS, GSM-localisation or a Bluetooth-localisation.

Thus, semantic descriptions of services will – in combination with a semantic-enabled policy language – allow expressing context-dependent security requirements which can be enforced at runtime.

4.7.2.5 Negotiation between requester and service

In order to establish a connection between a requester and a service, both have to agree on a common security configuration, including algorithms used for encryption and hashing, key lengths, authentication procedures, different parameters, etc. Thus, both communication partner need to have a common understanding of the syntax they use to express these mechanisms. However as Hydra will use dynamically loadable modules for security implementations, the specific security configuration will usually not be known in advance. Developers should instead focus on the protection goals they want to apply to a connection and then leave it to the middleware to find appropriate implementation modules at runtime. Hydra should then automatically select implementations which support the desired protection goal and are interoperable with the

communication partner's implementation. Semantic definitions of security requirements are therefore predestined as they allow applying reasoning techniques on the knowledge base consisting of device ontologies, security requirements and capabilities of a service and the security requirements defined by the developer. This way, modules which fulfil these various constraints can automatically be found and applied while developers can specify security requirements by their meaning instead of their implementations.

4.7.3 Possible Approaches and Related Work

There are a number of different approaches on integrating security aspects into semantic web services. In this section we will discuss the most important of them and explain how they could contribute to secure semantic web services in Hydra.

4.7.3.1 Security descriptions for services

In [11] Kagal et al. propose to use OWL-S annotations attached to web services in order to reference external security policies. These policies describe requirements as well as capabilities of a service and can thus be used for negotiating a common level of security between requester and service.
This approach could also be applied to Hydra – it addresses the issue of negotiating security requirements coming from different domains. An example would be a user joining an unknown Hydra environment with her Hydra-enabled device. In this case, the services in the environment have different security requirements and capabilities than the user. The description of these requirements could be attached to the environment's and the user's services, using OWL-S. In order to negotiate a common security level, the concepts[5] used by the Hydra environment and by the user should be contained in a common ontology to facilitate the negotiation process. Thus, a common Hydra security ontology, describing concepts for requirements and capabilities of a service is recommended[6].

4.7.3.2 Semantic policies

In general, the approach of attaching security descriptions to services does not require any special policy language. Kagal et al. however propose for expressing security policies the language "Rei" [10] which stands out of the plethora of policy languages by the fact that the concepts which make up the language itself are represented in ontologies. The idea of using description logic (DL) (e.g. ontologies) for policy languages has been investigated by several authors and begins to gain credence. Up to now, besides KAoS [28], Rei seems to be the only policy language which is based on DL (A comparison of Rei and KAoS can be found in [27]). In addition there are a number of approaches focussing at translating traditional policy languages to knowledge-based representations (like OWL-DL) [13] [14] [15]. Although representing the complete policy language as a knowledge base has the advantage that conventional reasoners can be used as policy decision and -analysing tools, this might not always be required. For the development of Hydra, it must further be taken into account that reasoning over DL is much less efficient than evaluation algorithms tailored to a specific policy language. More pragmatic approaches like [2] try not to map policies to description logics but to add semantic descriptions to traditional policy languages. This seems to be a more promising approach for Hydra as it allows using existent, approved and efficient policy evaluation algorithms and at the same time provides reasoning possibilities in case rule conditions are based on semantic descriptions. However, more detailed decisions on the choice of policy languages will be made during the policy manager design process.

[5] *Concept* here is meant in terms of description logic and refers to a *class* in OWL

[6] If user and environment used different ontologies, a mapping between these two would be required. As in general it is not possible to create a mapping automatically, both have to agree on a common ontology in advance.

4.7.3.3 Discovering secure services

Regarding the discovery of services according to their security properties, Kagal et al. propose using a "matchmaker" engine in order to look for services fulfilling certain capabilities (e.g. "is able to encrypt using PGP"). Another approach for semantic discovery is described in [12]. The authors present the discovery scheme "UbiSearch" which is aims at finding services according to their semantic descriptions in large-scale ubiquitous computing architectures. Therefore, services are structured according to a "semantic distance", meaning the neighbourhood of services which have similar descriptions. UbiSearch uses an overlay network for resolving queries and thus is suited for architectures with a large number of different services. While the notion of a semantic distance appears sensible, the UbiSearch-approach seems to be too complex and too demanding for an embedded systems middleware like Hydra – especially in terms of communication overhead. Instead, for the first try a semantic discovery mechanism based on broadcasts or centralised registries should work out for Hydra. A good example for such a mechanism is the "Semantic MatchMaker" engine described in [5]. The Semantic MatchMaker is a web service to which other web services can register announcing their OWL-S descriptions. The Semantic MatchMaker's interface then accepts queries for capabilities and returns services matching these capabilities[7].

4.7.3.4 Identity Management

While coalescing security requirements and capabilities (i.e. policies) with semantic descriptions probably provides the most benefit to Hydra as a middleware, there are other possibilities to leverage semantic web services for security aspects: In [4], Chowdhury et al. propose using ontologies for identity management. These semantically represented identities are used in [22] in order to realise secure access to semantic web services. Hydra could provide similar mechanisms for representing identities as an extension to a semantic access control mechanism. To what extend issues like identity management will be provided by the core Hydra middleware itself is however not clear yet and has to be further investigated in deliverable D7.3, chapter 7.

4.8 Automatic Generation of SWS proxies for Devices

To enable the Hydra middleware and consequently the Hydra developer to view and use all devices and services in a heterogeneous fashion, automatic generation of semantic web service proxies will be necessary for the non-Hydra enabled devices. When a non-Hydra device is discovered and identified, a proxy for this device is generated using the information in the device and service ontologies.

The service proxy will expose a web service interface that is semantically enriched using SAWSDL annotations referring to concepts in the service and device ontologies from which it was generated. This way, all devices are represented at the level of semantic device as described in D6.2 and are heterogeneous to the middleware and to developers using the SDK. For developers, some additional support for domain concepts can also be generated by the SDK. At design time, this may include automatic code generation to support the concepts that are inputs and outputs to the service. The lowering and lifting schema mappings can be used to handle this transparently to the Hydra developer, so that the concepts from the ontologies are first-class entities when composing applications in the development environment. All Hydra devices that a developer uses in composition will be semantic devices, where there will at least be a reference in the WSDL to the corresponding service in the service ontology. This will also be useful if Hydra is to support dynamic service composition, as the services will already be semantically enriched.

[7] A web interface of the Semantic MatchMaker can be tried out at http://www.daml.ri.cmu.edu/matchmaker

5. Concluding Remarks and Future Work

In this final chapter we briefly describe our plans for future work and summarise our conclusions.

5.1 Future Work

5.1.1 Performance issues

Because Hydra is supposed to create a middleware for embedded devices we have to tackle the following research questions:

- How do we enable efficient inclusion of embedded devices into semantic web services if those devices are not able to run semantic web services on their own and how does this impact the system architecture and the development of Hydra based solutions?
- Can semantic web services be brought to embedded devices and what are the minimum requirements and also what implications does this have on resource consumption?

5.1.2 Semantic Discovery of Networked Devices and Services

There are several issues to be further investigated for the management of the DAC and the discovery process. In this (2nd) iteration we have decided that all service composition will occur at design time. In the following iterations, the Hydra middleware may have to resolve at run time when a set of devices and services that are present in the network constitute a composite device, and place this composite device in the Hydra Device Application Catalogue.

The Hydra discovery functions will be able to discover other devices that use a number of different protocols; Bluetooth, UPnP, Zigbee etc. These may also be able to announce themselves to other devices using all these protocols. However, not all Hydra devices will be capable of this. The more limited devices will be able to handle web services (in order to be Hydra devices), and these may also need some way of announcing themselves on the network.

This work will be further pursued in workpackage 5, task T5.3 "Wireless Devices", while in workpackage 6 we will focus on the semantic discovery aspects.

5.1.3 Lightweight Orchestration of Device Services

In the future work, it is possible to take into account the orchestration using real-time service discovery and planning. The services in the orchestration work-flow can be defined in the terms of required goals instead of concretely specified services. According to specifications of standards, OWL-S and WSMO support this approach. OWL-S defines simple processes and WSMO uses so-called Goals to represent the required ideal services. In the real applications, this approach is often avoided by pre-computing of defined process sequences. Similarly, as in the case of service discovery, real-time reasoning with service IOPEs, is really much time consuming, when searching for suitable service in the work-flow.

5.1.4 Secure Semantic Web Services for Devices

There are still a number of challenges regarding security in combination with semantic web services. In this section, we will outline some of them and discuss how they could influence the hydra security architecture. Not all challenges here are real blockers – some of them are just points to be kept in mind while others are problems which cannot be solved without a reasonable overhead.

5.1.4.1 Appropriate design of ontologies

If ontologies are used to describe security requirements and capabilities, designing the ontology becomes a security-critical task. The challenge here is that a developer does not only need

knowledge about ontologies but also about security-specific issues. However ontologies provide a good basis for elaborated analysis tools which can be used to support a developer when designing security-relevant ontologies. In combination with a well-designed set of predefined concepts and ontologies, it should be able to cope with this challenge.

5.1.4.2 Protection of ontologies

As soon as ontologies contain security-relevant data, e.g. the description of a service's security properties, the integrity and authenticity of these ontologies must be ensured. If attackers could modify the semantic descriptions of services, they could trick the application into using insecure services, erroneously declared to be secure. In case the service discovery process relies on semantic descriptions, there exist even more attack vectors, as described below. Possible solutions to this problem depend among others on the way how ontologies are stored. Especially for decentralised repositories without any common root of trust, it becomes difficult to guarantee authenticity of ontologies.

5.1.4.3 Protecting the discovery process

Discovering services just by querying for required attributes carries the risk of falsely described services – intentionally or not. If an attacker could announce services with arbitrary semantic descriptions, she can easily carry out denial of service attacks on the discovery mechanism. For example by declaring a service by fulfilling the maximum level of security, the discovery mechanism will prefer the attacker's service to every other service and thereby potentially render the whole SOA useless. Solving this problem requires some authority which appraises semantic description compared to the actual behaviour of the service. Depending on what is described in the service's annotations, this might even require the authority to have access to the service's source code which is an unrealistic assumption. Reputation mechanisms can help to at least reduce the risk of falsely described services. However, they require a notable overhead in terms of communication and implementation. Simply assuming every description to be correct is of course trivial but can be feasible in closed-world scenarios where attackers cannot create arbitrary services.

5.1.4.4 Interoperability with non-semantic services / requesters

When using semantic web service technologies as a basis for security, one must decide whether the interoperability between semantic and non-semantic web services should be kept up. An example for this would be a non-Hydra based application which should still access some managers of the Hydra middleware, e.g. a non-Hydra application making use of a security manager available in the environment. The basic difference between keeping up the interoperability and abandoning it is that in the former case, requirements and capabilities cannot be expected to be resolved automatically. During the next steps, it should be evaluated which of both approaches is more suited for Hydra and which disadvantages a mixture of semantic and non-semantic services could have for the middleware.

5.1.5 Caching Principles

While the semantic technologies in combination with the service oriented architecture of Hydra, enhance the functionality and usability of the Hydra middleware, the quality of service must also be attained to sufficient levels. This may pertain to the accessibility of both devices and device services. As many Hydra applications will be designed for networked and distributed environments, it is foreseen that caching techniques could be exploited on several levels in the Hydra architecture to improve accessibility and performance of device and service use.

5.2 Conclusions

In this document we have described and analysed different standards for semantic web services. We continued with an analysis of the requirements for semantic interoperability in Hydra and described our approach to semantic web services design.

To summarise, Hydra's technological innovations in Semantic Web Service Design will be achieved in the following areas:

- *Ontology-based Device and Service Descriptions*: The whole Hydra middleware will be driven from ontology descriptions for devices and services, thus making Hydra a highly configurable solution that can be easily deployed in many different applications.
- *Semantic Discovery and Advertising of Networked Devices and their Services*: Hydra will use a multi-layered approach to device and service discovery covering physical, network and semantic discovery. The physical discovery detects devices, irrespective of which communication protocol they use. The network discovery is based on UPnP and makes a device know to the rest of the Hydra network. Finally the semantic discovery detects the type of device, depending on the device ontology.
- *Lightweight orchestration and composition of Device Services*: We advocate a lightweight approach to combining and composing calls to different device services. We will build on principles from approaches like BPEL, however existing orchestration approaches appear to be too complex and resource intensive to be used in Hydra. Therefore we will design a language that is simplified and tailored for use with devices.
- *Ontology-driven Invocation and Execution of Device Services*: We will use an approach based on a combination of the Hydra Device Ontology and SAWSDL for semantic annotation of device services to allow applications and other devices to dynamically invoke and execute services.
- *Secure Semantic Web Services for Devices*: In Hydra we will use semantic annotations to express security requirements and constraints on device services. This will play an important role both for discovery of services as well as invocation and execution of services.
- *Automatic generation of SWS Device Proxies*: To enable the Hydra middleware and consequently the Hydra developer to view and use all devices and services in a heterogeneous fashion, automatic generation of semantic web service proxies will be necessary for the non-Hydra enabled devices. When a non-Hydra device is discovered and identified, a proxy for this device is generated using the information in the device and service ontologies.
- *Caching principles*: As many Hydra applications will be designed for networked and distributed environments, it is foreseen that caching techniques could be exploited on several levels in the Hydra architecture to improve accessibility and performance of device and service use. As mentioned previously, work on caching principles will be part of our future work, rather than in this iteration.

6. References

[1] Alonso, G. *Web Services – Concepts, Architectures and Applications*. Springer, 2004.

[2] Ardagna, C., Damiani, E., Vimercati, S.d. and Samarati, C.F.P. Offline Expansion of XACML Policies Based on P3P Metadata *Web Engineering*, , 2005, 363-374.

[3] Berners-Lee, T., Hendler, J. and Lassila, O. The Semantic Web. *Scientific American* (May 2001). 28-37.

[4] Chowdhury, M.M.R., Gomez, J.M., Noll, J. and Crespo, A.G. SemID: Combining Semantics with Identity Management *Proceedings of SECURWARE: International Conference on Emerging Security Information, Systems and Technologies, 2007*, , 2007.

[5] Denker, G., Kagal, L., Finin, T., Paolucci, M., Srinivasan, N. and Sycara, K. Security For DAML Web Services: Annotation and Matchmaking *Proceedings of the Second International Semantic Web Conference (ISWC 2003)*, , 2003, 335-350.

[6] Harmelen, F.v. and I. Horrocks FAQs on OIL: The Ontology Inference Layer. *IEEE Intelligent Systems,* (Nov./Dec. 2000). 69-72.

[7] Hendler, J. and McGuinness, D. The DARPA Agent Markup Language. *IEEE Intelligent Systems* (Nov./Dec. 2000). 69-72.

[8] Huhns, M.N. and Singh, M.P. Service-Oriented Computing: Key Concepts and Principles. *IEEE Internet Computing*, *9* (1). 75-81.

[9] Hydra. D6.2 MDA Design Document *Hydra Project Deliverable*, IST project 2005-034891, 2007.

[10] Kagal, L. Rei: A Policy Language for the Me-Centric Project Enterprise Systems Data Management Laboratory, HP Laboratories Palo Alto, 2002, .

[11] Kagal, L., M. Paoucci, N. Srinivasan, G. Denker, T. Finin and Sycara, K. Authorization and Privacy for Semantic Web Services *IEEE Intelligent Systems (Special Issue on Semantic Web Services)* (19). 50-56.

[12] Kang, S., Kim, D., Lee, Y., Hyun, S.J., Lee, D. and B. Lee, B. Semantic Service Discovery Network for Large-Scale Ubiquitous Computing Environments. *ETRI Journal* ( ). 545-558.

[13] Kolovski, V., Hendler, J. and Parsia, B. Formalizing XACML Using Defeasible Description Logics *WWW '07: Proceedings of the 16th international conference on World Wide Web*, ACM, University of Maryland - College Park 2007, 677-686.

[14] Kolovski, V., Hendler, J., Parsia, B. and Katz, y. Expressing WS-Policies in OWL *Workshop on Policy Management for the Web - 14th International World Wide Web Conference*, , 2005.

[15] Kolovski, V., Hendler, J., Parsia, B. and Katz, y. Representing Web Service Policies in OWL-DL *4th International Semantic Web Conference, ISWC 2005*, , 2005.

[16] Kopecký, J., Moran, M., Vitvar, T., Roman, D. and Mocan, A. WSMO Grounding *WSMO Working Draft*, Available at: http://www.wsmo.org/TR/d24/d24.2/v0.1/, 2007.

[17] Kopecký, J., Roman, D., Moran, M. and Fensel, D. Semantic Web Services Grounding *Proceedings of the International Conference on Internet and Web Applications and Services (ICIW'06)*, Available: http://dip.semanticweb.org/documents/Kopecky-Semantic-Web-Services-Grounding.pdf, 2006.

[18] Martin, D., Paolucci, M., McIlraith, S., Burstein, M., McDermott, D., McGuinness, D., Parsia, B., Payne, T., Sabou, M., Solanki, M., Srinivasan, N. and Sycara, K. Bringing semantics to web services: The OWL-S approach *Int. Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, CA, 2004.

[19] Martin, D., Paolucci, M. and Wagner, M. Bringing Semantic Annotations to Web Services: OWL-S from the SAWSDL Perspective. *The 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC/ASWC2007)*, Busan, South Korea, 2007, 337--350.

[20] McGuinness, D.L. and Harmelen, F.v. OWL Web Ontology Language Overview *W3C Recommendation: http://www.w3.org/TR/owl-features/*, W3C, 2004.

[21] McIlraith, S.A., Son, T.C. and Zeng, H. Semantic Web Services. *IEEE Intelligent Systems* (March 2001). 69-72.

[22] Noll, J., Chowdhury, M.M.R., Kálmán, G. and Gomez, J.M. Semantically supported Authentication and Privacy in Social Networks *Proceedings of SECURWARE: International Conference on Emerging Security Information, Systems and Technologies, 2007*, , 2007.

[23] Paolucci, M., Kawamura, T., Payne, T.R. and Sycara, K. Importing the Semantic Web in UDDI - *Services and the Semantic Web (ESSW02)*, 2002.

[24] Paolucci, M., Kawamura, T., Payne, T.R. and Sycara, K. Semantic Matching of Web Services Capabilities *International Semantic Web Conference (ISWC2002)*, 2002.
[25] Paolucci, M., Wagner, M. and Martin, D. Grounding OWL-S in SAWSDL *The International Conference on Service Oriented Computing*, Vienna, Austria, 2007.
[26] Polleres, A. and Lara, R. A Conceptual Comparison between WSMO and OWL-S *WSMO Working Group working draft*, Available at: http://www.wsmo.org/2004/d4/d4.1/v0.1, 2005.
[27] Tonti, G., Bradshaw, J.M., Jeffers, R., Montanari, R., Suri, N. and Uszok., A. Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei, and Ponder *The SemanticWeb - ISWC 2003*, Springer 2003, 419-437
[28] Uszok, A. and Bradshaw, J. KAoS Policies for Web Services 2003, .
[29] W3C-OWLs. OWL-S: Semantic Markup for Web Services *W3C Member Submission : http://www.daml.org/services/owl-s/1.1/overview/*, W3C, 2004.
[30] W3C-SAWSDL. Semantic Annotations for WSDL and XML Schema. Farrell, J. and Lausen, H. eds. *W3C Recommendation: www.w3.org/TR/2007/REC-sawsdl-20070828/* W3C, 2007.
[31] WSML. Web Service Modeling Language (WSML) *W3C Member Submission : http://www.w3.org/Submission/WSML/*, W3C, 2005.
[32] ESSI WSMO working group. 2008-01. http://www.wsmo.org/wsml/
[33] ESSI WSMO working group. 2008-01. http://www.wsmo.org/

7. Appendix: Requirements for Hydra Semantic Web Services

This section will list the relevant Volere requirements that are addressed in this design document. The table lists the current requirements related to Service Oriented Architecture (SOA) and Semantic Web Services (SWS). Work on the requirements continues throughout the project.

ID	Description	Rationale	Fit Criteria
17	When applicable, middleware interfaces are exposed by WSA-compatible services	Web Service Architecture (WSA; http://www.w3.org/TR/ws-arch/) introduces a common definition of what a web service is and describes minimal characteristics of what is common to all web services. When web services are used in Hydra, they should comply to WSA	In min. 90% of all cases, Hydra web service interfaces are realized as WSA-compatible web services. In the remaining cases, web services use proprietary formats.
21	Hydra should be a Service-Oriented Architecture (SOA)	Hydra should be a SOA per the Description of Work of the project	Hydra is compatible to the SOA-definition by OASIS: http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf
104	Automatic Discovery of Services	It should be possible to configure the middleware to discover available services that meets defined criteria.	8 of 10 services are automatically discovered.
111	Dynamic Web Service Binding	Middleware should be able to after device discovery and categorisation expose a new device as a web service that can be called without re-compilation.	New devices are callable and controllable in 7 out of 10 cases.
112	Dynamic Web Service Generation	Configuration tool that is able to generate the necessary interfaces to wrap the device functionality as a web service.	7 of 10 device functionalities are automatically represented as web services
113	Composition (of services and devices)	In order to enhance or replace application level functions it will be useful to be able to compose services and devices from different providers and/or manufacturers into high level	Service composition during design-time is possible.

		services/devices	
114	Semantic enabling of device web services	Middleware should be able to attach semantic descriptions to device web services based on device ontology.	7 of 10 device are semantically enabled.
129	Support for Semantic Web Standards for Device Communication	Middleware should support different semantic web standards, including OWL-S, WSMO, and selected parts of WS-*	Support for at least OWL-S and WSMO
157	Availability of combined services	A developer wants to easily access a higher level service which is in fact a combination of multiple services	High level services, consisting of at least two basic services, can be composed manually by the developer but this will not be done automatically.
158	There should be a hook-up-service	When the developer creates a new application/device he wants to have a broker that can supply him with all available services that match certain criteria.	A request for a specific service according to specific keywords results in the provision of the corresponding service in 8 out of 10 cases
159	Service brokers must be organized in a hierarchical way	With hierarchical brokers the system becomes more robust and scalable. Users do not want that everything acts up in case of a fire and a broker goes down. Additionally hierarchical brokers allow for having certain rules/services only within a sub-domain.	Brokers are organized hierarchically
160	Search masks for device/service discovery	When the developer needs a service he wants to be able to define search criteria for discovery of services	Search criteria can be specified and are respected by search services
164	Support for Service standards	Middleware should support widely used standards for service description, discovery, orchestration and execution.	Standards defined by W3C and OASIS implemented.
180	Service mediating network connections according to different qualities	There should be a service which lists different network connections depending on specified properties (connection speed, encryption). Devices can then negotiate such connections with remote devices, without the need to take care about	In 9 out of 10 cases devices should be able to automatically negotiate their networking condition.

		the networking details	
185	Middleware provides basic services	In order to program AmI applications, the middleware must provide basic services. This makes life easier for application developers .Basic services provide e.g. methods to query available devices and services or to pass messages between components	Middleware provides a set of basic services that at least contain basic functionality, that is needed by all services, like communication and a service / device registry.
196	Basic Service Registry	Services should register at a basic service/module of the middleware in order to provide a base for service orchestration	All services should be itemised at the Basic service registry
197	Services define their communication needs in terms of needed QoS parameters	The services define their communication needs in terms of needed QoS parameters (needed bandwidth, needed quality...) without specifying the technical details. The middleware is free to choose the appropriate networking matching the specified needs	Every service specifies its QoS parameters
198	A service broker is responsible to provide services according to specific keywords	Service discovery should be enhanced by a service broker module/service as basic service of the middleware that enables the search for services according to specific keywords	Requests according to specific keywords will be provided a corresponding service in 8 out of 10 cases.
207	Service selection by context	In order to select an appropriate service for a specific task, contextual information, like the spatial position, must be taken into account. Hydra must provide a method to specify a desired service by contextual parameters. For example, if a certain room in a building is specified in a search request for a service, only services are returned that are relevant in the current user's location and context.	In search requests for a specific service, contextual information like a spatial position is allowed.
209	Middleware has a service for providing information about the technical	In order for the services to query the available infrastructure the middleware	A services at the middleware provides information about more

	environment/infrastructure	should provide such a service	than 95% of the technical environment/infrastructure
211	There are components/services in the middleware that integrate subsystems	The integration of basic systems to subsystems should ease the configuration of higher level services. Higher level services could then consist of a combination of basic systems	It should be possible to combine basic services to higher level services. At least one higher level service relying on a combination of basic services exists.
214	A decision component/service should exist	There should be a decision component that is able to take actions according to specified rules or reasoning components.	At least one decision component in the middleware
216	The middleware should have a graceful degradation service	Services should be organised in a cascade of services in order to allow an orchestration of services providing best possible services down to basic services automatically, according to their availability	Service orchestration is possible n a hierarchical way. An automatic selection of the best service is possible within max. 500 msec.
217	The middleware should ensure high robustness of services	In order to ensure the service support of important components in the system, the middleware should provide a highly robust service structure.	Breakdown of crucial services of the middleware in less than 1 case per 100 hours of operation.
225	Interactions and consequences of changes to services on other services should be highlighted	The developer should have a tool that helps him understand the complex interactions of services and the possible consequences of changes on one middleware service to other middleware services	A service monitor that is able to show interactions with other services is implemented
229	Services are responsible for authentication	The single service should be responsible for authentication request in order to ensure a robust and secure system	All security critical services trigger authentication requests
239	Automatic service diagnostic for security relevant services	Security relevant services should provide a self-diagnostic services that provides an overview of all security-relevant features	Self-diagnostics in all security relevant services implemented
290	Share service orchestration between users	Service orchestration definition should be shared between developer users, in order to allow a distribution	Service orchestration definitions can be shared between users

		of useful service orchestration to other developers	
291	Quality of Service as search criteria for service selection	The selection of appropriate services for a given task requires the reflection of QoS-related search criteria such as cost, performance, etc.	QoS-criteria can be used in the selection of services in 95% of all cases
320	Separate domain-oriented services and user interface services architecturally	This is a standard architectural design tactic to enhance modifiability	90% of the modules of the architecture properly separate layers for domain services and interfaces.
325	Support aggregation and separation of devices and services	Devices and services may exist in a separate application where they should not be influenced by nearby (wireless) devices such as in the case of an apartment. Thus it should be possible to view a set of services/devices as an aggregate that is separated and isolated from other sets of services/devices	Check support for aggregation and separation of devices/services
329	Middleware provides domain-independent services	A lot of the services needed in the apartment scenario are also needed in other scenarios (persistence, logging, visualization, ...). These should be abstracted and built and provided as part of Hydra	Large parts of the building-automation scenario can be built by reusing configurable services from across other application domains.
358	Developer must be able to semantically define security requirements	If developers are to make devices that can co-operate through other protocols and security mechanisms, they have to be able to describe the inherent security requirements in a semantic interoperable language. It is not enough just to use a specific protocol's security as this does NOT tell WHY he uses it and WHAT he really needs for the application to proceed.	On the one hand Hydra supports the semantic description of security requirements and provides mechanisms to translate those requirements into device specific protocols automatically. On the other hand Hydra provides means in order to analyse (prospectively) existing device specific proprietary security protocols. Hydra can detect incompatibilities of different protocols' security mechanisms.
366	Services should run on embedded devices	Service-orientation is a good match for many embedded devices. Web services will	Hydra supports services on embedded devices (Initial target should be Develco's

		provide a gateway to many applications and it would be beneficial to be able to structure all of the communication in a system using the same primitives. Depending on the resources (energy, processing capacity) available such a service may run on the device or on a proxy	DevCom 02 ZigBee module)
389	Service browsing in device ontology	It must be possible to view services as central building blocks, thus an application developer should be able to browse the device ontology from a service perspective, in addition to a device perspective.	A developer can find services and use them in development, without an a priori knowledge of the devices that implement the services.
394	Stateful service orchestration	In order to specify service workflows we need to be able to keep state between the execution of the stateless services.	Service orchestration can be done by creating service workflow definition.
419	Backbone - Device services and resources announcement through the Gateway	Each device either Hydra-enabled or non-Hydra-enabled (through proxies) must announce its services and resources in the Backbone through its Gateway	90% of devices announce their services and resources in the Backbone through the Gateway

8. Appendix: SWS related Managers

In deliverable D6.2 "MDA Design Document" all managers of WP6 are specified. Here we include the three that is most relevant to the Semantic Web Service Design for reference purposes.

8.1 Application Service Manager

8.1.1 Purpose

The purpose of the Application Service Manager is to discover, create and execute semantic (web) services on top of devices. It adds a semantic layer and complements above the Application Device Manager with a service perspective. Services might map to several device functionalities.

Main Functions:

- Service discovery
- Semantic service creation (service orchestration/clustering and mapping to device service)

8.1.2 Related WP6 requirements

[Hydra-104] Automatic Discovery of Services	
Status:	Part of specification
Requirement Type:	Functional
Workpackage:	WP6
Rationale:	It should be possible to configure the middleware to discover available services that meets defined criteria.
Source:	St. Augustin
Fit Criteria:	8 of 10 services are automatically discovered.
Developer Satisfaction:	high
Developer Dissatisfaction:	high

[Hydra-113] Composition (of services and devices)	
Status:	Part of specification
Requirement Type:	Functional
Workpackage:	WP6
Rationale:	In order to enhance or replace application level functions it will be useful to be able to compose services and devices from different providers and/or manufacturers into high level services/devices
Source:	WP6 MDA Focus Group, WP6 eHealth Focus Group
Fit Criteria:	Service composition during design-time is possible.
Developer Satisfaction:	high
Developer Dissatisfaction:	high

[Hydra-114] Semantic enabling of device web services	
Status:	Part of specification

Requirement Type:	Functional
Workpackage:	WP6
Rationale:	Middleware should be able to attach semantic descriptions to device web services based on device ontology.
Source:	WP6 SoA Focus Group
Fit Criteria:	7 of 10 devices are semantically enabled.
Developer Satisfaction:	very high
Developer Dissatisfaction:	high

[Hydra-119] Domain modelling support	
Status:	Part of specification
Requirement Type:	Functional
Workpackage:	WP6
Rationale:	The middleware and IDE should be able to interface with application domain frameworks representing core concepts and functions of specific application domains. These could in the most basic form be represented by UML Profiles, or domain ontologies.
Source:	WP6 MDA focus group
Fit Criteria:	The Hydra IDE supports at min 2 defined domain modelling frameworks.
Developer Satisfaction:	high
Developer Dissatisfaction:	high
Dependencies:	117

[Hydra-120] Multiple Device Virtualisations	
Status:	Part of specification
Requirement Type:	Functional
Workpackage:	WP6
Rationale:	It should be possible to have several different views/virtualisations of a device depending on context and applications.
Source:	WP6 MDA Focus Group
Fit Criteria:	At least 2 different virtualisations are provided
Developer Satisfaction:	high
Developer Dissatisfaction:	high

[Hydra-129] Support for Semantic Web Standards for Device Communication	
Status:	Part of specification
Requirement Type:	Functional
Workpackage:	WP6
Rationale:	Middleware should support different semantic web standards, including OWL-S, WSMO, and selected parts of WS-*
Source:	WP SoA Focus Group
Fit Criteria:	Support for at least OWL-S and WSMO
Developer Satisfaction:	high
Developer Dissatisfaction:	high

[Hydra-325] Support aggregation and separation of devices and services	
Status:	Part of specification
Project:	Hydra
Requirement Type:	Functional
Workpackage:	WP6
Rationale:	Devices and services may exist in a separate application where they should not be influenced by nearby (wireless) devices such as in the case of an apartment. Thus it should be possible to view a set of services/devices as an aggregate that is separated and isolated from other sets of services/devices
Source:	UAAR focus group
Fit Criteria:	Check support for aggregation and separation of devices/services
Developer Satisfaction:	neutral
Developer Dissatisfaction:	neutral

[Hydra-372] Interfacing with external systems	
Status:	Part of specification
Requirement Type:	Functional
Workpackage:	WP6
Rationale:	Searching and using external services in decision support and application intelligence must be supported
Source:	WP 6 Focus Group, WP2 Input
Fit Criteria:	Access to external systems using web service protocols (WS-I Basic Profile) is supported
Developer Satisfaction:	neutral
Developer Dissatisfaction:	neutral

[Hydra-376] Security requirements must be part of the Hydra MDA	
Status:	Part of specification
Requirement Type:	Functional
Workpackage:	WP6
Rationale:	Security must be defined to be resolved semantically
Source:	WP 6 Focus group Kosice
Fit Criteria:	Security model can be defined semantically
Developer Satisfaction:	high
Developer Dissatisfaction:	high

8.1.3 Components

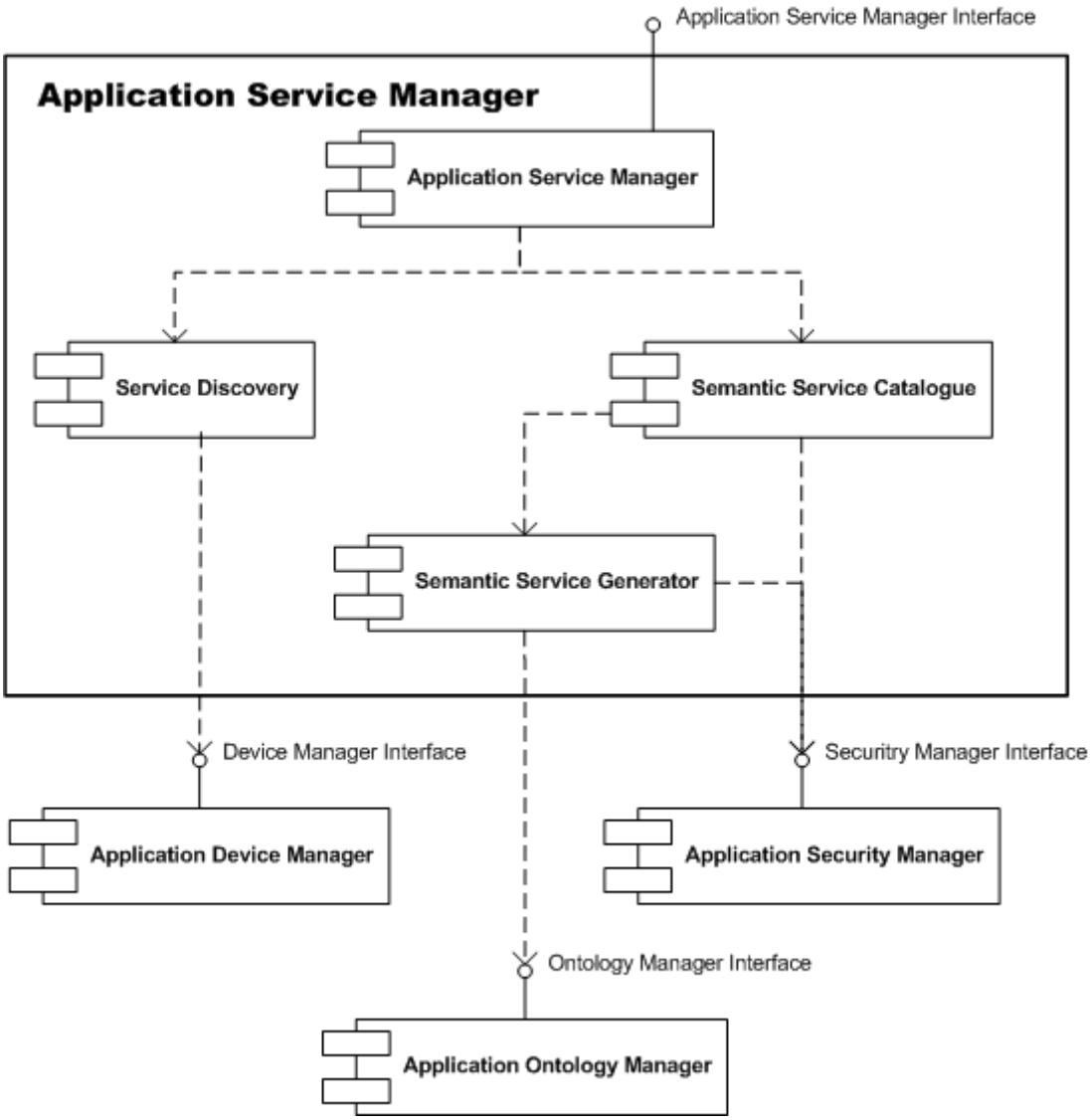

Figure 19: Application Service Manager

Service Discovery Module

One of the major functions of the Service Manager is to discover new services in the network. This is taken care of by the Service Discovery Module. It will use the Device Manager to find out about services offered by different devices.

Semantic Service Catalogue:

The Semantic Service Catalogue keeps track of and manages all service offered within one application. It can be queried about existing services. It can also provide semantic service interfaces for the different services upon request.

Semantic Service Generator

The Semantic Service Generator is responsible for generating a semantic service interface for services offered by devices. It will create a software wrapper around the device services which other modules can use. The generated software will support a semantic-based service interface. It will support several semantic web standards, at least OWL-S and WSMO.

8.1.4 Dependencies

Application Service Manager, Application Ontology Manager and Application Security Manager

8.1.5 Interface

string ApplicationServiceManager::ProcessErrorMessage (XmlNode *theMessage*)

Processes an error message.

Parameters:

theMessage The error message as an XML Node.

Returns:

A description of the error.

string ApplicationServiceManager::ProcessErrorMessageString (string *theMessage*)

Processes an error message.

Parameters:

theMessage The error message as a string.

Returns:

A description of the error.

bool ApplicationServiceManager::HasService (string deviceid, string *serviceid*)

Checks if a service is available.

Parameters:

service serviceid The service name.
deviceid The device to be queried

Returns:

True if service is available otherwise false.

string ApplicationServiceManager::GetServiceDescription (string *devicetype*, string *serviceid*)

Retrieves a device description.

Parameters:

devicetype The device type as a string.
serviceid The service id as a string.

Returns:

A string containing a service description in XML format.

XmlNode ApplicationServiceManager::GetServiceDescriptionAsXML (string *devicetype*, string *serviceid*)

Retrieves a device description.

Parameters:

devicetype The device type as a string.
serviceid The service id as a string.

Returns:

An XmlNode containing a service description.

string ApplicationServiceManager::GetServices(string *type*)

Retrieves a list of available services.

Parameters:

type The device service type as a string.

Returns:

A string containing the list of available devices services in XML format.

XmlNode ApplicationServiceManager::GetServicesAsXML (string *type*)

Retrieves a list of available services.

Parameters:

type The device service type as a string.

Returns:

An XmlNode containing the list of available service.

string ApplicationServiceManager::Invoke(XmlNode invokeMessage)

Generic method to invoke any method in a service on a device.

Parameters:

invokeMessage The invoking message containing serviced, methodname, parameters, values

Returns:

The result of invoking the method.

8.2 Application Orchestration Manager

8.2.1 Purpose

The Application Orchestration Manager provides support for composite services and workflows. It is an execution engine for the Hydra Device Orchestration Language ("DOLL").

Main Functions:

- Execute call sequences consisting of invocations of Device services
- Provide scheduling of notifications and service calls for Hydra applications

8.2.2 Related WP6 requirements

[Hydra-113] Composition (of services and devices)	
Status:	Part of specification
Requirement Type:	Functional
Workpackage:	WP6
Rationale:	In order to enhance or replace application level functions it will be useful to be able to compose services and devices from different providers and/or manufacturers into high level services/devices
Source:	WP6 MDA Focus Group, WP6 eHealth Focus Group
Fit Criteria:	Service composition during design-time is possible.
Developer Satisfaction:	high
Developer Dissatisfaction:	high

[Hydra-392] Rules for selection of alternative devices	
Status:	Part of specification
Requirement Type:	Functional
Workpackage:	WP6
Rationale:	The developer user should be able to specify how devices can replace or complement each other. This is relevant in situations when a device fails and another device exists which can provide a replacement service, or, when different levels of quality of service are available.
Source:	WP6 eHealth focus group

Fit Criteria:	In the SDK, constructs are available that allow the developer to specify rules for when and how devices and services can be interchanged and combined.
Developer Satisfaction:	neutral
Developer Dissatisfaction:	neutral

[Hydra-376] Security requirements must be part of the Hydra MDA	
Status:	Part of specification
Requirement Type:	Functional
Workpackage:	WP6
Rationale:	Security must be defined to be resolved semantically
Source:	WP 6 Focus group Kosice
Fit Criteria:	Security model can be defined semantically
Developer Satisfaction:	high
Developer Dissatisfaction:	high

8.2.3 Components

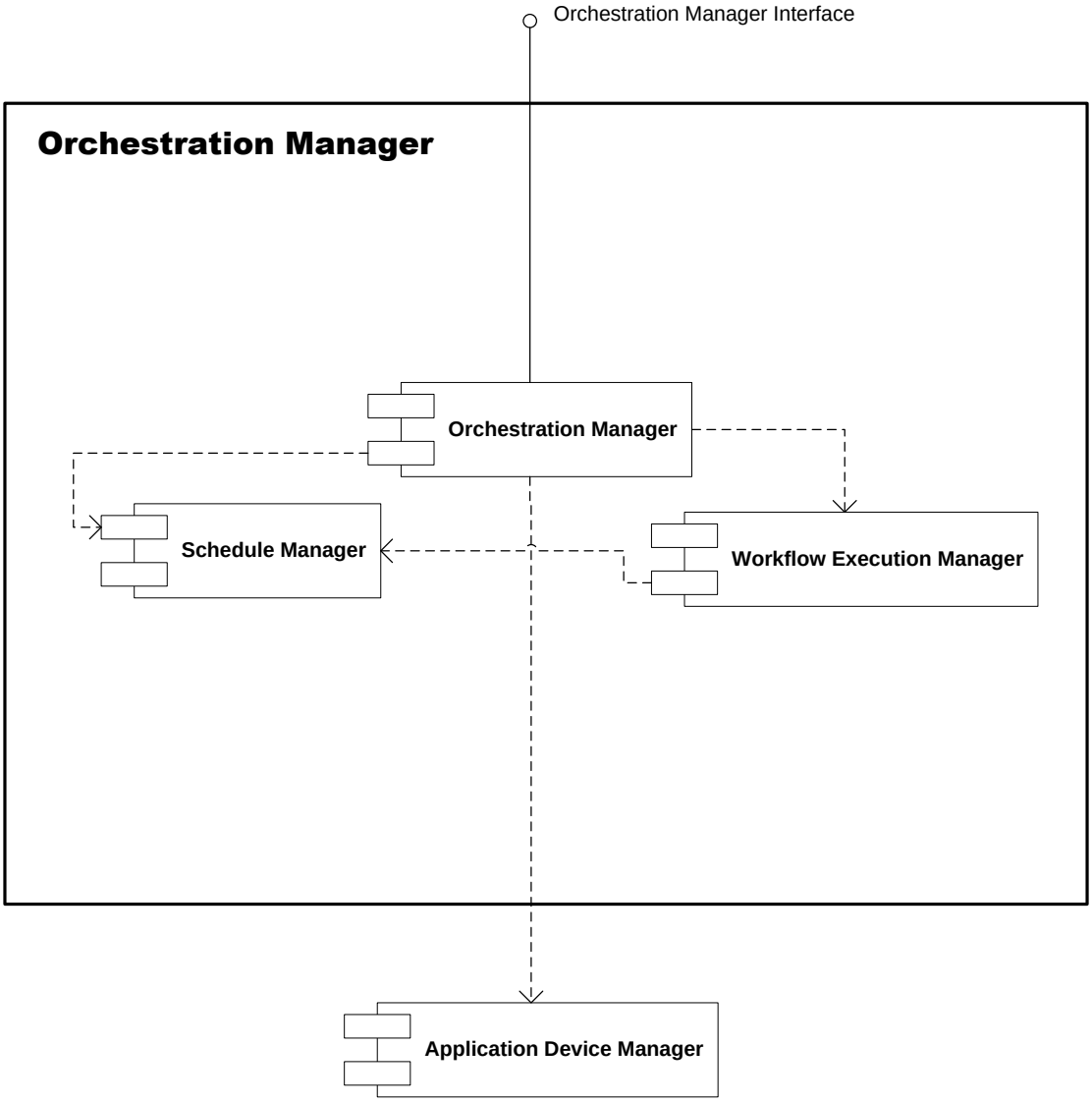

Figure 20: Application Orchestration Manager

Schedule Manager: The scheduler is responsible for running tasks or notifying applications when a specific criterion is met. Such a criteria can be a specific (possibly recurring) time, system startup, system shutdown.

Workflow Execution Manager: The workflow execution module interprets process descriptions and executes a set of services. These processes may represent a complex service composed of other services or part of a Hydra application.

Dependencies: Application Device Manager

8.2.4 Interface

XmlNode OrchestrationManager::LoadProcessDescription (XmlNode processDescription) Loads a process description into the Orchestration Manager.

Parameters:

processDescription The ontology deviceId.

Returns:
A XML node containing the result of the operation and invocation data or the description of any errors that occurred during method invocation.

XmlNode OrchestrationManager::ListProcessDescriptions () Lists process descriptions previously loaded into the Orchestration Manager.

Parameters:

Returns:
A XML node containing all process descriptions loaded into the Orchestration Manager.

XmlNode OrchestrationManager::InvokeProcessDescription (XmlNode invocationData) Invokes a process description previously loaded into the Orchestration Manager.

Parameters:
invocationData An XML node with data identifying the process and invocation data for invocation.

Returns:
A XML node containing the result of and data returned from the invocation or the description of any errors that occurred during invocation.

8.3 Device Device Manager

8.3.1 Purpose

The Device Device Manager handles several service requests and manages the responses.

Main Functions:

- Maps requests to device services
- Response generation
- Advertising Hydra device description
- Advertises device services

8.3.2 Related WP6 requirements

[Hydra-91] Any Hydra device should have an associated description	
Status:	Part of specification
Requirement Type:	Functional
Workpackage:	WP6
Rationale:	For management, search and discovery purposes, all Hydra enabled devices should be described (classified) according to the Hydra device ontology.
Source:	WP6 MDA scenario
Fit Criteria:	Any device associated to a Hydra application is also included in the Hydra device ontology, and its description can be retrieved.
Developer Satisfaction:	high
Developer Dissatisfaction:	high

[Hydra-92] Rule-based configuration of devices	
Status:	Part of specification
Requirement Type:	Functional
Workpackage:	WP6

Rationale:	The possibility for the developer to specify device behaviour using rules. It should be possible to derive and re-use rules from pre-existing or generic rule sets for application domains. Possibility to hide device specific details.
Source:	WP6 MDA Focus Group and WP6 eHealth focus group
Fit Criteria:	The functionality (services) of a device is accessible (by user or application) thru a rule-based interface.
Developer Satisfaction:	high
Developer Dissatisfaction:	high

[Hydra-108] Device discovery	
Status:	Part of specification
Requirement Type:	Functional
Workpackage:	WP6
Rationale:	Middleware should be able to detect new device that enters the network
Source:	St. Agustin
Fit Criteria:	7 of 10 devices are discovered
Developer Satisfaction:	very high
Developer Dissatisfaction:	high

[Hydra-109] Device Virtualization	
Status:	Part of specification
Requirement Type:	Functional
Workpackage:	WP6
Rationale:	The complexity of devices may be hidden, or simplified, by means of virtual device interfaces; these would correspond to "views" on device descriptions as provided by the Hydra device models (ontologies).
Source:	WP6 MDA scenario focus group
Fit Criteria:	An existing virtualization can be used to find exactly one proper Hydra device.
Developer Satisfaction:	neutral
Developer Dissatisfaction:	neutral

[Hydra-111] Dynamic Web Service Binding	
Status:	Part of specification
Requirement Type:	Functional
Workpackage:	WP6
Rationale:	Middleware should be able to after device discovery and categorisation expose a new device as a web service that can be called without re-compilation.
Source:	WP6 SoA Focus Group
Fit Criteria:	New devices are callable and controllable in 7 out of 10 cases.
Developer Satisfaction:	very high
Developer Dissatisfaction:	very high

[Hydra-114] Semantic enabling of device web services	
Status:	Part of specification

Requirement Type:	Functional
Workpackage:	WP6
Rationale:	Middleware should be able to attach semantic descriptions to device web services based on device ontology.
Source:	WP6 SoA Focus Group
Fit Criteria:	7 of 10 devices are semantically enabled.
Developer Satisfaction:	very high
Developer Dissatisfaction:	high

[Hydra-120] Multiple Device Virtualisations	
Status:	Part of specification
Requirement Type:	Functional
Workpackage:	WP6
Rationale:	It should be possible to have several different views/virtualisations of a device depending on context and applications.
Source:	WP6 MDA Focus Group
Fit Criteria:	At least 2 different virtualisations are provided
Developer Satisfaction:	high
Developer Dissatisfaction:	high

[Hydra-376] Security requirements must be part of the Hydra MDA	
Status:	Part of specification
Requirement Type:	Functional
Workpackage:	WP6
Rationale:	Security must be defined to be resolved semantically
Source:	WP 6 Focus group Kosice
Fit Criteria:	Security model can be defined semantically
Developer Satisfaction:	high
Developer Dissatisfaction:	high

8.3.3 Components

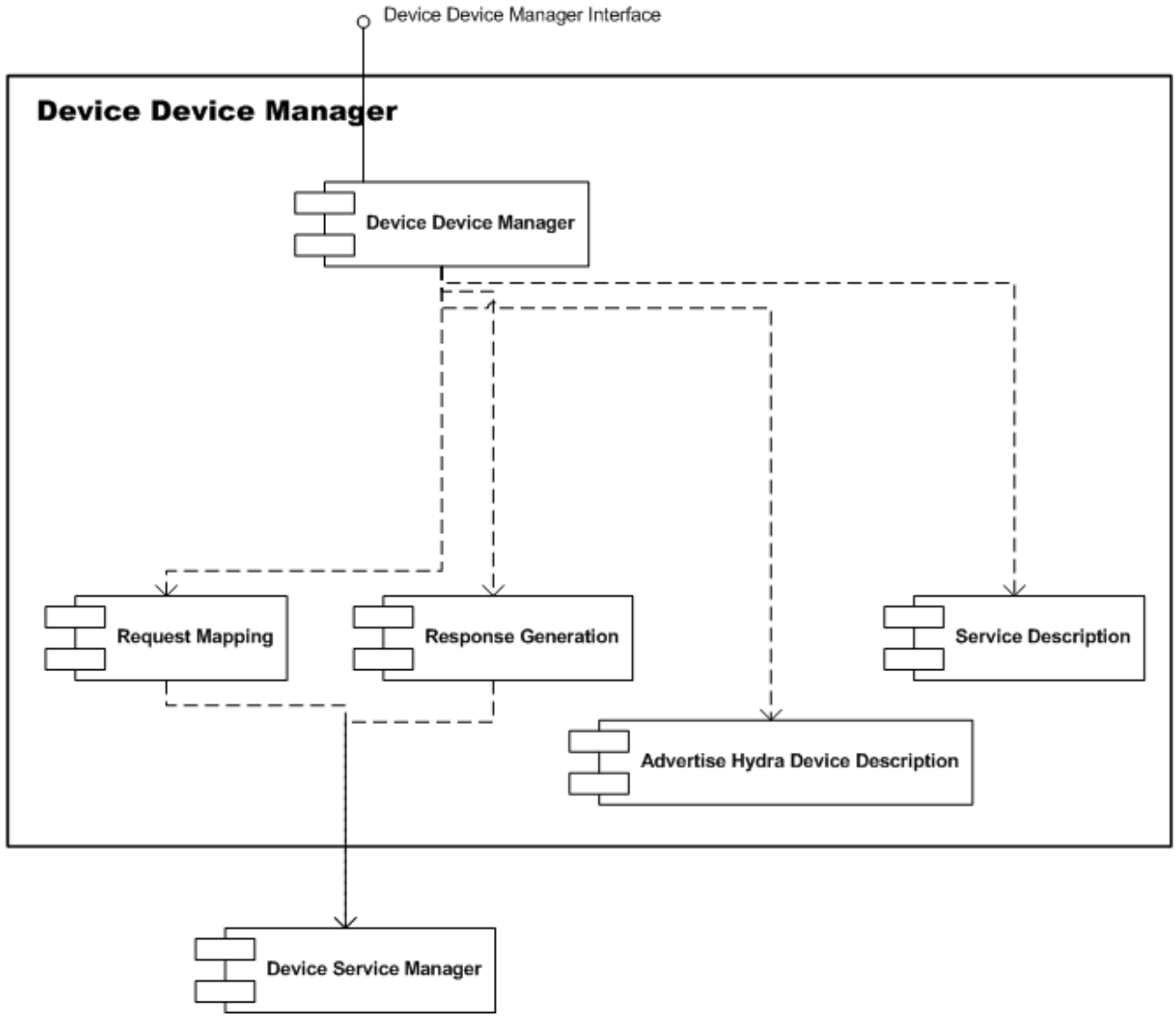

Figure 21: Device Device Manager

Advertise

This module is responsible for broadcasting the existence of the device to the outside world. It will support several discovery protocols, at least UPnP (Universal Plug and Play).

Request Mapping

This module maps a request from an outside caller to an internal service in the device.

Response Generator

This module maps translates the result of an internal service in the device to a response to the caller.

Service Description

This module can advertise and provide the service description of the device.

8.3.4 Dependencies

Device Service Manager

8.3.5 Interface

string DeviceDeviceManager::RegisterError(string *property*, string *errorcode*)

Registers an error condition.

Parameters:

property The error property as string.
errorcode The error code as string.

Returns:

A string containing the registered error.

string DeviceDeviceManager::SendErrorMessage(string *message*)

Sends an error message for a specific device.

Parameters:

message The error message as string.

Returns:

A string containing the sent error message.

string DeviceDeviceManager::Invoke(string *serviceid*, string *methodName*, string *parameters*, string *values*)

Executes a specific method for a service (using the device service manager).

Parameters:

serviceid The serviceid as string.
methodName The methodName as string.
parameters A comma delimited string with the parameter names.
parameters A comma delimited string with the parameter values (Matched against "parameters").

Returns:

A string indicating the result of the execution.

string DeviceDeviceManager::GetDeviceStatus ()

Retrieves the device status (using the device service manager).

Returns:

A string with the device status.

string DeviceDeviceManager::AddDAC(string dacaddress)

Adds a device application catalogue to this device list of catalogues when the device is discovered.

Returns:

A string with the device status.

string DeviceDeviceManager::GetDACList()

Returns the list of device application catalogues where the device has been discovered.

Returns:

A string with the device application catalogues.