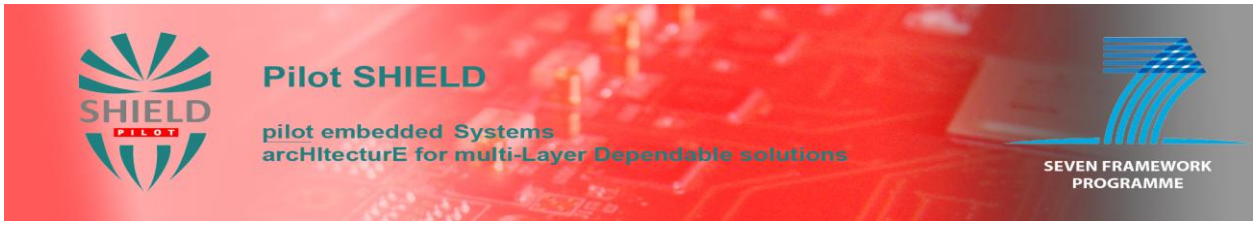

Project no: 100204

p-SHIELD

pilot embedded Systems architecture for multi-Layer Dependable solutions

Instrument type: Capability Project

Priority name: Embedded Systems (including RAILWAYS)

D5.3: pSHIELD Semantic Models Report

Due date of deliverable: M18 (30th September 2011)
Actual submission date: M19 (15th November 2011)

Start date of project: 1st June 2010 Duration: 19 months

Organisation name of lead contractor for this deliverable: SE

Revision [v1.0]

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2012)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Document Authors and Approvals			
Authors		**Date**	**Signature**
Name	**Company**		
Andrea Fiaschetti	Univ. “La Sapienza”		
Andrea Taglialatela	TRS		
Sergio Wanderlingh	TRS		
Giuseppe Fusco	TRS		
Andrea Morgagni	SelexElsag		
Renato Baldelli	SelexElsag		
Andi Palo	Univ. “La Sapienza”		
Vincenzo Suraci	Univ. “La Sapienza”		
Mohammad Chowdhury	CWIN		
Carlo Bruni	Univ. “La Sapienza”		
Roberto Cusani	Univ. “La Sapienza”		
Gaetano Scarano	Univ. “La Sapienza”		
Francesco Delli Priscoli	Univ. “La Sapienza”		
Alberto Isidori	Univ. “La Sapienza”		
Giorgio Koch	Univ. “La Sapienza”		
Claudio De Persis	Univ. “La Sapienza”		
Antonio Pietrabissa	Univ. “La Sapienza”		
Ljiljana Mijic	THYIA		
Spase Drakul	THYIA		
Gordana Mijic	THYIA		
Vlado Drakulovski	THYIA		
Nastja Kuzmin	THYIA		
Reviewed by			
Name	**Company**		
Josef Noll	Movation		
Sarfraz Alam	CWIN		
Andrea Fiaschetti	Univ. “La Sapienza”		
Approved by			
Name	**Company**		
Andrea Morgagni	SelexElsag		

Modification History		
Issue	**Date (DD/MM//YY)**	**Description**
Version 1	15/11/2011	Final version (v1.0)

Contents

Figures

Tables

Acronyms

Acronym	Meaning
ESD	Embedded System Device
ESs	Embedded Systems
L-ESD	Legacy Embedded System Device
MS	Middleware Service
MwA	Middleware Adapter
NC	Node Capability
NoA	Node Adapter
NS	Network Service
NwA	Network Adapter
pS-ESD	pSHIELD Embedded System Device
pS-MS	pSHIELD Middleware Service
pS-OS	pSHIELD Overlay Service
pS-P	pSHIELD Subsystem
pS-P	pSHIELD Proxy
pS-SPD-ESD	SPD Embedded System Device
SPD	Security Privacy Dependability
CC	Common Criteria
FUA	Faults with Unauthorized Access
HMF	Human-Made Faults
NFUA	Not Faults with Unauthorized Access
NHMF	Nonhuman-Made Faults
SoC	System on Chip

- This page intentionally left blank -

1 Executive Summary

Semantic technologies are becoming more and more attractive as they enable exploiting the benefits of the explicit knowledge of a domain in an effective way.

In pSHIELD, semantic technologies shall address the interoperability issues among different SPD technologies by enabling their composition. This goal shall be accomplished by building a framework that, in a consistent fashion, will provide a **methodology** for ontology building and verification, a suitable **meta-model** of the SPD in Embedded Systems, and a set of elementary functional components for **semantic management** of systems.

This framework will lay a groundwork to build, as a second step, an ontology of the SPD modules, capabilities and interfaces, that thanks to a semantic-aware model of all the exchanged information and control flows between the node, network, middleware and overlay layer, will allow an effective way to represent and reason about all the relevant entities by means of a common (shared) and consistent schema.

The structure of WP5, as conceived in the Technical Annex, is summarized in Figure 1.1, where one of the main novelty addressed by these activities (i.e. the composability mechanism) is represented as a closed loop system. In this context, Task 5.1 provides the enabling (semantic) technologies for system modeling, measuring and control.

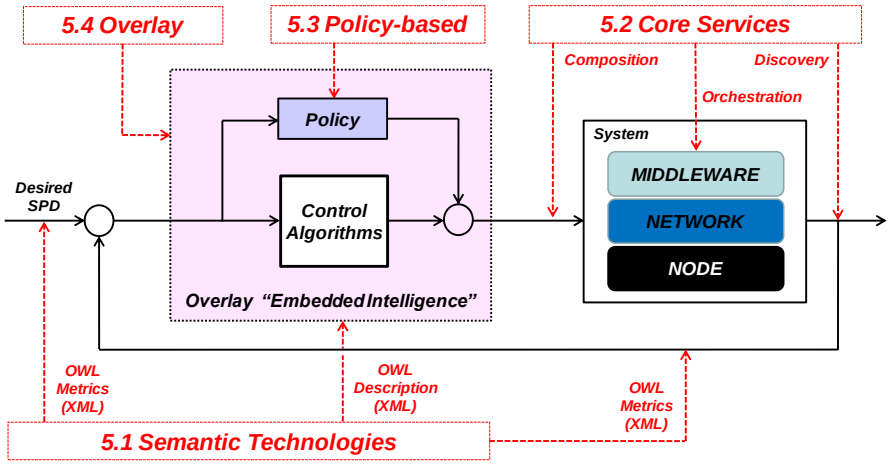

Figure 1.1 - Work Package 5 Structure

The document is structured as follows: in Section 4 an overview of the Semantic Tecnologies is provided; in Section 5 the Ontology Formalization procedure adopted for the pSHIELD project is described; then in section 6 the pSHIELD model is outlined, while in Section 7 some consideration about the potential inference mechanism underlying this models are reported, with particular focus on semantic composition.

The purpose of the present document is to introduce and describe the pSHIELD Semantic Models and procedures developed in Task 5.1. This deliverable is enriched with the internal prototype already presented in D5.1.

2 Introduction

2.1 Semantic interoperability

Over the last few years much work has been conducted in regards to the research topic of fully interoperability. The use of specific data models implies that making a whole system from its components is a process which is very time consuming and prone to the introduction of artificial layers that cause performance and overall control decay

The advantages of a successful model integration are obvious for many reasons:

- Quality improvement of model due to the availability of *large and complete information*.
- Improvement of *existing analysis* and application of the *new analysis*.
- Cost reduction resulting from the *multiple use of existing information sources*.
- Avoidance of redundant data and conflicts that can arise from *redundancy*.

As we shift from the problem of information integration to those of model integration, however, difficulties arising from organizational, competence questions and many other technical problems have to be solved. We distinguish different integration levels, that need to be solved in order to achieve complete integrated access to information:

- Syntactic Integration: Many standards have evolved that can be used to integrate different information sources. Beside classical database interfaces such as ODBC, web-oriented standards such as HTML and XML are gaining importance.
- Structural Integration: The first problem that passes a purely syntactic level is the integration of heterogeneous structures. This problem is normally solved by mediator systems defining mapping rules between different information structures.
- Semantic Integration: In the following, we use the term semantic integration or semantic translation, respectively, to denote the resolution of semantic conflicts, that make a one to one mapping between concepts or terms impossible.

Our approach provides an overall solution to the problem of information integration, taking into account all three levels of integration and combining several technologies, including standard markup languages, mediator systems and ontologies. In order to overcome the obstacles mentioned earlier, it is not sufficient to solve the heterogeneity problems separately. It is important to note that these problems can only be solved with a system taking all three levels of integration into account.

- **Syntactic Integration**: The typical task of syntactic data integration is, to specify the information source on a syntactic level. This means, that different data type problems can be solved (e. g. short int vs. int and/or long). This first data abstraction is used to re-structure the information source. The standard technology to overcome problems on this level are wrappers. Wrappers hide the internal data structure model of a source and transform the contents to a uniform data structure model.
- **Structural Integration**: The task of structural data integration is, to re-format the data structures to a new homogeneous data structure. This can be done with the help of a formalism that is able to construct one specific information source out of numerous other information sources. This is a classical task of a middleware which can be done with CORBA on a low level or rule-based mediators on a higher level. Mediators provide flexible integration of several information systems since it combines, integrates, and abstracts the information provided by the sources. Normally the sources are encapsulated by wrappers. Popular implementations of mediators have are rule driven: Usually, the rules in the mediator describe how information of the sources can be mapped to the integrated view. In simple cases, a rule mediator converts the information of the sources

into information on the integrated view. The mediator uses the rules to split the query, which is formulated with respect to the integrated view, into several sub-queries for each source and combine the results according to query plan. A mediator has to solve the same problems which are discussed in the federated database research area, i. e. structural heterogeneity (schematic heterogeneity) and semantic heterogeneity (data heterogeneity) Structural heterogeneity means that different information systems store their data in different structures. Semantic heterogeneity considers the content and semantics of an information item. In rule-based mediators, rules are mainly designed in order to reconcile structural heterogeneity. Where as discovering semantic heterogeneity problems and their reconciliation play a subordinate role. But for the reconciliation of the semantic heterogeneity problems, the semantic level must also be considered. Contexts are one possibility to describe the semantic level. A context contains ”meta data relating to its meaning, properties (such as its source, quality, and precision), and organization”. A value has to be considered in its context and may be transformed into another context (so-called context transformation).

- **Semantic Integration**: The semantic integration process is by far the most complicated process and presents a real challenge. As with database integration, semantic heterogeneities are the main problems that have to be solved within real time and embedded systems.
    - Generic semantic heterogeneity: Heterogeneity resulting from field- and object-based databases.
    - Contextual semantic heterogeneity: Heterogeneity based on different meanings of concepts and schemes.

In this project, we will focus on contextual semantic heterogeneity which is based on different semantics of the local schemata. In order to discover semantic heterogeneities, a formal representation is needed. Lately, WWW standardized markup languages such as XML and RDF have been developed by the W3C community for this purpose (W3C, 1998), (W3C, 1999). We will describe the value of these languages for the semantic description of concepts and also argue that we need more sophisticated approaches to overcome the semantic heterogeneity problem. Actually, we pursue an intergration/ interoperation process based on Ontologies.

XML and RDF have been developed for the semantic description of information sources. In order to overcome the purely visualization-oriented annotation provided e. g. by HTML, XML was proposed as an extensible language allowing the user to define his own tags in order to indicate the type of it’s content. Therefore, it followed that the main benefit of XML lies actually in the opportunity to exchange data in a structured way. Recently, this idea has been emphasized by introducing XML schemata that could be seen as a definition language for data structures. In the following paragraphs we sketch the idea behind XML and describe XML schema definitions and their potential use for data exchange. A data object is said to be XML document if it follows the guidelines for wellformed XML documents provided by the W3C community. The specification provide a formal grammar used in well-formed documents. In addition to the general grammar, the user can impose further grammatical constraints on the structure of a document using a document type definition (DTD). A XML document is valid if it has an associated type definition and complies to the grammatical constraints of that definition. A DTD specifies elements that can be used in an XML document. In the document, the elements are delimited by a start and an end tag. It has a type and may have a set of attribute specifications consisting of a name and a value.The additional constraints in a DTD refer to the logical structure of the document, this especially includes the nesting of tags inside the information body that is allowed and/or required. Further restrictions that can be expressed in a DTD concern the type of the attributes and default values to be used when no attribute value is provided.

An XML schema itself is, an XML document defining the valid structure of an XML document in the spirit of a DTD. The elements used in a schema definition are of the type ’element’ and have attributes that are defining the restrictions already mentioned above. The information in such an element is a list of further element definitions that have to be nested inside the defined element. Furthermore, XML schema have some additional features that are very useful to define data structures such as:

- Support for basic data types.
- Constraints on attributes such as occurrence constraints.

- Sophisticated structures such as type definition derived by extending or restricting other types.
- A name-space mechanism allowing the combination of different schemata.

We will not discuss these features at length. However, it should be mentioned that the additional features make it possible to encode rather complex data structures. This enables us to map data-models of applications from whose information we want to share with others on an XML schema. From this point, we can encode our information in terms of an XML document and make it (together with the schema, which is also an XML document) available over the internet. This procedure has a big potential in the actual exchanging of data. However, the user must to commit to our data-model in order to make use of the information. We must point out that an XML schema defines the structure of data providing no information about the content or the potential use for others. Therefore, it lacks an important advantage of meta-information.We argued that XML is designed to provide an interchange format for weakly structured data by defining the underlying data-model in a schema and by using annotations, from the schema, in order to clarify the role of single statements. Two things are important in this claim from the information sharing point:

- XML is purely syntactic/structural in nature.
- XML describes data on the object level.

Consequently, we have to find other approaches if we want to describe information on the meta level and define its meaning. In order to fill this gap, the RDF standard has been proposed as a data model for representing meta-data about web pages and their content using an XML syntax. The basic model underlying RDF is very simple, every kind of information about a resource which may be a web page or an XML element is expressed in terms of a triple (resource, property, value). Thereby, the property is a two-placed relation that connects a resource to a certain value of that property. This value can be a simple data-type or a resource. Additionally, the value can be replaced by a variable representing a resource that is further described by nested triples making assertions about the properties of the resource that is represented by the variable. Furthermore, RDF allows multiple values for a single property. For this purpose, the model contains three builtin data types called collections, namely an unordered lists (bag), ordered lists (seq), and sets of alternatives (alt) providing some kind of an aggregation mechanism. A further requirement arising from the nature of the web is the need to avoid name-clashes that might occur when referring to different web-sites that use different RDF-models to annotate meta-data. RDF defines name-spaces for this purpose. Name-spaces are defined by referring to an URL that provides the names and connecting it to a source id that is then used to annotate each name in an RDF specification defining the origin of that particular name: source id:name

A standard syntax has been developed to express RDF-statements making it possible to identify the statements as meta-data, thereby providing a low level language for expressing the intended meaning of information in a machine processable way. The very simple model underlying ordinary RDF-descriptions leave a lot of freedom for describing meta-data in arbitrary ways. However, if people want to share this information, there has to be an agreement on a standard core of vocabulary in terms of modeling primitives that should be used to describe meta-data. RDF schemes (RDF/S) attempt to provide such a standard vocabulary. Looking closer at the modeling components, reveals that RDF/S actually borrows from frame systems well known from the area of knowledge representation. RDF/S provides a notion of concepts (class), slots (property), inheritance (SubclassOf, SubslotOf) and range restrictions (Constraint Property). Unfortunately, no well-defined semantics exist for these modeling primitives in the current state. Further, parts such as the re-identification mechanism are not well defined even on an informal level. Lastly, there is no reasoning support available, not even for property inheritance.

After introducing the W3C standards for information exchange and meta-data annotation we have to investigate their usefulness for information integration with reference to the three layers of integration. Firstly, we previously discovered that XML is only concerned with the issue of syntactic integration. However, XML defines structures as well, except there are no sophisticated mechanism for mapping different structures. Secondly, RDF is designed to provide some information on the semantic level, by enabling us to include meta-information in the description of a web-page. In the last section we mentioned, RDF in it's current state fails to really provide semantic descriptions. Rather it provides a common syntax and a basic vocabulary that can be used when describing this meta-data. Fortunately, the designers of RDF are aware that there is a strong need for an additional 'logical level' which defines a clear semantics for RDF-expressions and provides a basis for integration mechanisms.

Our conclusion about current web standards is that using XML and especially XML schemata is a suitable way of exchanging data with a well defined syntax and structure. Furthermore, simple RDF provides a uniform syntax for exchanging meta-information in a machine-readable format. However, in their current state neither XML nor RDF provides sufficient support for the integration of heterogeneous structures or different meanings of terms. There is a need for semantic modeling and reasoning about structure and meaning. Promising candidates for semantic modeling approaches can be found in the areas of knowledge representation, as well as, in the distributed databases community. We will discuss some of these approaches in the following sections.

Recently, the use of formal ontologies to support information systems has been discussed. The term 'Ontology' has been used in many ways and across different communities. If we want to motivate the use of ontologies for information integration we have to define what we mean when we refer to ontologies. In the following sections, we will introduce ontologies as an explication of some shared vocabulary or conceptualization of a specific subject matter. Further, we describe the way an ontology explicates concepts and their properties and finally argue for the benefit of this explication in many typical application scenarios. In general, each person has an individual view on the world and the things he/she has to deal with every day. However, there is a common basis of understanding in terms of the language we use to communicate with each other. Terms from natural language can therefore, be assumed to be a shared vocabulary relying on a (mostly) common understanding of certain concepts with very little variety. This common understanding relies on specific idea of how the world is organized. We often call these ideas a conceptualization of the world. These conceptualizations provide a terminology that can be used for communication between people. The example of our natural language demonstrates, that a conceptualization cannot be universally valid, but rather a limited number of persons committed to that particular conceptualization. This fact is reflected in the existence of different languages which differ even more (English and Japanese) or much less (German and Dutch). Confusion can become worse when we are considering terminologies developed for a special scientific or economic areas. In these cases, we often find situations where one term refers to different phenomena.

The use of the term 'ontology' in philosophy and in computer science serves as an example. The consequence of this confusion is, a separation into different groups, that share terminology and its conceptualization. These groups are then called information communities. The main problem with the use of a shared terminology according to a specific conceptualization of the world is that much information remains implicit. When a mathematician talks about a binomial normal he is referring to a wider scope than just the formula itself. Possibly, he will also consider its interpretation (the number of subsets of a certain size) and its potential uses (e. g. estimating the chance of winning in a lottery). Ontologies set out to overcome this problem of implicit and hidden knowledge by making the conceptualization of a domain (e. g. mathematics) explicit. This corresponds to one of the definitions of the term ontology most popular in computer science (Gruber, 1993): An ontology is an explicit specification of a conceptualization. An ontology is used to make assumptions about the meaning of a term available. It can also be viewed an explication, of the context a term, it is normally used in. Lenat (Lenat, 1998) for example, describes context in terms of twelve independent dimensions that have to be know in order to understand a piece of knowledge completely.

There are many different ways in which an ontology may explicate a conceptualization and the corresponding context knowledge. The possibilities range from a purely informal natural language description of a term corresponding to a glossary up, to a strictly formal approach, with the expressive power of full first order predicate logic or even beyond (e. g. Ontolingua (Gruber, 1991)). Jasper and Uschold (Jasper and Uschold, 1999) distinguish two ways in which the mechanisms for the specification of context knowledge by an ontology can be compared:

- Level of Formality: The specification of a conceptualization and its implicit context knowledge, can be done at different levels of formality. As already mentioned above, a glossary of terms can also be seen as an ontology, despite its purely informal character. A first step to gain more formality, is to describe a structure to be used for the description. A good example of this approach is the standard web annotation language XML (see section). The DTD is an ontology describing the terminology of a web page on a low level of formality. Unfortunately, the rather informal character of XML encourages its misuse. While the hierarchy of an XML specification was originally designed to describe a layout, it can also be exploited to represent sub-type hierarchies, (van Harmelen and Fensel, 1999) which may lead to confusion. Fortunately, this problem can be solved by assigning formal semantics to the structures used for the description of the ontology. An example of this is the conceptual modeling language CML (Schreiber et al.,

1994). CML offers primitives that describe a domain which can be given a formal semantic in terms of first order logic (Aben, 1993). However, a formalization is only available for the structural part of a specification. Assertions about terms and the description of dynamic knowledge is not formalized which offers total freedom for a description. On the other, there are specification languages which are completely formal. A prominent example is the Knowledge Interchange Format (KIF) (Genesereth and Fikes, 1992) which was designed to enable different knowledge-based systems to exchange knowledge. KIF has been used as a basis for the Ontolingua language (Gruber, 1991) which supplies formal semantics to that language as well.

- Extend of Explication: The other comparison criterion is, the extend of explication that is reached by the ontology. This criterion is strongly connected with the expressive power of the specification language used. We already mentioned DTD's which are mainly a simple hierarchy of terms. Furthermore, we can generalize this by saying that, the least expressive specification of an ontology consists of an organization of terms in a network using two-placed relations. The idea of this goes back to the use of semantic networks in the seventies. Many extensions of the basic idea examined have been proposed. One of the most influential ones was, the use of roles that could be filled out by entities showing a certain type (Brachman, 1977). This kind of value restriction can still be found in recent approaches. RDF schema descriptions (Brickley and Guha, 2000), which might become a new standard for the semantic descriptions of web-pages, are an example of this. An RDF schema contains class definitions with associated properties that can be restricted by socalled constraint-properties. However, default values and value range descriptions are not expressive enough to cover all possible conceptualizations. A more expressive power can be provided by allowing classes to be specified by logical formulas. These formulas can be restricted to a decidable subset of first order logic. This is the approach of description logics (Borgida and Patel-Schneider, 1994). Nevertheless, there are also approaches that allow for even more expressive descriptions. In Ontolingua for example, classes can be defined by arbitrary KIF-expressions. Beyond the expressiveness of full first-order predicate logic, there are also special purpose languages that have an extended expressiveness to cover specific needs of their application area. Examples are; specification languages for knowledge-based systems which often including variants of dynamic logic to describe system dynamics.

Ontologies are useful for many different applications, that can be classified into several areas. Each of these areas, has different requirements on the level of formality and the extend of explication provided by the ontology. We will review briefly common application areas, namely the support of communication processes, the specification of systems and information entities and the interoperability of computer systems. Information communities are useful because they ease communication and cooperation among members with the use of shared terminology with well defined meaning. On the other hand, the formalization of information communities makes communication between members from different information communities very difficult. Generally, because they do not agree on a common conceptualization. Although, they may use the shared vocabulary of natural language, most of the vocabulary used in their information communities is highly specialized and not shared with other communities. This situation demands for an explication and explanation of the use of terminology. Informal ontologies with a large extend of explication are a good choice to overcome these problems. While definitions have always played an important role in scientific literature, conceptual models of certain domains are rather new. Nowadays systems analysis and related fields like software engineering, rely on conceptual modeling to communicate structure and details of a problem domain as well as the proposed solution between domain experts and engineers. Prominent examples of ontologies used for communication are Entity-Relationship diagrams and Object-oriented Modeling languages such as UML. ER-diagrams as well as UML are not only used for communication, they also serve as building plans for data and systems guiding the process of building (engineering) the system. The use of ontologies for the description of information and systems has many benefits. The ontology can be used to identify requirements as well as inconsistencies in a chosen design. Further, it can help to acquire or search for available information. Once a systems component has been implemented, its specification can be used for maintenance and extension purposes. Another very challenging application of ontology-based specification is the reuse of existing software. In this case, the specifying ontology serves as a basis to decide if an existing component matches the requirements of a given task. Depending on the purpose of the specification, ontologies of different formal strength and expressiveness are to be utilized. While the process of communication design decisions and the acquisition of additional information normally benefit from rather informal and expressive ontology representations (often graphical), the directed search for information needs a rather strict specification with a limited vocabulary to limit the computational effort. At

the moment, the support of semiautomatic software reuse seems to be one of the most challenging applications of ontologies, because it requires expressive ontologies with a high level of formal strength.

The previously discussed considerations might provoke the impression that the benefits of ontologies are limited to systems analysis and design. However, an important application area of ontologies is the integration of existing systems. The ability to exchange information at run time, also known as interoperability, is an valid and important topic. The attempt to provide interoperability suffers from problems similar to those associated with the communication amongst different information communities. The important difference being the actors are not people able to perform abstraction and common sense reasoning about the meaning of terms, but machines. In order to enable machines to understand each other, we also have to explicate the context of each system on a much higher level of formality.

Ontologies are often used as Inter-Linguas in order to provide interoperability: They serve as a common format for data interchange. Each system that wants to inter-operate with other systems has to transfer its data information into this common framework. Interoperability is achieved by explicitly considering contextual knowledge in the translation process.

For an appropriate support of an integration of heterogeneous information sources an explicit description of semantics (i. e. an ontology) of each source is required. In principle, there are three ways how ontologies can be applied:

- a centralized approach, where each source is related to one common domain ontology,
- a decentralized approach, where every source is related to its own ontology, or
- a hybrid approach, where every source is related to its own ontology but the vocabulary of these ontologies stem from a common domain ontology

A common domain ontology describes the semantics of the domain in the SIMS mediator (Arens et al., 1996). In the global domain model of these approaches all terms of a domain are arranged in a complex structure. Each information source is related to the terms of the global ontology (e. g. with articulation axioms (Collet et al., 1991)). However, the scalability of such a fixed and static common domain model is low (Mitra et al., 1999), because the kind of information sources which can be integrated in the future is limited. In OBSERVER (Mena et al., 1996) and SKC (Mitra et al., 1999) it is assumed, that a predefined ontology for each information source exists. Consequently, new information sources can easily be added and removed. But the comparison of the heterogeneous ontologies leads to many homonym, synonym, etc. problems, because the ontologies use their own vocabulary. In SKC (Mitra et al., 1999) the ontology of each source is described by graphs. Graph transformation rules are used to transport information from one ontology into another ontology (Mitra et al., 2000). These rules can only solve the schematic heterogeneities between the ontologies. In MESA (Wache et al., 1999) the third hybrid approach is used. Each source is related to its source ontology. In order to make the source ontologies comparable, a common global vocabulary is used, organized in a common domain ontology. This hybrid approach provides the biggest flexibility because new sources can easily be integrated and, in contrast to the decentralized approach, the source ontologies remain comparable. In the next section we will describe how ontologies can help to solve heterogeneity problems.

3 Terms and definitions

This section lists the applicable documents

Ref	Document Title	Issue/Date
TA	pSHIELD Technical Annex	1
M0.1	Formalized Conceptual Models of the Key pSHIELD Concepts	1
M0.2	Proposal for the aggregation of SPD metrics during composition	1
D2.1.1	pSHIELD Systems requirements and specifications	1
D2.2.1	pSHIELD metrics definition	1
D5.1	pSHIELD Semantic Models Prototype	1

3.1 SPD Dictionary

A comprehensive dictionary of the SPD concepts is provided by the project glossary, that is a relevant and essential step in the ontology building process (refer to the section “ Building the glossary.”).

4 Semantic Technologies

4.1 Ontologies and Semantic Web Software Tools

An ontology is a formal specification of a conceptualisation [27], where concepts are distinguished by axioms and definitions describing an area of knowledge [28].

The World Wide Web represents a huge repository of information which can be retrieved and used. Unfortunately, information is represented with no meaning associated, since the meaning of retrieved information can be (re-)established only in the process of interpreting the information by humans. As a result, information scattered throughout the current (and traditional) version of the web is almost totally useless for software, non-human users (machine agents).

In attempt to respond to this situation, the term “Semantic Web” was coined by Tim Berners-Lee and his colleagues [24] referring to a “web for machines” as opposed to a web to be read by humans. In their understanding, “*The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation*.”

The Semantic Web has a developing layered architecture which is often represented with a diagram in Figure 4.1.

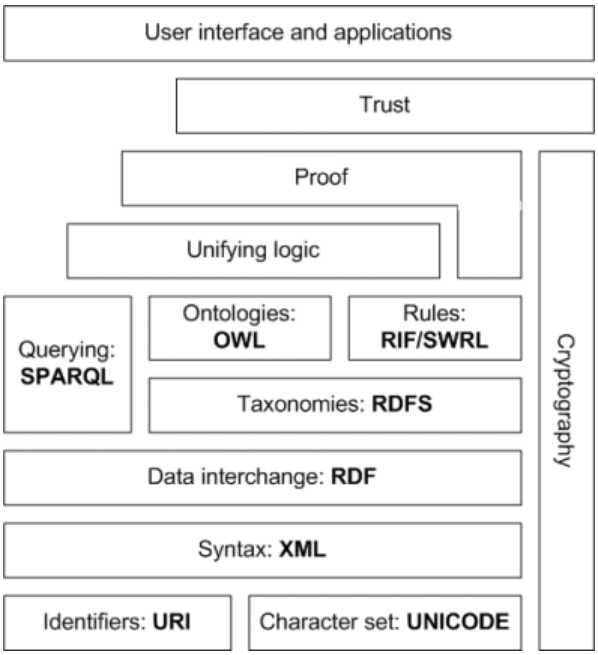

Figure 4.1 Semantic web layers

Unicode and URIs: The concept of Unicode is the standard used for computer character representation. This provides a basis for representing characters used in most of the languages in the world. URIs on the other hand stand for Uniform Resource Identifiers and provide the standards for identifying and locating resources. A resource can be anything that has an identity such as a web site, a document, an image and a person. Protocols that exist on this layer are TCP/IP, SSL and HTTP, as the protocols for data transmission. They are built on top of more basic communication layers. With these protocols one can transmit the web pages over the Internet. At this level one does not deal with syntax or the semantics of the documents.

XML: XML and its related standards, such as Namespaces, and Schemas, form a common medium for structuring data on the Web. They do not communicate the meaning of the data, but nonetheless are well

established within the Web already. XML is a language used to represent data in a structural way. It describes what is in the document, not what the documents looks like, while XML Schema provides grammars for legal XML documents. On the other hand, Namespaces allows the combination of different vocabularies.

Resource Description Framework (RDF): RDF is the first proper layer of the Semantic Web. RDF is a simple metadata representation framework. By using URIs it Identifies Web based resources, using a graph model that describes relationships between resources. RDF essentially uses XML syntax but has support to express semantics. One needs to use RDF for integrating and exchanging information in a meaningful way on the web. While XML documents are exchanged over protocols such as TCP/IP, HTTP and SSL, RDF documents are built using XML.

RDF Schema: a simple type modelling language for describing the classes of resources along with their properties in the basic RDF model. It provides a simple reasoning framework for inferring the types of resources.

Ontologies: A richer language for providing more complex constraints on the types of resources and their properties. Ontology is considered the backbone for the semantic web architecture provides a machine-processable semantics and a sharable domain which can facilitate communication between people and different applications.

Logic and Proof: An (automatic) reasoning system provided on top of the ontology structure to make new inferences. Thus, using such a system, a software agent can make deductions as to whether a particular resource satisfies its requirements or not (and vice versa). The Logic layer is placed above the ontology layer. It is supposed that information will be extracted from the web according to this logic. Proof is the layer placed above the Logic layer. It is assumed to be a language used in a manner that describes for agents why they should believe the results.

Trust: The purpose of final layer of the layered architecture is to know trustworthiness of the information by asking questions in Semantic Web. This assures the quality of that information. The Semantic Web uses in reasoning while searching towards the exactness on web data for the search query hence we can say that Semantic Web helps the Web machine process better. Trust depends on the source of information as well as the policies available on the information source which can prevent unwanted applications or user from access to these sources.

The Semantic Web is the opportunity for providing, finding and processing information via the Internet with the help of (machine) agents which are capable of dealing with the semantics of the information. The idea is to transform information into something meaningful to actors who seek to enhance their knowledge in order to satisfy a specific concern or accomplish a specific task related to their particular context.

The vision of the Semantic Web is based on the employment of semantic technologies that allow the meaning of information and the meaning of associations between information to be known and processed at execution time. To fulfill the promises and enable semantic technologies to work, there must be a *knowledge model* (of some part) of the world that is used to provide meaning to information to be processed within an application. The knowledge model has the form of a semantic model which differs from other kind of models [25]:

- *Connections*: meaning is represented through *connectivity*. The meaning of terms, or concepts, in the model is established by the way they connect to each other.
- *Multiple views*: a semantic model expresses multiple viewpoints and several interconnected models could be used to represent different aspects.
- *Sharing*: semantic models represent knowledge about the world in which systems operate and are *shared* across applications.
- *Reasoning capability*: use of a model is often referred to as “reasoning over the model”. The reasoning can range from a very simple process of graph search to intricate inferencing.

Although the role of a semantic model can be played by a simple taxonomy, nowadays use of semantically richer ontologies (*ontological models*) dominates.

Although most common definition states that “An ontology is a specification of conceptualisation”, a more detailed definition can make things a bit clearer: for our purpose, and a practical point of view, *an ontology is a network of connections defining explicit relationships (named and differentiated) between concepts*.

New knowledge can be *derived* by examining the connections between concepts. Simple ontologies are just networks of connections, richer ontologies include rules and constraints governing these connections. The semantic web is not so much a technology as an infrastructure, enabling the creation of meaning through standards, mark-up languages, and related processing tools. To represent ontologies in a formal way, several languages can be used. The Semantic Web principles are implemented in the layers of Web technologies and standards. The most common ontology languages are briefly described below (all the presented languages are supervised by the World Wide Web Consortium [26].

At the beginning, the idea of the semantic web tried just to enhance the current version of the web. It started out with a document oriented approach. The basic idea was to make web pages identifiable by computers as information resources carrying not only information (readable only by humans) but the meaning of this information as well. The meaning was added by *annotating* these pages with semantic mark-up. Ontologies here define a shared conceptualization of the application domain at hand and provide the basis for defining metadata, that have a precisely defined semantics, and that are therefore machine-processable. The idea of semantically annotated web pages with machine-interpretable description of their content aimed at automated processes of searching and accessing pages enabling human users to better utilise information stored on the web. In addition to human users, the semantic web enables the participation of non-human users as well. These machine agents do not need to deal with whole web pages. Instead of this, they exchange chunks of data with each other. Although they can communicate using different protocols, technology of web services has become a dominant way of communication with and using services of applications in the web environment.

Formerly, the problem of *interoperability* of different agents was tackled by translation technologies, most commonly by field to field mapping. The semantic web enables agents to exchange chunks of data with meaning associated to the data using semantic technologies. Advanced applications can use ontologies to relate the information to a semantic model of a given domain. In this way semantic technologies offer a new way to integrate different applications. Nowadays, the field of semantic interoperability is the most addressed problem connected with the idea of the semantic web.

4.1.1 Description Logics

Description Logics (DLs) [29] are a family of general-purpose knowledge representation formalisms based on subsets of first-order predicate logic, where reasoning amounts to the verification of logical consequences. Many ontology languages (e.g., OIL, DAML+OIL and OWL) are based on expressive description logics, and in particular on the SHIQ family of description logics.

4.1.1.1 Reasoners

DIG stands for the DL Implementation Group, a group devoted to the implementation of description logic systems. Their key contribution is the development of the DIG interface, which provides a common interface for DL reasoners thus allowing a variety of ontology tools, such as Protégé and OilEd, to make use of reasoners. We review some reasoners in what follows.

Because writing a Description Logic reasoner is a non-trival task, and highly optimised 3rd party Description Logic reasoners have been developed Current complete and efficient DL systems for very expressive DLs are FaCT, [30] RACER [31] and Pellet. [32]

4.1.1.1.1 FaCT++[1]

FaCT (Fast Classification of Terminologies) is a highly optimized DL classifier written in Common Lisp for terminologies expressed in the SHIQ logic [30]. It supports reasoning with knowledge bases containing CGIs, but cannot deal with individuals or concrete datatype domains. Furthermore, FaCT does not support multiple T-boxes and does not provide mechanisms for removing concept definitions. With FaCT++ a reimplementation of FaCT in C++ is available [31]. It implements the logic SHIF(D−) and is based on a new internal architecture that introduces new optimizations. While FaCT++ implements A-box reasoning, the support for concrete domains is restricted to Integers and Strings (therefore D−).

FaCT ++ is the evolution of the OilEd companion inference mechanism, FaCT (Fast Classification of Terminologies). Programmed in C++ for higher efficiency, this reasoner is released under a GNU public license and is available for download both as a binary file and as source code. FaCT++ supports SHOIQ with simple datatypes (i.e., for OWL-DL with qualifying cardinality restrictions). It implements a table based decision procedure for general TBoxes (subsumption, satisfiability, classification) and incomplete support of ABoxes (retrieval) [34].

4.1.1.1.2 RACER[2]

RACER is another highly optimized tableau calculus reasoner implemented in Common Lisp**Errore. L'origine riferimento non è stata trovata.**. It supports all optimization techniques that are incorporated into FaCT, plus some additional optimizations dealing with number restrictions and A-box reasoning, which are dynamically selected based on a static analysis of the given knowledge base and query. Like FaCT it implements SHIQ(D−), but additionally supports A-box reasoning services and retraction of A-Box assertions, as well as multiple T- and A-boxes. Furthermore, RACER provides facilities for algebraic reasoning, including concrete domains for dealing with restrictions over integers, linear polynomial equations over reals, nonlinear multivariate polynomial equations over complex numbers and equalities and inequalities of strings. However, Racer does not support complete reasoning with respect to nominals. Transforming individuals in the enumerated classes to disjoint atomic concepts, as is done by RACER, only approximates nominals.

RacerPro was the first OWL reasoner available. Its first (freeware) version is known as Racer and was released in 2002. Today's commercial 1.9 version provides many features over the original Racer engine, including: an expressive query language nRQL; support for OWL DL (almost completely), not employing the unique name assumption, as required by OWL DL; and a rule engine that provides for efficient and application-controlled ABox augmentation based on rules. A model-checking facility is integrated with nRQL as well.

4.1.1.1.3 Pellet[3]

Pellet is a sound and complete tableau reasoner for SHIN(D) as well as SHON(D) and a sound but incomplete reasoner for SHION(D) [32]. Additionally, Pellet provides sound and complete reasoning for nominals by using the algorithms developed for SHOQ(D) [35].
It is implemented in Java and incorporates a number of special features. Pellet supports ontology analysis and repair, datatype reasoning for built-in primitive XML Schema datatypes, conjunctive A-box query and a direct implementation of entailment checking.

Pellet is an open source Java OWL DL inference mechanism that can be used with the JENA API. Pellet implements tableau algorithms that manipulate description logic expressions. It can be used online by submitting a valid ontology URI. Pellet is part of a Semantic Web tool suite made available by the MindSwap Group at the University of Maryland. Besides the inference mechanism, MindSwap also makes an ontology editor (SWOOP) and an OWL photo annotation tool (PhotoStuff) publicly available.

[1] http://owl.man.ac.uk/factplusplus/
[2] http://www.racer-systems.com/
[3] http://www.mindswap.org/2003/pellet/

4.1.2 Web Ontology Language

It has been predicted that ontologies will play a pivotal role in the Semantic Web. The World Wide Web Consortium (W3C) sketched the vision of a 'second generation' Web in which Web resources will be more readily accessible to automated processes [36].

To overcome the limited expressiveness of RDFS, a more expressive Web Ontology Language (OWL) [39] has been defined by the W3C based on the former proposal DAML + OIL, [37] which itself is the result of merging the frame-based American proposal DAML-ONT with the DL-based European language OIL (Figure 4.2).

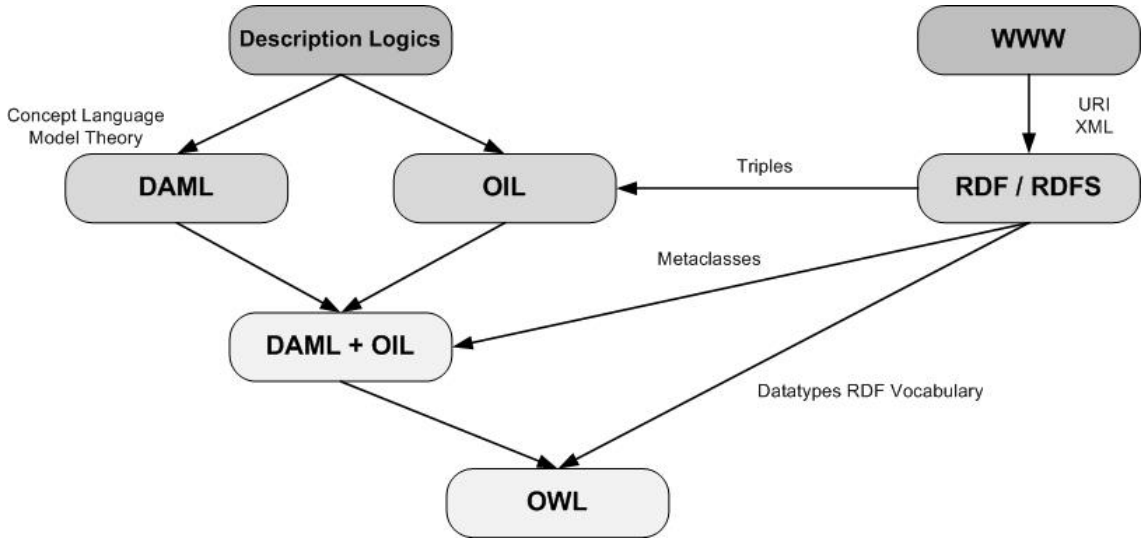

Figure 4.2 OWL influences

To fulfil the aim for a standardized and broadly accepted ontology language for the Semantic Web, the language needed a well defined syntax and semantics with sufficient expressive power and efficient reasoning support.

Formal semantics and reasoning support is usually provided by mapping the ontology language to a known logical formalism. In the case of OWL, expressive Description Logics (DLs) have been selected as the semantic foundation, enabling efficient reasoning by the application of existing DL reasoners.
OWL is layered on top of XML, RDF and RDFS [40]. Syntactically, OWL can be seen as a specific dialect of RDFS (for example, OWL individuals are defined using RDF descriptions).
Each valid OWL document has to be a valid RDFS document (but not vice versa). However, this XML/RDF syntax of OWL does not provide a very readable syntax. Therefore, the more readable OWL abstract syntax has been defined [38].

4.1.2.1 Sublanguages

OWL is declined into three increasingly expressive sublanguages (Figure 4.3), which are respectively OWL Lite, OWL DL and OWL Full [52]. OWL Full is the most expressive language, which allows all of the OWL constructs discussed in the previous section and covers all of RDFS. Because of the lack of a clear separation between classes and individuals in its sublanguage RDFS, OWL Full comprises some higher-order features that result in the undecidability of key inference problems [53]. OWL DL, a decidable fragment of first-order logic, uses exactly the same vocabulary as OWL Full, but includes some additional syntactic restrictions. With these restrictions, OWL DL corresponds to the Description Logic (DL) SHOIN(D), decidable in NExpTime. OWL Lite is a fraction of OWL DL aiming to provide a useful subset of language features that are easier to present to naive users. The expressivity of OWL Lite is about that of the DL language SHIF(D) with an ExpTime worst-case complexity. In contrast to OWL DL, OWL Lite excludes syntax constructs for unions, complements, and individuals in descriptions or class axioms. In addition, it limits cardinalities to either 0 or 1 and nested descriptions to concept identifiers. However, these restrictions come with relatively little loss in expressive power. With help of indirection and syntactical tricks, all of OWL DL can be captured in OWL Lite, except those descriptions containing either individuals or cardinalities greater than 1.28

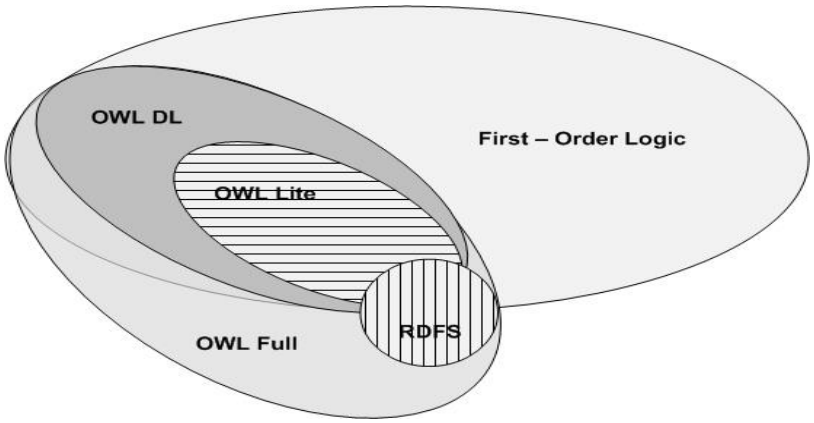

Figure 4.3 OWL sublanguages

OWL is defined as a vocabulary, just as are RDF and RDF Schema, but it has a richer semantics. Hence, an ontology in OWL is a collection of RDF triples, which uses such vocabulary. The definition of OWL is organized as three increasingly expressive sublanguages:

- OWL Lite offers hierarchies of classes and properties, and simple constraints with enough expressive power to model thesauri and simple ontologies.
- OWL DL increases expressiveness and yet retains decidability of the classification. OWL DL offers all OWL constructs, under certain limitations
- OWL Full is the complete language, without limitations, but it ignores decidability issues.

OWL Full can be viewed as an extension of RDF, whereas OWL Lite and OWL DL are extensions of restricted forms of RDF. In more detail, we have that: every OWL (Full, DL, or Lite) document is an RDF/XML document; every RDF/XML document is an OWL Full document; not all RDF/XML documents are OWL DL (or OWL Lite) documents.

RDF is more expressive than OWL Lite and OWL DL exactly because the RDF data model imposes no limitations on how resources (URIrefs) can be related to each other. For example, in RDF, a class can be an instance of another class, whereas OWL DL and OWL Lite require that the sets of URIrefs that denote classes, properties, and individuals be mutually disjoint. Therefore, care must be taken when translating from RDF to OWL Lite or OWL DL.

Since nominals are non-standard with respect to traditional DL research, practical complete algorithms for the full OWL DL language were unknown for a long time before the recent result of Horrocks and Sattler [54]. However, this work still leaves the question open how to implement a sound, complete and high-performance reasoning system supporting the XML Schema Datatypes of OWL that goes beyond the standard concrete domains (Integer, Real, String) of traditional DL languages. The logic containments are shown in Figure 4.4.

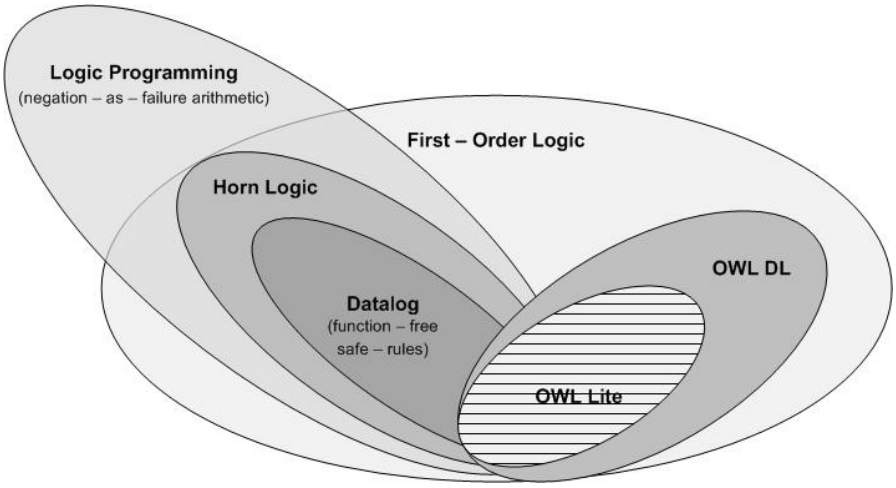

Figure 4.4 Logic containments

4.1.3 Development Environments

Building an ontology for a fairly large domain can easily become a huge and complex task. Such big undertakings cannot be a success without the existence of a proper methodology, which guides the development as with any software engineering artefact. Ontological Engineering refers to the set of activities that concern the ontology development process, the ontology life cycle, the methods and methodologies for building ontologies, and the tool suites and languages that support them.

Because constructing ontologies is a time-consuming task, software tools have been build to support developers on most ontology engineering tasks. Current ontology development environments not only support the knowledge engineer in the creation of ontologies by providing simple editing environments, but also offer graphical ontology navigation, validation and debugging support, version management and solutions for linking to ontologies from external sources. In the following sections, we introduce the three most innovative development environments for OWL ontologies, namely Protégé [41], SWOOP [42] and OntoTrack [43].

4.1.3.1 Protégé[4]

The Protégé system is an environment for knowledge-based systems development that has been evolving for over a decade [41]. Protégé began as a small application designed for a medical domain, but has developed into a much more general-purpose set of knowledge-engineering tools, which recently also included enhanced support for OWL [44], build on top of HP's Jena libraries [45]. The current version, Protégé 3.1, can be run on a variety of platforms, supports an enhanced graphical user interface, interacts with standard storage formats such as relational databases, XML and RDF, and has been used by numerous individuals and research groups.
Protégé can be integrated with several reasoning engines (such as RACER [31]) to support the working knowledge engineer. The system is freely available from, and actively developed and supported by, Stanford SMI, USA.

4.1.3.2 Swoop

SWOOP (Semantic Web Ontology Overview and Perusal) [42] is a Java-based graphical ontology editor that employs a web-browser metaphor for its design and usage. It provides a hyperlink-based navigation across ontological entities, history buttons for traversal and bookmarks that can be saved for later reference. As well as the RDF/XML syntax of OWL, it also supports the official abstract syntax of OWL DL [38]. Like other environments, SWOOP also comes with multiple ontology support. Validation support and species validation is realized by a tight coupling with the Pellet reasoner [32]. Among SWOOP's advanced features are an extensive version management support in which change sets can be attached to collaborative annotation messages created using an Annotea plugin [42], the possibility of partitioning ontologies automatically[46] by transforming them into separate E-connected ontologies [47] and to run sound and complete conjunctive A-box queries written in RDQL [48]. More recently, a debug mode has been added that supports glass-box and black-box debugging by exposing the internal workflow of its DL-reasoner Pellet in a meaningful and readable manner to help users in understanding the cause of the detected inconsistencies [49][50].

4.1.3.3 OntoTrack

The ontology authoring tool OntoTrack [43] overcomes the drawbacks concerning search and navigation speed of other editors that use two functionally disjunctive interfaces for either editing or browsing, by integrating both 'in one view'. The system combines a sophisticated graphical layout with mouse-enabled editing features optimized for efficient navigation and manipulation of large ontologies, and features an explanation component [51]. Its engine provides sophisticated layout algorithms enabling animated expansion and de-expansion of class descendants, zooming, thumbnail views and panning. Its most innovative feature is the instant feedback functionality. By synchronizing every single editing step with an external reasoner, relevant modelling consequences are instantaneously detected and visualized. Currently, OntoTrack is able to handle most of OWL Lite excluding individuals, datatype properties and annotation properties.

[4] http://protege.stanford.edu/

4.1.3.4 JENA API[5]

JENA is a Java API whose goal is to support the development of Semantic Web applications. JENA is open source, and it was originally developed at the HP Labs Semantic Web Programme. It supports RDF, RDF Schema, and OWL, and includes an inference mechanism.
The JENA API transforms an ontology into an object-oriented abstract data model, and allows its primitives (classes and relationships) to be treated as objects thereafter. Therefore, JENA allows ontology elements to be manipulated by object oriented programming languages.

[5] http://jena.sourceforge.net

4.2 Ontology representations

A number of possible languages can be used to represent ontologies; many of them evolved from creation of ontology construction methodologies. The Open Knowledge Base Connectivity (OKBC) [1] model and languages like “Knowledge Interchange Format (KIF)” [2] are examples that have become the bases of other ontology languages.

Several languages use frame logic which is basically an object-oriented approach defining frames and attributes (classes and properties). There are also several languages based on description logic, e.g. Loom [3], DAML+OIL [4], or later evolved Web Ontology Language (OWL) [5] standard.

Representation languages can be divided in terms of different abstraction levels used to structure the representation itself:

a) *Extensional level*: the model is formulated by specifying every object from the domain.

b) *Intensional level*: objects are defined by means of (necessary and sufficient) conditions for belonging to the domain.

c) *Meta-level*: concepts from intensional level are abstracted, higher level concepts are specified, and previous concepts are seen as instances of new meta-concepts.

Some issues emerge from analysis of ontology representation, concerning the scope and modality of context expression: these criteria consider the basic formal nature of languages and that various languages deal with the representation of incomplete information in different way

1) We can express:

   a) *Class and relations*: languages aiming at representing objects, classes and relations.

   b) *Actions and processes*: languages that provide specialized representation structures for describing dynamic characteristics of the domain, such as actions, processes, and workflows (they usually can represent static aspects of domain too, but only in elementary level).

   c) *Everything*: languages that may be used for any kind of contexts and applications.

2) The context can be expressed in the following ways:

   a) *Programming languages*: allow representation and manipulation of data in several ways and according to various paradigms, leading to a cleaner separation between data structures and algorithms that handle them. Object oriented paradigm is preferred in recent years. This approach is generally associated with a number of concepts, such as complex objects, object identity, methods, encapsulation, typing and inheritance. Example can be language F-logic [6], logical formalism that tries to capture the features of object-oriented approaches to computation and data representation. F-Logic forms the core of systems such as Ontobroker [7].

   b) *Conceptual and semantic database models:* semantic (or conceptual) models were introduced as schema design tools. Examples of proposed semantic data models are ER and Extended ER data model, FDM (Functional data model), SDM (Semantic Data Model). Semantic models provide more powerful abstractions for the specification of databases.

   c) *Information system / software formalisms*: here belong different formalisms for information system design, especially in object-oriented design. Most widely used formalism is Unified Modelling Language (UML). UML was designed for human-to-human communication of models for building systems in object-oriented programming languages. Over the years its use has been extended to a variety of different aims, including the design of databases schemas, XML document schemas, and knowledge models.

   d) *Logic-based*: very important class of languages is based on logic. Such languages express a domain-ontology in terms of the classes of objects that are of interest in the domain, as well as the relevant relationships holding among such classes. These languages have a formal well-defined semantics. Three different types of logic-based languages exist – languages based on

first-order predicate logic (e.g. KIF [2]), languages based on description logics (e.g. OWL [5]), and process-action specificationlanguages (e.g. PSL [8]).

e) *Frame-based*: frame is a data structure that provides a representation of an object or a class of objects or a general concept or predicate. Some systems define only a single type of frame, other have two or more types, such as class frames and instance frames. The *slots* of a frame describe attributes of represented concept. They may also have other components in addition to the slot name, value and value restrictions, for instance the name of a procedure than can be used to compute the value of the slot – facets. Frames are usually organized into taxonomies. Through taxonomic relations, classes may be described as specializations of more generic classes with inheritance capability. Frame-based ontology languages were often used in many knowledge-based applications, like Ontolingua [9], OCML [10], OKBC [11] or XOL [12].

f) *Graph-based*: formalisms based on various kinds of graph based or graph-oriented notations. Semantic networks [13] and conceptual graphs [14] originated from the Artificial Intelligence community. OML/CKML (Conceptual Knowledge Markup Language) [15] is a framework and markup language for knowledge and ontology representation based on conceptual graphs. Topic Maps [16] are recent proposal originated from the XML community.

g) *XML-related formalisms*: XML [17] is a tag-based language for describing tree structures with a linear syntax and it is a standard language for exchange of information in the Web. Given the popularity of XML in exchange of information, XML-related languages have been considered as suitable for ontology representation. Important languages are based on Resource Description Framework (RDF) [18]. These provide a foundation for processing metadata about documents.

3) We can interpret expression in the following ways:

a) *Single model*: ontology should be interpreted in such a way that only one model of the corresponding logical theory is a good interpretation of the formal description.

b) *Several models*: ontology should be interpreted as specifying what we know about the domain with the reservation that the amount of knowledge we have about the domain can be limited (e.g. first-order logic based languages).

4.3 Semantic Web Ontology languages

Description Logics (DLs) are a family of logic-based knowledge representation formalisms designed to represent and reason about the knowledge of an application domain in a structured and well understood way. The basic notions in DLs are *concepts and roles*, which denote sets of objects and binary relations, respectively. Most of today's semantic web ontology languages are DL-based. Also many of them are XML-related, or they possible XML notation. Several ontology languages have been designed for use in the web. Among them, the most important are OIL [19], DAML-ONT [20] and DAML+OIL [21]. More recently, a new language, OWL [5], is being developed by the World Wide Web Consortium (W3C) Web Ontology Working Group, which had to maintain as much compatibility as possible with pre-existing languages and is intended to be proposed as the standard Semantic Web ontology language. The idea of the semantic Web is to annotate web pages with machine-interpretable description of their content. In such a context, ontologies are expected to help automated processes to access information, providing structured vocabularies that explicate the relationships between different terms.

Extensible Markup Language (XML) [17] was widely accepted and used as a convenient information representation and exchange format. *XML itself don't carry semantics*, but is serves as the base syntax for the leading ontology languages that we shall survey. Later additions like XML-DTD (Document Type Definition) and XML-Schema, added some syntactic rules like enumerations, cardinality constrains, and data types, but still lacked even simple semantics like inheritance. The purpose of XML Schema is therefore to declare a set of constraints that an XML document has to satisfy in order to be validated. With respect to DTD, however, XML Schema provides a considerable improvement, as the possibility to define much more elaborated constraints on how different part of an XML document fit together, more sophisticated nesting rules, data-typing. Moreover, XML-Schema expresses shared vocabularies and allows machines to carry out rules made by people. Among a large number of other rather complicated features.

Resource Description Framework (RDF) [18] is a standard way for defining of simple descriptions. RDF is for semantics - a clear set of rules for providing simple descriptive information. RDF enforces a strict notation for the representation of information, based on resources and relations between them. As referred to in its name, RDF strength is in its descriptive capabilities, but is still lacks some important features required in an ontology language such as inferences for example. However, ontology languages built on top of RDF as a representation and description format. The RDF data model provides three object types: *resources*, *properties*, and *statements*. Resource may be either entire Web page, a part of it, a whole collection of pages or an object that is not directly accessible via the Web, property is a specific aspect, characteristic attribute, or relation used to describe a resource, statement is a triple consisting of two nodes and a connecting edge. These basic elements are all kinds of RDF resources. According to the latter description, a subject is a resource that can be described by some property. The predicate defines the type of property that is being attributed. Finally, the object is the value of the property associated with the subject.

RDF Schema (RDFS) [22] enriches the basic RDF model, by providing a vocabulary for RDF, which is assumed to have certain semantics. Predefined properties can be used to model instance of and subclass of relationships as well as domain restrictions and range restrictions of attributes. Indeed, the RDF schema provides modelling primitives that can be used to capture basic semantics in a domain neutral way. That is, RDFS specifies metadata that is applicable to the entities and their properties in all domains. The metadata then serves as a standard model by which RDF tools can operate on specific domain models, since the RDFS meta-model elements will have a fixed semantics in all domain models. RDFS provides simple but powerful modelling primitives for structuring domain knowledge into classes and sub classes, properties and sub properties, and can impose restrictions on the domain and range of properties, and defines the semantics of containers.

Web Ontology Language (OWL) The next layer in the Semantic Web architecture is Web Ontology Language (OWL) [5], a language for Web ontologies definition and instantiation. OWL enhances RDF vocabulary for describing properties and classes: relations between classes (e.g. subclasses), cardinality, equality, richer typing of properties, characteristics of properties (e.g. symmetry) and instances. OWL is the W3C recommendation for ontology definition, but other standards also support similar characteristics (DAML+OIL). Several tools support modelling with OWL and DAML+OIL. The OWL language also provides three increasingly expressive sublanguages: *OWL Lite*, *OWL DL*, and *OWL Full*, each offers a different level of expressiveness at the trade-off for simplicity, thus offering a suitable sub language parts available for use according to expressibility needs. There also exists OWL based enhancement to web

services oriented languages, aiming to handle semantic descriptions of such services. OWL-S [23] is framework for containing and sharing ontological description of the capabilities and characteristics of a Web service.

An **OWL-S** specification includes three sub-ontologies that define essential types of knowledge about a service – *service profile* describes the outlining interface and characteristics of the service, a *process profile* defines the control flow of the service and the *service grounding* provides mapping with communication-level protocols. OWL-S has similar characteristics with a number of related protocols. The popularity of OWL-S in the Semantic Web community, as Web services description language, adds to the attractiveness of the language.

A deeper analysis of these technologies is performed in the following paragraph.

4.4 Trends in Semantic Web Services Technologies

Targeting new SPD middleware and overlay funtionalites prototypes related to Semantic Web Srvices (SWS) is requiring a deep understing of the trends in SWS technologies. Web services [8a] have added a new level of functionality to the current Web by taking a first step towards seamless integration of distributed software components using web standards.

Problem statement: The Web service technologies around SOAP (XML Protocol Working Group, 2003), WSDL and UDDI operate at a syntactic level and, therefore, although they support interoperability (i.e., interoperability between the many diverse application development platforms that exist today) through common standards, they still require human interaction to a large extent: The human programmer has to manually search for appropriate Web services in order to combine them in a useful manner, which limits scalability and greatly curtails the added economic value of envisioned with the advent of Web services.

Recent research aimed at making web content more machine-processable, usually subsumed under the common term Semantic Web [9a] are gaining momentum also, in particular in the context of Web services usage. **The semantic markup shall be exploited to automate the tasks of Web service discovery, composition and invocation, thus enabling seamless interoperation between them while keeping human intervention to a minimum** [10a].

The convergence of semantic Web with service oriented computing is manifested by SWS technology. It addresses the major challenge of automated, interoperable and meaningful coordination of Web Services to be carried out by intelligent software agents.

SWS technologies were proposed in order to pursue the vision of the semantic Web presented in [9a] whereby intelligent agents would be able to exploit semantic descriptions in order to carry out complex tasks on behalf of humans [11a]. This early work on SWS was the meeting point between semantic Web, Agents, and Web services technologies. Gradually, however, research focussed more prominently on combining Web services with semantic Web technologies in order to better support the discovery, composition, and execution of Web services, leaving aspects such as systems' autonomy more typical of agent--based systems somewhat aside [12a].

In one of the seminal papers on SWS, McIlraith et al. proposed the usage of Semantic Web technologies in order to markup Web services to make them machine-interpretable and accordingly facilitate automatic Web service discovery, execution, composition, and interoperability [11a].

Here, we will focus on the trends around SWS and research and development, as well as standardization efforts on semantic Web services. Activities related to semantic Web services have involved developing conceptual models or ontologies, algorithms and engines that could support machines in semi-automatically or automatically discovering, selecting, composing, orchestrating, mediating and executing services. We will provide an overview of the area after nearly a decade of research.We will present the main principles and conceptual models proposed thus far including OWL-S, WSMO [12a].

These four specifications related to Semantic Web Services were submitted to the World Wide Web Consortium (W3C) in 2004–2005.

1. OWL-S [23a]
2. Semantic Web Services Framework (SWSF) [26a]
3. Web Service Semantics (WSDL-S) [25a]
4. Web Service Modeling Ontology (WSMO) [24a]

For the most part, it is unclear now whether these specifications compete against, complement, or supersede one another. There are ongoing integration efforts. For instance, WSMO uses WSDL-S as its Service Grounding mechanism. Also, there is a W3C Semantic Web Services Interest Group that is chartered with discussing the integration of this work [111a]. A clear understanding of the relationship among these four submissions and the types of problems in which they can be applied is necessary for the adoption of Semantic Web Services in general.

Investigating SWS technologies and making comaparison between them we do not believe it is a matter of deciding who the winner is, but rather how these different efforts can collaborate in producing a

standard that encompasses the benefits of each or provides clear instructions for use and integration into semantic solutions. Semantic Web services are an ongoing, dynamic research field. Nevertheless, they have the potential of providing a foundational pillar for the next generation of Web technology and they fit in our research effort to investigate how these technologies will enhance SPD middleware and overlay functionalities, as emphasized by the submission of WSMO and OWL-S to the W3C as starting points for upcoming standardization efforts.

The Semantic Web [54a] approach refers to the idea of making the overwhelming amount of data on the Web machine-processable. This shall be achieved by annotating web content with consensual formalizations of the knowledge published, often referred to under the common term of *ontologies*. Language proposals for describing ontologies include the Resource Description Framework (RDF) [55a], RDF Schema [56a], the Web Ontology Language (OWL) [57a] and the Web Service Modeling Language (WSML) family of languages [58a].

The Semantic Web working group of the W3C develops technologies and standards for the semantic description of the Web. The goal of these efforts is to make the Web understandable by machines and to increase the level of autonomous interoperation between computer systems [99a]. Several standards and languages were already introduced to address this objective. The most relevant are the Resource Description Framework (RDF) [102a], and the Web Ontology Language (OWL) [103a], [101a], [104a], [105a].

However, the goal of what is called SWS [59a] is the fruitful combination of Semantic Web technology and Web services. By using ontologies as the semantic data model for Web Service technologies Web Services have machine-processable annotations just as static data on the Web. Semantically enhanced information processing empowered by logical inference eventually shall allow the development of high quality techniques for automated discovery, composition, and execution of Services on the Web, stepping towards seamless integration of applications and data on the Web. Two relevant initiatives have to be considered in the context of Semantic Web services. Chronologically, the first one is OWL-S [60a], an upper level ontology for describing Web services, specified using OWL.

Questioning several deficiencies in this model [61a], there is proposed a new framework for Semantic Web Services by the Web Service Modeling Ontology (WSMO) [62a] which refines and extends the Web Service Modeling Framework (WSMF) [63a] to a meta-ontology for Semantic Web services. **WSMF defines a rich conceptual model for the development and the description of Web services based on two main requirements: maximal decoupling and strong mediation. WSMO is accompanied by a formal language, the Web Service Modeling Language (WSML) that allows one to write annotations of Web services according to the conceptual model. Also an execution environment (WSMX) [64a] for the dynamic discovery, selection, mediation, invocation, and inter-operation of Semantic Web services based on the WSMO specification is under development. [53a]**.

OWL-S [110a] is ontology for semantically describing services. The major difference to WSMO is that OWL-S does not consider the resolution of heterogeneity explicitly. While WSMO includes mediators as one of its key conceptual elements, OWL-S assumes that heterogeneity will be overcome by the underlying service infrastructure. For a complete comparison of both initiatives see [109a]. METEOR-S[6] aims at integrating current (syntactical) Web Service initiatives for description, composition, etc. with Semantic Web technologies. The METEOR-S (METEOR for Semantic Web services) project was focused on the usage of semantics for the complete lifecycle of semantic Web processes, namely, annotation, discovery, composition, and execution. However, METEOR-S does not provide a conceptual model for the description of business services and does not specifically address the integration of heterogeneous businesses.

The Semantic Web Services Initiative (SWSI) [91a] was an initiative of academic and industrial researchers, which has been composed to create infrastructure that combines Semantic Web and Web Services to enable the automation in all aspects of Web services. In addition to providing further evolution of OWL-S [92a], SWSI will also be a forum for working towards convergence of OWL-S with the products of the WSMO [33a] / WSML [94a] / WSMX [95a] research effort. WSMO is a complete ontology for the definition of Semantic Web Services. It follows the WSMF as a vision of Semantic Web Services. WSML is a family of languages that allow Semantic Web Service designers to define Semantic Web Services in a formal language. The WSMX provides a standard architecture for the execution of Semantic Web

[6] http://lsdis.cs.uga.edu/Projects/METEOR-S/

Services. Besides these major standards and initiatives, there are two ongoing projects being developed in the US, the LSDIS METEOR-S project [96a], and in Europe, the DERI SWWS project [97a], [98a].

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

This section is providing some details on the OWL-S, which is the result of a collaborative effort by researchers at several universities and organizations, including BBN Technologies, Carnegie Mellon University, Nokia Research Centre, Stanford University, SRI International, USC Information Sciences Institute, University of Maryland, Baltimore County, University of Toronto, Vrije Universiteit Amsterdam, University of Southampton, De Montfort University and Yale University.

The first approach for Semantic Web services has been provided by OWL-S [73, 74a]. Using OWL as the description language, OWL-S defines an upper ontology for semantically describing Web services that is comprised of three top-level elements: A service in OWL-S is described by means of three elements, as shown in **Errore. L'origine riferimento non è stata trovata.**.

1. The ***Service Profile*** describes what the service does. It explains what the service accomplishes, details limitations on its applicability and quality of service, and specifies requirements the service requester must satisfy to use it successfully. This information is used by consumers during the discovery of the service.

2. The ***Service Process Model*** describes how to use the service. It details the semantic content of requests, the conditions under which particular outcomes will occur, and, where necessary, the step-by-step processes leading to those outcomes.

3. The ***Service Grounding*** specifies the details of how to access/invoke a service. It includes communication protocol, message formats, serialization techniques and transformations for each input and output, and other service-specific details such as port numbers used in contacting the service [19a].

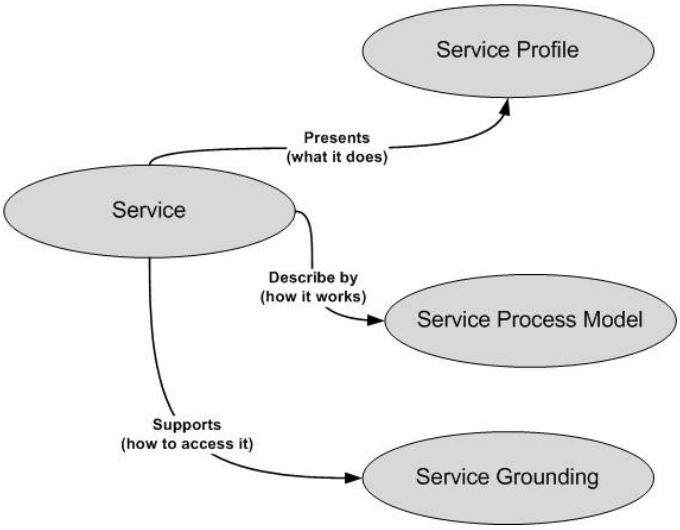

Figure 4.5 OWL-S Service Description Elements

Therewith, OWL-S provides a model for semantically describing Web services and serves as a basis for various research and development activities on Semantic Web service technologies [75a]. However, OWL-S is criticized for conceptual weaknesses and incompleteness: the meaning of the description elements is not clearly defined and thus used ambiguously, leading to misinterpretations and incompatible service descriptions; furthermore, although OWL-S allows other languages like KIF and SWRL for process descriptions besides OWL, their formal intersection is not defined, hence a coherent formalism for semantically describing Web services is not provided [61a]. However, the OWL-S provides developers with a strong language to describe the properties and capabilities of Web Services in such a way that the descriptions can be interpreted by a computer system in an automated manner [19a]. In the following, we briefly summarize the underlying standard ontology language OWL and then present each of the main elements of OWL-S service descriptions.

The information provided by an OWL-S description includes

- ontological description of the inputs required by the service
- outputs that the service provides
- preconditions and post conditions of each invocation

The goal of OWL-S is to enable applications to discover, compose, and invoke Web Services dynamically. Dynamic service discovery, composition, and invocation will allow services to be introduced and removed seamlessly from a services-rich environment, without the need to modify application code. If the information it needs to achieve its goals can be described in terms of an ontology that is shared with service providers, an application will be able to detect new services automatically as they are introduced and adapt transparently as the programmatic interfaces of services change. Although it is not the only technology being pursued to support dynamic environments, OWL-S is far enough along in its development to be used as a proof of concept, if not a potential solution.

Consequently, the emerging concept of Semantic Web services aims at providing more sophisticated Web Service technologies along with support for the Semantic Web. Mentioned first in [11a] and [32a], **Semantic Web services shall utilize ontology's as the underlying data model in order to support semantic interoperability between Web services and its clients and apply semantically enabled mechanisms for automated discovery, composition, conversation, and execution of Web Services. Therefore, exhaustive description frameworks are required that define the semantic annotations of Web services needed for automatically determine their usability**.

OWL-S is an upper ontology used to describe the semantics of services based on the W3C standard ontology OWL and is grounded in WSDL. It has its roots in the DAML Service Ontology (DAML-S) released in 2001, and became a W3C candidate recommendation in 2005. OWL-S builds on top of OWL and consists of three main upper ontologies: the Profile, the Process Model, and the Grounding (Figure 3.17).

The OWL-S initiative [65a] uses OWL to develop an ontology of services, covering different aspects of Web services, among them functionality. To describe their functionality, services are viewed as processes that (among other things) have pre-conditions and effects. However, the faithful representation of the dynamic behavior of such processes (what changes of the world they cause) is beyond the scope of a static ontology language like OWL. In AI, the notion of an action is used both in the planning and the reasoning about action communities to denote an entity whose execution (by some agent) causes changes of the world (see e.g. [67, 70a]). Thus, it is not surprising that theories developed in these communities have been applied in the context of Semantic Web services. For example, [11, 39a] use the situation calculus [67a] and GOLOG [68a] to formalize the dynamic aspects of Web services and to describe their composition. In [65a], OWL-S process models are translated into the planning language of the HTN planning system SHOP2 [69a], which are then used for automatic Web service composition.

4.4.1.1 OWL-S Discovery and Execution Elements

By itself, OWL-S is a language for the markup of Web Services. It becomes useful when there are tools to exploit Web Services described using OWL-S constructs.

An example of an OWL-S toolkit is CMU's OWL-S Development Environment (CODE), created by Carnegie Mellon University's Intelligent Software Agents Lab. CODE supports the complete OWL-S Web Services development process—from the generation of OWL-S descriptions (from Java code or WSDL documents) to the deployment and registration of the service. CODE is implemented as an Eclipse plug-in that supports activities for service providers and client application developers[7]. In addition to tools for the description of services, CODE includes the OWL-S Matchmaker and the OWL-S Virtual Machine (VM) elements. The OWL-S Matchmaker serves as a "catalog" of services defined using OWL-S. Service providers register OWL-S descriptions of services with the OWL-S Matchmaker. Client applications can query the OWL-S Matchmaker with an ontological description of the desired inputs and outputs. The OWL-S Matchmaker matches the request with its catalog of services and returns a ranked list of services that most closely match the request.

[7] Eclipse is an open source community engaged in providing a vendor-neutral software development platform. Read more about Eclipse at http://www.eclipse.org/

The OWL-S VM is used to invoke services using OWL-S. After the client application selects a service from the ranked list of services, it formulates its request using the format specified by the OWL ontology and sends the request to the OWL-S VM. Using Extensible Style Language Transformations (XSLT)[8] present in the Service Grounding element, the OWL-S VM reformats the request to match the format required by the service. Then, it invokes the service on behalf of the client. When it receives a response to that step, the OWL-S VM uses another XSLT transformation in the Service Grounding element to reformat the response into a format matching that of the ontology. Finally, the OWL-S VM sends the response back to the client application. In this manner, the client application does not need to know anything about how to interact with the actual service; the OWL-S VM acts as a mediator for the request and response.

Both the OWL-S Matchmaker and the OWL-S VM elements have an Application Programming Interface (API) for Java applications to discover and invoke services [17a].

Fortunately, the Intelligent Software Agents Lab at Carnegie Mellon University has played a critical role in the development of OWL-S from its very inception [112a]. This group developed CODE. Thus, CODE was selected as the development environment, and support was graciously provided by Naveen Srinivasan of the Intelligent Software Agents Lab.

4.4.1.1.1 Service Profile

The OWL-S profile ontology is used to describe what the service does, and is meant to be mainly used for the purpose of service discovery. An OWL-S service profile or signature encompasses its functional parameters, i.e. hasInput, hasOutput, precondition and e.ect (IOPEs), as well as non-functional parameters such as serviceName, serviceCategory, qualityRating, textDescription, and meta-data (actor) about the service provider and other known requesters. Please note that, in contrast to OWL-S 1.0, in OWL-S 1.1 the service IOPE parameters are de.ned in the process model with unique references to these definitions from the profile.

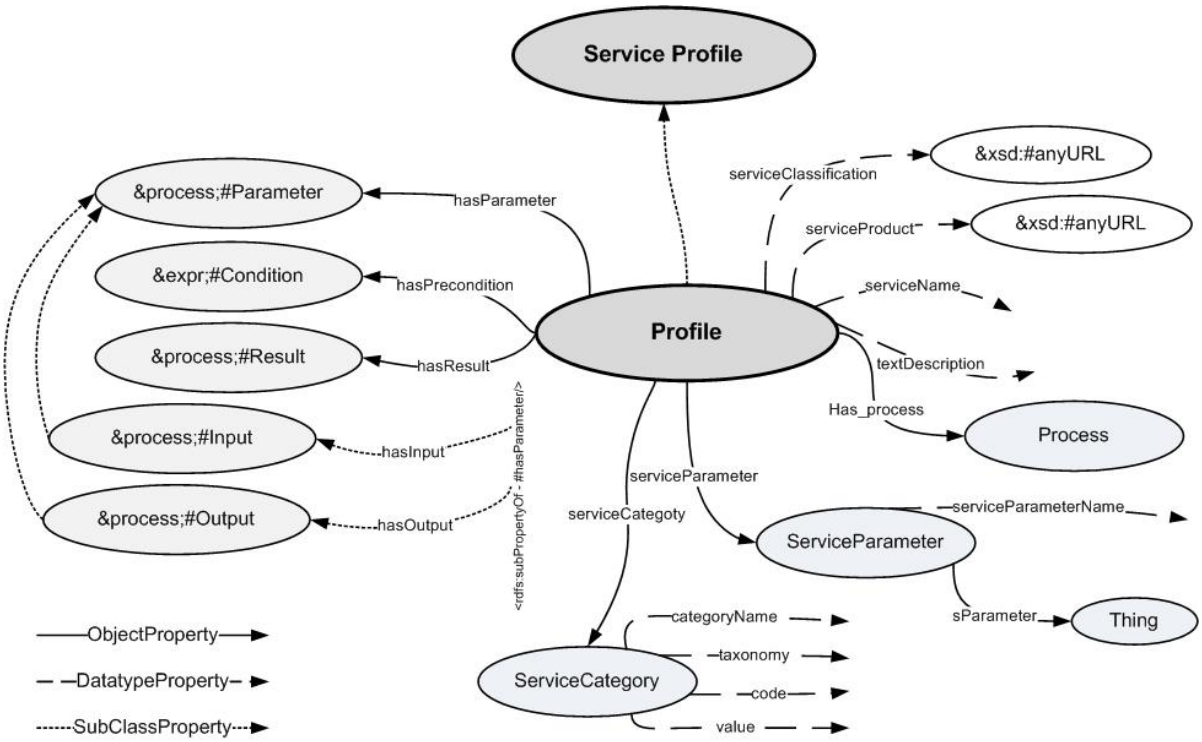

Figure 4.6 OWL-S service profile structure

[8] In its simplest definition, *XSLT* is a language used to transform XML documents into other XML documents. Currently, XSLT is the only type of transformation supported by the OWL-S VM

Inputs and outputs relate to data channels, where data flows between processes. Preconditions specify facts of the world (state) that must be asserted in order for an agent to execute a service. Effects characterize facts that become asserted given a successful execution of the service in the physical world (state). Whereas the semantics of each input and output parameter is defined as an OWL concept formally specified in a given ontology, typically in decidable OWL-DL or OWL-Lite, the preconditions and effects can be expressed in any appropriate logic (rule) language such as KIF, PDDL, and SWRL. Besides, the profile class can be sub classed and specialized, thus supporting the creation of profile taxonomies which subsequently describe different classes of services.

4.4.1.1.2 Service Process Model

An OWL-S process model describes the composition (choreography and orchestration) of one or more services that is the controlled enactment of constituent processes with respective communication pattern. In OWL-S this is captured by a common subset of workflow features like split + join, sequence, and choice Figure 4.7.

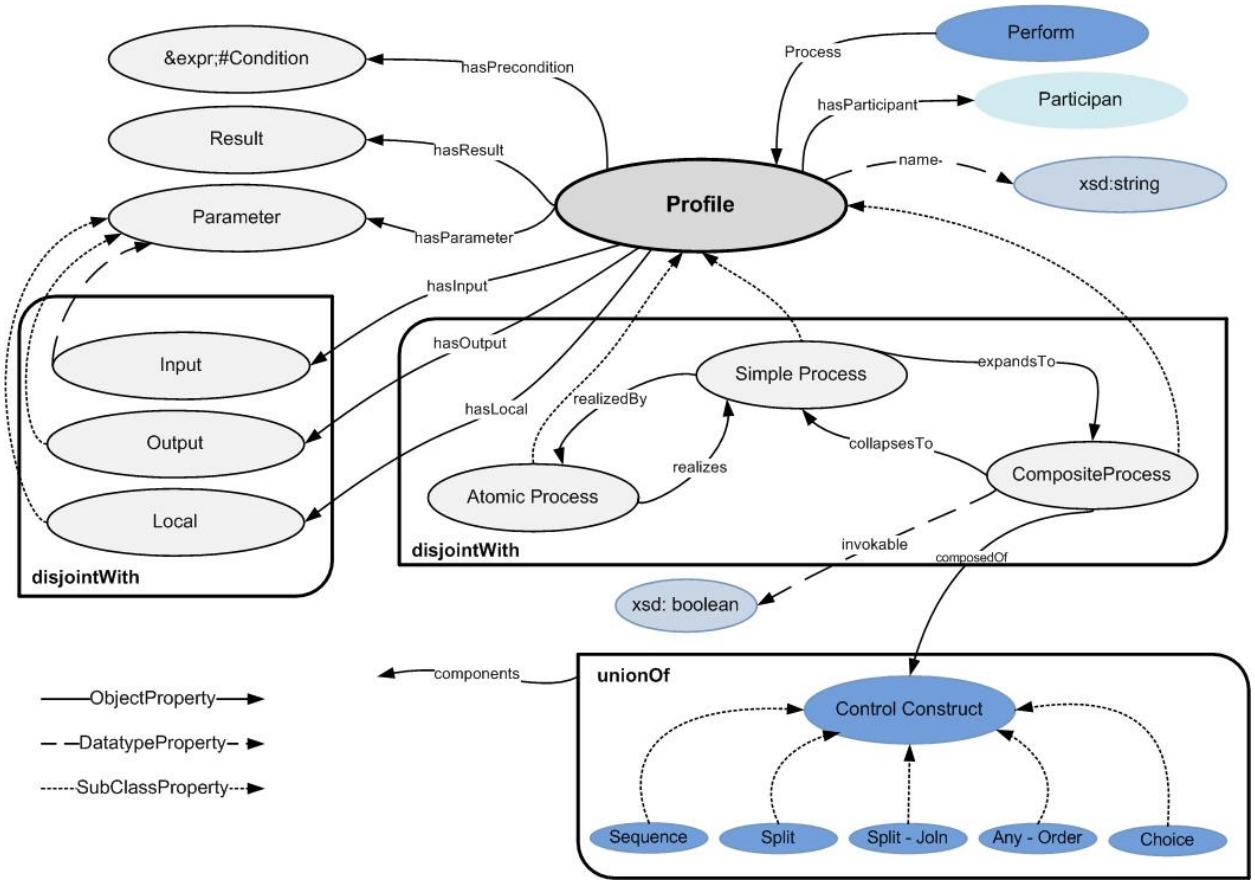

Figure 4.7 OWL-S service process model

Originally, the process model was not intended for service discovery but the profile by the OWL-S coalition. More concrete, a process in OWL-S can be atomic, simple, or composite. An atomic process is a single, black-box process description with exposed IOPEs. Simple processes provide a means of describing service or process abstractions which have no specific binding to a physical service, thus have to be realized by an atomic process, e.g. through service discovery and dynamic binding at runtime, or expanded into a composite process.

Composite processes are hierarchically defined workflows, consisting of atomic, simple and other composite processes. This process workflows are constructed using a number of different control flow operators including Sequence, Unordered (lists), Choice, If-then-else, Iterate, Repeat-until, Repeat-while, Split, and Split + Join. In OWL-S 1.1, the process model also speci.es the inputs, outputs, preconditions, and effects of all processes that are part of a composed service, which are referenced in the profiles of the respective services. An OWL-S process model of a composite service can also specify that its output

is equal to some output of one of its sub processes whenever the composite process gets instantiated. Moreover, for a composite process with a Sequence control construct, the output of one sub process can be de.ned to be an input to another sub process (binding).

Unfortunately, the semantics of the OWL-S process model are left undefined in the official OWL-S documents. Though there are proposals to specify these semantics in terms of, for example, the situation calculus, and the logic programming language GOLOG based on this calculus [27a].

4.4.1.1.3 Service Grounding

The grounding of a given OWL-S service description provides a pragmatic binding between the logic-based and XMLS-based service definitions for the purpose of facilitating service execution. Such a grounding of OWL-S services can be, in principle, arbitrary but has been exemplified for grounding in WSDL to pragmatically connect OWL-S to an existing Web Service standard Figure 4.8.

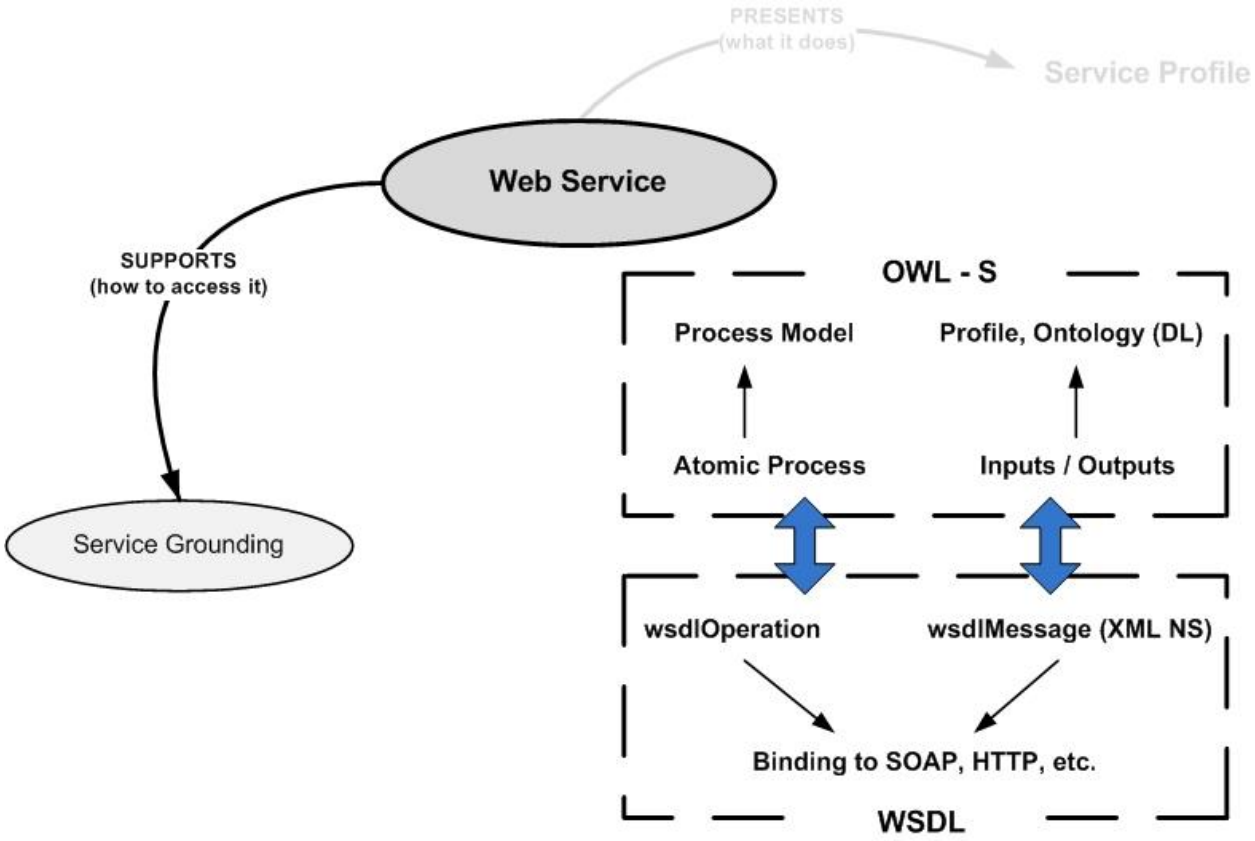

Figure 4.8 Grounding of OWL-S in WSDL

In particular, the OWL-S process model of a service is mapped to a WSDL description through a thin (incomplete) grounding: Each atomic process is mapped to a WSDL operation, and the OWL-S properties used to represent inputs and outputs are grounded in terms of respectively named XML data types of corresponding input and output messages. Unlike OWL-S, WSDL cannot be used to express pre-conditions or effects of executing services. Any atomic or composite OWL-S service with grounding in WSDL is executable either by direct invocation of the (service) program that is referenced in the WSDL file, or by a BPEL engine that processes the WSDL groundings of simple or orchestrated Semantic Web Services.

4.4.1.2 Software Support

One prominent software portal of the semantic Web community is SemWebCentral9 developed by InfoEther and BBN Technologies within the DAML program in 2004 with BBN continuing to maintain it today. As a consequence, it comes at no surprise that this portal o.ers a large variety of tools for OWL

and OWL-S service coordination as well as OWL and rule processing. Examples of publicly available software support of developing, searching, and composing OWL-S services are as follows. [18a] http://projects.semwebcentral.org/

- *Development* OWL-S IDE integrated development environment[9], the OWL-S 1.1 API[10] with the OWL-DL reasoner Pellet[11] and OWL-S editors.
- *Discovery* OWL-S service matchmakers OWLS-UDDI[12], OWLSM[13] and OWLS-MX[14] with test collection OWLS TC2.
- *Composition* OWL-S service composition planners OWLS-XPlan[15], GOAL[16].

The model problem experience has proven that regardless of its contributions to the semantics community, OWL-S is not ready to support the dynamic discovery, composition, and invocation of services, mostly due to the scarcity of tool support. Nonetheless, despite its problems, OWL-S has tremendous potential if given the proper resources and opportunities. Being able to define the inputs and outputs of a service in terms of ontology is a huge step towards dynamic discovery, composition, and invocation without user intervention. Unfortunately, funding—which is what makes technologies real—has stopped for OWL-S tool development. Additional investment in completing and debugging CODE (or other tools that are as further along as CODE) would demonstrate the feasibility of the technology and encourage potential adoption from industry that in turn would provide feedback for further improvements. With development, OWL-S could enable applications to dynamically discover, compose, and invoke new services to solve problems and gain information in ways that were not available when those applications were created. OWL-S has the capability to embed semantic meaning into the collections of services available in services-oriented computing environments, which will allow applications to be developed without knowledge of specific services that may or may not be available. That capability could also make SOAs more robust and flexible. Collections of services that are defined using OWL-S would allow applications to be tolerant of faults in a dynamic and transparent way by simply accessing other services with similar semantic meanings. OWL-S could also enable services to be registered and removed at runtime dynamically without causing downtime in client applications.

4.4.2 Web Service Modeling Ontology (WSMO)

WSMO [28, 29, 76a] is a member submission to W3C of an ontology that aims at describing all relevant aspects for the partial or complete automation of discovery, selection, composition, mediation, execution and monitoring of Web services. WSMO has its roots in the Web Service Modeling Framework (WSMF) and in Problem- Solving Methods [30a], notably the Unified Problem Solving Method Development Language UPML [31a], which have been extended and adapted in order to support the automation of the aforementioned tasks for manipulating Web services.

The Web Service Modeling Ontology (WSMO) aims at describing all relevant aspects related to general services which are accessible through a Web service interface with the ultimate goal of enabling the (total or partial) automation of the tasks (e.g., discovery, selection, composition, mediation, execution, monitoring, etc.) involved in both intra- and inter- enterprise integration of Web services. WSMO has its conceptual basis in the Web Service Modeling Framework (WSMF) [32a], refining and extending this framework and developing a formal ontology and set of languages [10a].

Taking the Web Service Modeling Framework WSMF as its conceptual basis [32a], the WSMO project is an ongoing research and development initiative for defining a capacious framework for Semantic Web services along with a description language (the Web Service Modeling Language WSML [78a]) and a reference implementation (the Web Service Execution Environment WSMX [77a]).

[9] http://projects.semwebcentral.org/projects/owl-s-ide/
[10] http://projects.semwebcentral.org/projects/owl-s-api
[11] http://projects.semwebcentral.org/projects/pellet/
[12] http://projects.semwebcentral.org/projects/mm-client/
[13] http://projects.semwebcentral.org/projects/owlsm/
[14] http://projects.semwebcentral.org/projects/owls-mx/
[15] http://projects.semwebcentral.org/projects/owls-xplan/
[16] http://www.smartweb-project.de

In order to provide a detailed synopsis of the aims, challenges, and respective solutions for Semantic Web services, the following explains the design principles and element definitions of WSMO in detail.

4.4.2.1 WSMO Design Principles

Since Semantic Web services aim at turning the Internet from an information repository for human consumption into a world-wide system for distributed Web computing by combining Semantic Web technologies and Web services, WSMO, as any other framework for Semantic Web services description needs to integrate the *basic Web design principles*, the *Semantic Web design principles*, as well as *design principles for distributed, service oriented computing for the Web*. This section enumerates and discusses the design principles of WSMO.

4.4.2.2 WSMO Basic Concepts

WSMO defines the modeling elements for describing Semantic Web services based on the conceptual grounding set up in the Web Service Modeling Framework (WSMF) [63a], wherein four main components are defined: ontologies, Web services, goals, and mediators.

WSMO inherits these four top elements, further refining and extending them. *Ontologies* represent a key element in WSMO since they provide (domain specific) terminologies for describing the other elements. They serve a twofold purpose: defining the formal semantics of the information, and linking machine and human terminologies.

Web services connect computers and devices using the standard Web-based protocols to exchange data and combine data in new ways. Their distribution over the Web confers them the advantage of platform independence. Each Web service represents an atomic piece of functionality that can be reused to build more complex ones. Web services are described in WSMO from three different perspectives: non-functional properties, functionality and behavior. *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. The coexistence of goals and Web services as non-overlapping entities ensures the decoupling between request and Web service. This kind of stating problems, in which the requester formulates objectives without regard to Web services for resolution, is known as *the goal-driven approach*, derived from *the AI rational agent approach*. *Mediators* describe elements that aim to overcome the mismatches that appear between the different components that build up a WSMO description. The existence of mediators allows one to link possibly heterogeneous resources. They resolve incompatibilities that arise at different levels: **–** data level - mediating between different used terminologies, more specifically solving the problem of ontology integration process level - mediating between heterogeneous communication patterns. This kind of heterogeneity appears during the communication between Web services.

This deliverable briefly introduced WSMO, one of the most salient efforts in the domain of Semantic Web services. Semantic Web Services are a key application area for Intelligent Agent Systems because of the necessity for semantic descriptions frameworks as a basis for such Intelligent Systems. Thus, WSMO and its formalization WSML provide the infrastructure for such systems.

Several implementations are already available or under development. The first version of the Web Service Execution Environment (WSMX) has been available since June 2004 at the Source Forge portal. Apart from WSMX which marks a reference architecture and implementation for a WSMO compliant execution environment there already exists several other ready-to-use implementations and tools for WSMO.

4.4.2.3 Top-level elements of WSMO

Following the key aspects identified in the Web Service Modeling Framework, WSMO identifies four top-level elements as the main concepts which have to be described in order to define Semantic Web Services:

Ontologies provide the terminology used by other WSMO elements to describe the relevant aspects of the domains of discourse. In contrast to mere terminologies that focus exclusively on syntactic aspects,

ontologies can additionally provide formal definitions that are machine-processable and thus allow other components and applications to take actual meaning into account.

Web services represent computational entities able to provide access to services that, in turn, provide some value in a domain; Web service descriptions comprise the capabilities, interfaces and internal working of the service.

All these aspects of a Web service are described using the terminology defined by the ontologies.
Goals describe aspects related to user desires with respect to the requested functionality; again, Ontologies can be used to define the used domain terminology, useful in describing the relevant aspects of goals. Goals model the user view in the Web service usage process, and therefore are a separate top-level entity.

Finally, Mediators describe elements that handle interoperability problems between different WSMO elements. We envision mediators as the core concept to resolve incompatibilities on the data, process and protocol level, i.e. in order to resolve mismatches between different used terminologies (data level), in how to communicate between Web services (protocol level) and on the level of combining
Web services (process level).

4.4.2.3.1 Ontologies

In compliance to the vision of the Semantic Web, WSMO uses ontologies as the underlying data model for Semantic Web services. This means that all resource descriptions and all information interchanged during collaboration execution is based on ontologies, thereby providing the basis for semantically enhanced information processing and ensuring semantic interoperability between Semantic Web services. In accordance to the AI-theory of ontologies [80a], WSMO ontologies consist of the following elements: *Concepts* describe the entities of a domain that are characterized by *Attributes*; *Relations* describe associations between concepts, whereby subsumption and membership relationships define the taxonomic structure of ontology. An *Instance* is a concrete individual of a concept, and *Axioms* define constraints and complex aspects of the domain in terms of logical expressions. Regarding engineering methodologies developed for the Semantic Web [81a], ontology design in WSMO demands and supports *modularization*, i.e. small-sized and concise ontologies, *decoupling*, i.e. distributed and multi-party ontology development and *ontology mediation* for resolving possibly occurring mismatches between loosely coupled ontologies for a specific usage scenario.

4.4.2.3.2 Web Services

WSMO defines a description model that encompasses that information needed for automatically determining the usability of a Web service. WSMO Web service description is comprised of four elements: (1) *non-functional properties*, (2) a *capability* as the functional description of the service; summarized as service interfaces, (3) a *choreography* that describes the interface for service consumption by a client, and (4) an *orchestration* that describes how the functionality of the service is achieved by aggregating other Web services. These notions describe the functionality and behavior of a Web service, while its internal implementation is not of interest.

4.4.2.3.3 Goals

In order to facilitate automated Web service usage and support ontological separation of user desires, service usage requests, and Web service descriptions, Goals in WSMO allow specifying objectives that clients - which can be humans or machines - wish to achieve. The general structure of WSMO Goal descriptions is similar to Web service descriptions. The client can specify the functionality expected in a *requested capability* that can consist of the same elements as Web service capabilities as discussed above. Also, a Goal can carry information on the expected behavior of an acceptable Web service in so-called *requested interfaces* that can define the excepted communication behavior for consuming a Web service with respect to its Choreography Interface as well as restrictions on other Web services aggregated in the orchestration of an acceptable Web service (e.g. only Web services are accepted that utilize a trusted payment facility). It is important to remark that Goal descriptions are defined from the client perspective, thereby decoupled from Web service descriptions.

4.4.2.3.4 Mediators

Mediation is concerned with handling heterogeneity, i.e. resolving possibly occurring mismatches between resources that ought to be interoperable. Heterogeneity naturally arises in open and distributed environments, and thus in the application areas of Semantic Web services. Hence, WSMO defines the concept of Mediators as a top level notion

Mediator-orientated architectures as introduced in [82a] specify a mediator as an entity for establishing interoperability of resources that are not compatible a priori by resolving mismatches between them at runtime. The aspired approach for mediation relies on declarative description of resources whereupon mechanisms for resolving mismatches work on a structural, semantic level, in order to allow generic, domain independent mediation facilities as well as reuse of mediators. Concerning the needs for mediation within Semantic Web services, WSMO distinguishes three levels of mediation:

1. Data Level Mediation - mediation between heterogeneous data sources; within ontology-based frameworks like WSMO, this is mainly concerned with ontology integration.
2. Protocol Level Mediation - mediation between heterogeneous communication protocols; in WSMO, this mainly relates to choreographies of Web services that ought to interact.
3. Process Level Mediation - mediation between heterogeneous business processes; this is concerned with mismatch handling on the business logic level of Web services (related to the orchestration of Web services).

4.4.2.4 Semantic Web Service Technologies

Hence, the main working areas of Semantic Web service technologies for enabling automated Web service usage address the following aspects wherefore we outline most recent approaches below.

- Discovery and Selection: how to determine appropriate Web services for solving the goal of a client, and how to select the concrete Web service to be used in case several service can be used?
- Composition: if there does not exists a Web service that can completely satisfy a more complex goal, how to determine Web services that can be combined for solving the goal, and how to determine a suitable execution order for these services?
- Conversation Validation: given the behavior descriptions of Web services and clients that are ought to interact, how to determine whether the information interchange expected by each party can be achieved successfully?
- Mediation: how to resolve mismatches and heterogeneities that hamper Web services and clients to interoperate?
- Execution Support: how to manage and control execution of Web services, and how to ensure information interchange with respect to Web scale?

4.4.2.5 Discovery

Discovery within Web services is concerned with detecting suitable Web services for achieving a requester's objective, i.e. a goal. As outlined introductory, UDDI supports discovery in conventional Web service technology as follows. The service requester, which in this setting is expected to be a system developer, browses a UDDI repository, retrieving information on available Web services. Then, he manually inspects the usability of a Web service and integrates the respective technical service invocation into the target application. As this technology support appears to be unsatisfactory for automated Web service usage, Semantic Web service discovery aims at automatically determining the usability of a Web service by semantic matchmaking.

Referred to as functional discovery, the general approach is to inspect certain logical relationships between the semantic capability descriptions of requests (i.e. Goals) and Web services. If such a relationship holds, the Web service is considered to be usable for resolving the client's goal. On basis of several preceding works on semantic matchmaking [83a], [84a], the Web service discovery framework defined for WSMO [85a] identifies five matchmaking notions as the core for functional discovery as shown in Figure 4.9. The important characteristic of these notions is that each one denotes a different logical relationship that has to hold for considering a Web service to be suitable for achieving a given Goal.

For instance, the Exact Match holds if and only if for each possible ontology instance that can satisfy the Web service holds that it also satisfies the Goal, and that there exists no possible ontology instance that satisfies only the Goal or the Web service. In contrast, the Intersection Match holds if there exists one possible instance that can satisfy both the Goal and the Web service. Hence, in order to precisely express client objectives with respect to discovery, WSMO Goals carry an additional non-functional property type Of Match denoting the matchmaking notion to be applied.

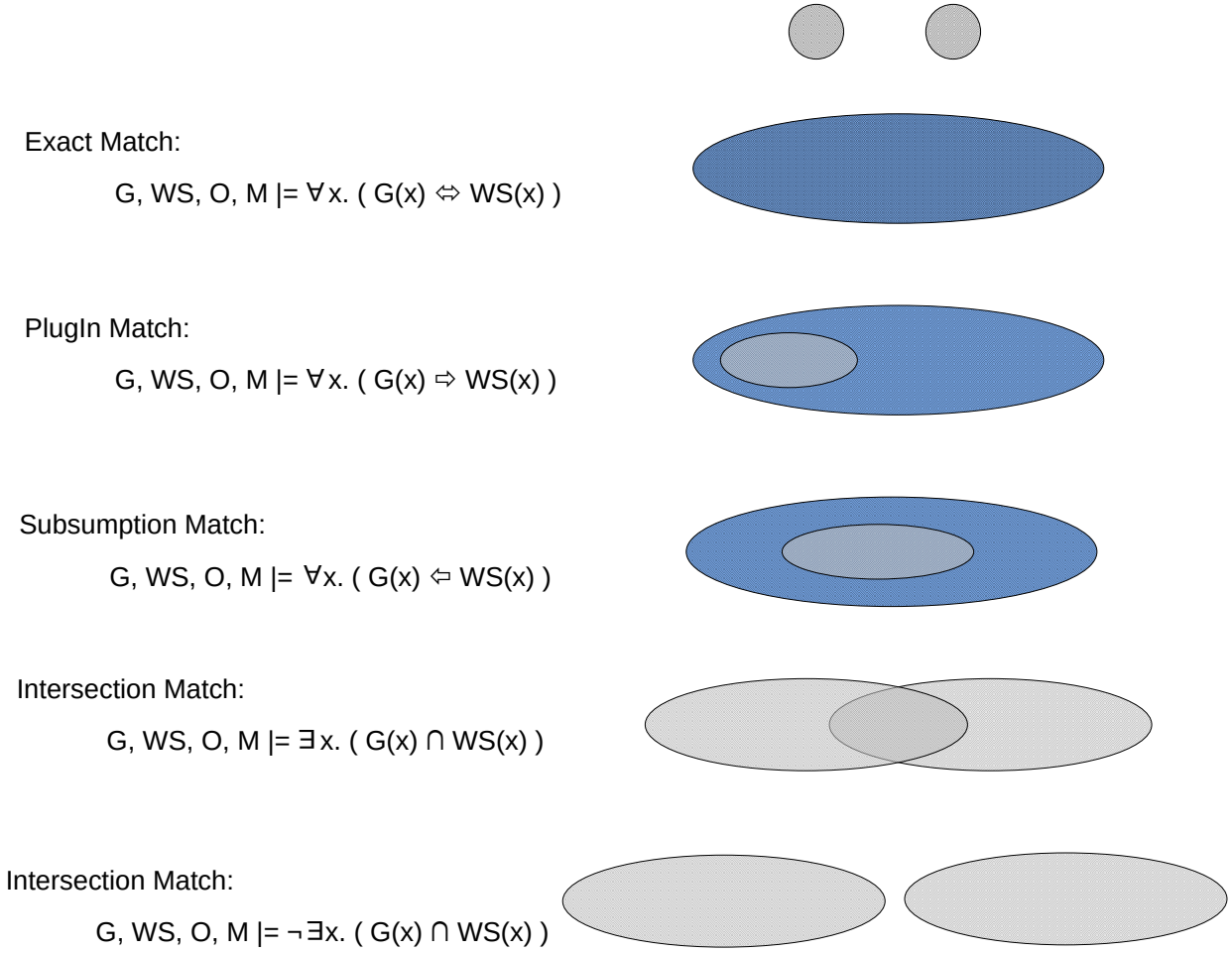

Figure 4.9 Semantic Discovery Matchmaking Notions

Several prototypical realizations for semantically enabled discovery are existing: [84a] and [86a] apply Description Logic reasoners, the approach in [87a] is based on Frame- and Transaction Logic, and [88a] use a FOL theorem proves as the technical platform for semantically enabled discovery. However, all these approaches are restricted to the specific reasoning support provided by the used tools. Hence, it is expected that these approaches will converge towards an integrated discovery framework for Semantic Web services that provides appropriate reasoning facilities[17].

4.4.3 IRS-III

The Internet Reasoning Service (*IRS-III*) is an infrastructure for publishing, locating, executing and composing Semantic Web services, organized according to the WSMO framework. *WSMO Studio* is a Semantic Web Service editor compliant with WSMO. The WSMO Studio will be available as a set of Eclipse plug-ins that will allow easy reusability and extension from 3rd parties. wsmo4j is a Java API and a reference implementation for building Semantic Web services applications compliant with WSMO. Like WSMX, it is also being developed as an Open Source project. The Semantic Web Fred [55a] combines

[17] Description Logic and Logic Programming are the most prominent decidable subsets of First Order Logic; existing reasoners for each subset provide specific reasoning support. For discovery, as well as for several other reasoning tasks on the Semantic, facilities from both fractions are required. Hence, ongoing efforts aim at defining the maximal intersection of Description Logic and Logic Programming languages in order to facilitate integrated reasoning support. See [89a] for further discussion.

combine agent technology with WSMO, in order to provide advanced support for Semantic Web applications.

The IRS project (http://kmi.open.ac.uk/projects/irs) has the overall aim of supporting the automated or semi-automated construction of semantically enhanced systems over the internet. IRS-I supported the creation of knowledge intensive systems structured according to the UPML framework and IRS-II integrated the UPML framework with Web Service technology. IRS-III [33a] has incorporated and extended the WSMO ontology [34a] so that the implemented infrastructure allows the description, publication and execution of Semantic Web Services (SWS). The meta-model of WSMO describes four top level elements (in italics hence forth):

- *Ontologies*,
- *Goals*,
- *Web Services*, and
- *Mediators*.

Ontologies provide the foundation for semantically describing data in order to achieve semantic interoperability and are used by the three other WSMO elements.

Goals define the tasks that a service requester expects a *web service* to fulfill. In this sense they express the service requester's intent. *Web services* represent the functional behavior of an existing deployed *Web Service*. The description also outlines how Web Services communicate (*choreography*) and how they are composed (*orchestration*).

Mediators describe the connections between the components above and represent the type of conceptual mismatches that can occur. In particular, WSMO provides four kinds of *mediators*: *oo-mediators* link and map between heterogeneous ontologies; *ww-mediators* link *web services* to *web services*; *wg-mediators* connect *web services* to *goals*; *gg-mediators* link different *goals*.

IRS-III provides the representational and reasoning mechanisms for implementing the WSMO meta-model mentioned above in order to describe Web Services. Additionally, IRS-III provides a powerful execution environment which enables these descriptions to be associated to a deployed Web Service and instantiated during selection, composition, mediation and invocation activities. [35a]

IRS (Internet Reasoning Service) is a framework to support Semantic Web Services, in which services can be described by their semantics, discovered, invoked and monitored. IRS-III consists of three main components, IRS Server, IRS Publisher and IRS Client. IRS Server stores and reasons with Web Service and Goal descriptions. IRS Publisher generates wrappers for programs as Web Services. Invocation of Web Services via Goal descriptions is supported by the IRS Client.

The notions of Goal and Mediator are particular characteristic of IRS-III and WSMO ontology. While a Web Service is a description of a method and concerned with the specification of mechanisms and execution, a Goal is a general description of a problem and concerned with describing a problem rather than mechanisms. As a result, Goals are suitable for describing a service for a user whose concern is not the technical details of the solution.

4.4.3.1 The IRS-III Framework

IRS-III is based on a distributed architecture composed of the IRS-III server, the publishing platforms and clients which communicate through the SOAP protocol, as shown in Figure 4.10. The server handles ontology management and the execution of knowledge models defined for WSMO. The server also receives SOAP requests (through the API) from client applications for creating and editing WSMO descriptions of *goals*, *web services* and *mediators* as well as goal-based invocation. At the lowest level the IRS-III Server uses an HTTP server written in Lisp, which has been extended with a SOAP handler.
The publishing platforms allow providers of services to attach semantic descriptions to their deployed services and provide handlers to invoke services in a specific language or platform (Web Services WSDL, Lisp code, Java code, and Web applications). When a Web Service is published in IRS-III the information about the publishing platform (URL) is also associated with the *web service* description in order to be invoked. The IRS-III server is written in Lisp and is available as an executable file.

The publishing platforms are delivered as Java Web applications; and client applications use the Java API. [35a]

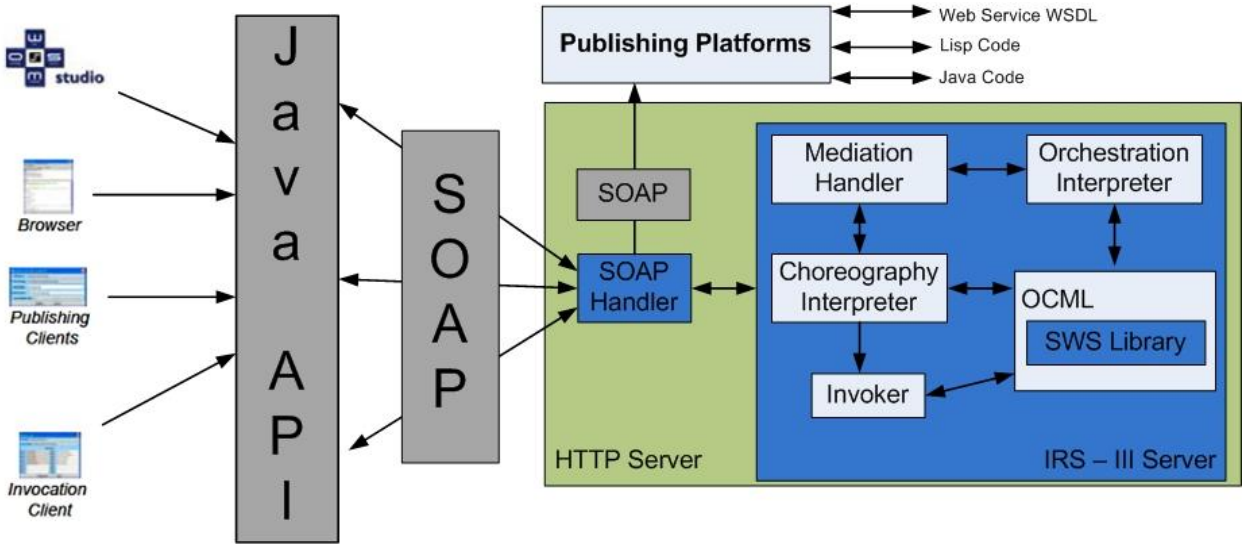

Figure 4.10 The IRS-III framework

The main components of IRS-III are explained in the following:

SWS Library – At the core of the IRS-III server is the SWS library where the semantic descriptions are stored using our representation language OCML. The library is structured into knowledge models for *goals*, *web services* and *mediators*. Domain *ontologies* and knowledge bases (instances) are also available from the library.

Choreography Interpreter – This component interprets the *grounding* and *guarded transitions* of the *choreography* description when requested by the mediation handler.

Orchestration Interpreter – This component interprets the workflow of the orchestration description when requested by the mediation handler.

Mediation Handler – The brokering activities of IRS-III including selection, composition and invocation are each supported by a specific mediation component within the mediation handler. These activities may involve executing a mediation service or mapping rules declared in a *mediator* description.

Invoker – The invoker component of the server communicates with the publishing platform, sending the inputs from the client and bringing the result back to the client.

4.5 Service Composition

SEMANTIC web service (SWS) composition process is generally performed when no available single or composite service can satisfy the required request and a combination which can satisfy request can be generated from available single or composite services.

Regarding service composition methods different techniques can be found in the bibliography. These techniques are usually divided into the categories of static (services to be composed decided at design time) and dynamic composition (services to be composed decided at runtime), or automatic (no user intervention) and manual composition (user-driven composition). The service composition approaches are categorized according to the research area they base their methods on.**Errore. L'origine riferimento non è stata trovata.** AI planning introduces the principles of Artificial Intelligence into service composition, Semantic Web approaches exploits the semantic characteristics of a service and Middleware approaches consider different middleware solutions.

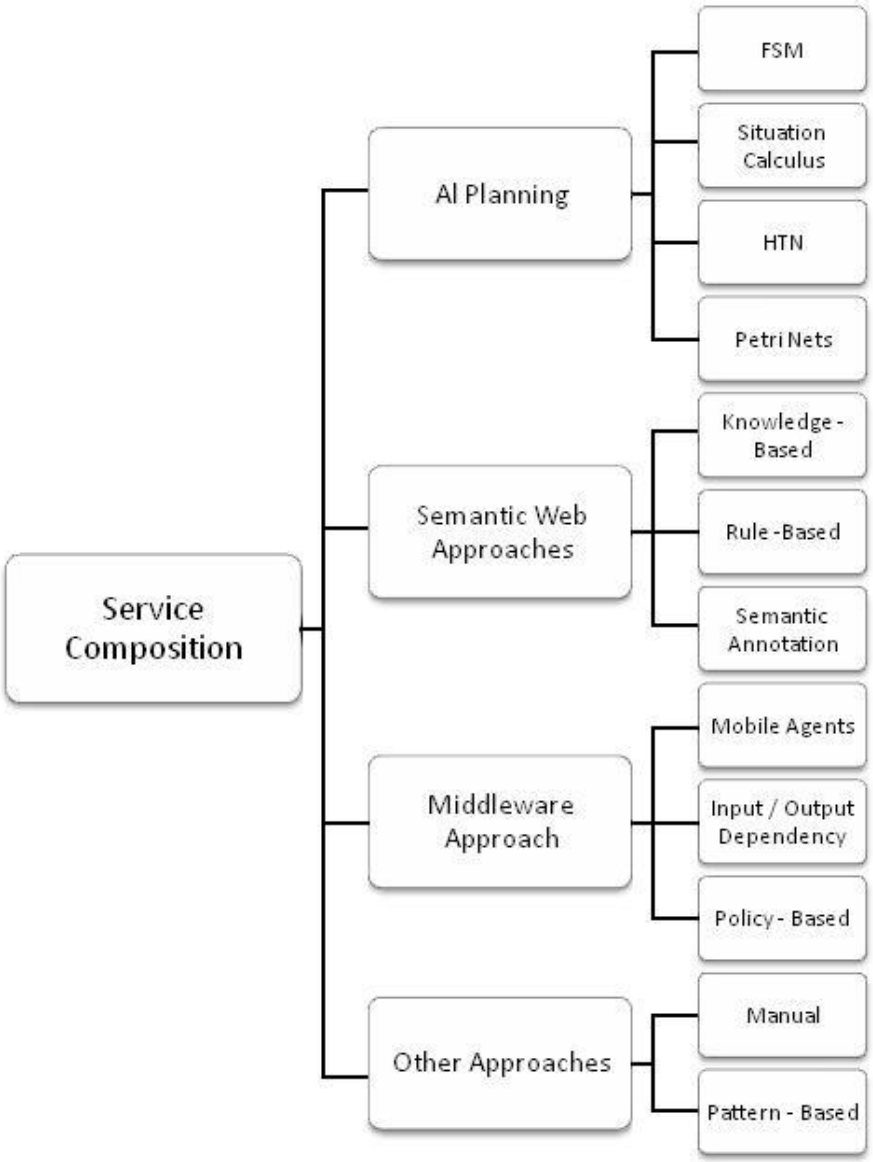

Figure 4.11 Service Composition Categorization

In general, there are several kinds of service composition. The traditional approach is called ***manual composition*** where users program and tell the system what to do during all the composition process development steps. Processes are defined using a process execution language like BPEL. Many plug-ins for tools like Net-Beans [44a], JOpera [45a] are available to enable manual composition.
The problem with such an approach is that it demands too much knowledge on the part of the user and it becomes more and more difficult with the explosion of web services resources.

The second approach is called ***automatic composition*** (without human involvement). It is used when the requestor has no process model but has a set of constraints and preferences. It is based on finding services for executing predefined abstract processes. The tools try to discover the available web services that semantically match as much as possible the user's needs [46a]. Several approaches for automatic service composition have been introduced [47a], including solutions based on Hierarchical Task Network (HTN), Golog [48a], Artificial Intelligence (AI) planning or Rule-Based planning [47, 49-50a]. However, automatic composition is still viewed as a task of high complexity because of the rapid proliferation of available services to choose from and the composition result risks to differ from the user's original goal.

The third approach is called ***semiautomatic or interactive composition***. In this kind of composition, the system usually helps users to find, filter and integrate automatically the desired services by matching the users requests with the available services. Moreover, it enables end users to intervene continuously during the composition process. Some efforts like OWL-S [51a], METEOR-S [52a] use semantic

description of web services to help improving the discovery and composition processes. We believe that the semantic of the provided services could be used by tools or systems to guide the user to limit the available choices and to define his preferences to finally reach the composition goal.

4.5.1 Automated Web Service Composition Methods

Therefore, building composite Web services with an automated or semiautomated tool is critical. To that end, several methods for this purpose have been proposed. In particular, most researches conducted fall in the realm of workflow composition or AI planning.

Web service composition can be seen as the construction of a process to attain a certain goal. This is a problem which has been investigated extensively by research in Artificial Intelligence (AI). A classical planning problem includes a description of the initial state of the world, a description of the desired goal, and a description of the possible actions which may be executed. We make an overview of AI approaches to planning that have been investigated for the purpose of Web service composition.

4.5.1.1 Situation calculus

Situation calculus adapt and extend the Golog language for automatic construction of Web services. Golog is a logic programming language built on top of the situation calculus. The general idea of this method is that software agents could reason about Web services to perform automatic Web service discovery, execution, composition and inter-operation. The user's request (generic procedure) and constraints can be presented by the first-order language of the situation calculus (a logical language for reasoning about action and change).

In Situation Calculus a dynamic world is modeled as progressing through a series of situations as a result of various actions being performed within the world. A situation represents a history of action occurrences. The constant 0 *S* describes the initial situation where no actions have occurred yet. The transition to the successor situation of *s* as result of an action *a* is denoted using *do* (*a*, *s*). Lastly, the truth value of statements given a situation is modeled by fluents which are denoted by predicate symbols and take a situation as their last argument. In [38a] the authors argue that composition of Web services, viewed as execution trees, can be realized, at least under certain assumptions, in reasoning about actions formalisms, making specific use of Situation Calculus. In [39a] the use of an augmented version of Golog is suggested. Golog is a high-level logic programming language, built on top of the Situation Calculus, for the specification and execution of complex actions in dynamical domains. The authors extended Golog to provide knowledge and sensing actions in order to become a suitable formalism for representing and reasoning about the semantic Web services composition problem. The general idea of the method described is that software agents could reason about Web services to perform automatic Web service discovery, execution, composition and inter-operation.

4.5.1.2 Hierarchical Task Networks

Hierarchical Task Network (HTN) planning is an artificial intelligence (AI) planning method.

Hierarchical Task Network (HTN) [40a] planning is a method of planning by task decomposition. In contrary to the other concepts of planning, the central concept of HTNs is not states, but tasks. An HTN based planning system decomposes iteratively the desired task into a set of sub-tasks until the resulting set of tasks consists only of atomic (primitive) tasks, which can be executed directly by invoking some atomic operations. During each iteration of task decomposition, it is tested whether certain given conditions are violated (e.g. exceeding a certain amount of resources) or not. The planning problem is successfully solved, if the desired complex task is decomposed into a set of primitive tasks without violating any of the given conditions. An approach of using HTN planning in the realm of Web Services was proposed in [41a], facilitating the planning system SHOP2, which uses DAML-S Web service descriptions for automatic service composition.

Each state of the world is represented by a set of atoms, and each action corresponds to a deterministic state transition. However, HTN planners differ from classical AI planners in what they plan for, and how they plan for it. The objective of an HTN planner is to produce a sequence of actions that perform some activity or task. The description of a planning domain includes a set of operators similar to those of

classical planning, and also a set of methods describing how to decompose a task into sub-tasks (smaller tasks). Given a planning domain, the description of a planning problem will contain an initial state like that of classical planning—but instead of a goal formula; the problem specification will contain a partially ordered set of tasks to accomplish. Planning proceeds by using the methods to decompose tasks recursively into smaller and smaller sub-tasks, until the planner reaches primitive tasks that can be performed directly using the planning operators. For each non primitive task, the planner chooses an applicable method, instantiates it to decompose the task into sub-tasks, and then chooses and instantiates methods to decompose the sub-tasks even further. If the plan later turns out to be infeasible, the planning system will need to backtrack and try other methods. [69a]

SIRIN ET AL. presents in [66a] a prototypical implementation of a service composer which bases on the HTN-planner SHOP2 [69a]. The prototype transforms atomic services described in OWL-S into primitive HTN-tasks and composed services also described in OWL-S into non primitive HTN-tasks. As user request an initial state and a partially ordered set of services has to be provided instead of the goal of the user. This means the user has to compose the needed services on a high level on his own. If the services in the user request are assembled out of other services then they are decomposed by the prototype into atomic services. This decomposition will be optimised according to a given optimization rule in case several different decompositions are possible. This approach is more powerful than the Meteor-S prototype, since it can handle a replacement of two atomic activities by a superseding atomic activity. However like the Meteor-S approach the prototype presented by SIRIN ET AL. is very limited according its composition power since it heavily relies on given human modeled service compositions. But this limitation in power has naturally a positive effect on composition complexity, making HTN-planning relatively fast in comparison to the forthcoming approaches.

In the SHOP2 planner is applied for automatic composition of Web services, which are provided with DAML-S descriptions. SHOP2 is a Hierarchical Task Network (HTN) planner. The concept of task decomposition in HTN planning is very similar to the concept of composite process decomposition in DAML-S process ontology. The HTN planner is more efficient than other planning language, such as Golog.

SHOP2 is a domain-independent HTN planning system, which won one of the top four awards out of the planners that competed in the 2002 International Planning Competition. HTN planning is an AI planning methodology that creates plans by task decomposition. HTN planners differ from classical AI planners in what they plan for, and how they plan for it. The objective of an HTN planner is to produce a sequence of actions that perform some activity or task. The description of a planning domain includes a set of operators similar to those of classical planning, and also a set of methods, each of which is a prescription for how to decompose a task into subtasks. Planning proceeds by using methods to decompose tasks recursively into smaller and smaller subtasks, until the planner reaches primitive tasks that can be performed directly using the planning operators.

4.5.2 Semi-automated Composition of Services

Composition of services is a step by step assembling of self-contained services to process with a specific capability. Each assembling step consist at least of the following tasks:

- Service Discovery is the task of finding a service. Since services can be provided by various providers all over the world, finding of services is a non-trivial task. The challenges in finding proper services are the potentially huge number of service repositories as well as of different languages, law spaces, ontologies and business models. Searching for services on the world scale is the worst case according the named challenges.
- Service Matchmaking denotes the task of selecting a proper service from a list of existing services for assembling. Due to the potentially huge amount of services a matchmaking mechanism should have a high precision in selecting a proper service. This is important to avoid overwhelming the modeler with unsuitable services. Furthermore, the variability of possible service capabilities is high due to the high complexity of the world and the various business scenarios. This also demands from a matchmaking mechanism a high recall, especially for rare and special services. High recall means, that ideally all matching services are found. In case capabilities of rare or special services are needed, high recall is important to ensure that at least one of the existing and matching services is retrieved.
- Data-/Control-flow Linkage defines the logical sequence of services and specifies the needed data transfers. Depending on the desired capability of the final service composition and the

capabilities of the existing services the control-flow of the services may vary from a simple sequence to complex control structures including branches, parallelism and loops. Furthermore data-flow can be in the simplest case just an opaque propagation of the output of a service to the input of another service. However in most cases data-flow will imply transformation of data and data structures to overcome different representations of the same or similar, but matching concepts.

5 Methodology for the design of pSHIELD Ontology

5.1 Overview

A sound framework for an effective exploitation of ontologies in ES should provide a formal methodology to build, verify and maintain an ontology.

As a matter of fact, in order to make available large-scale, high quality domain ontologies, effective and usable methodologies are needed to facilitate the process of ontology building. Ontology building is a task that pertains to the ontology engineers, that we classify as knowledge engineers (KE) and domain experts (DE). Even though automatic ontology learning methods (such as text mining) significantly support ontology engineers, speeding up their task, there is still the need of a significant manual effort, in the integration and validation of the automatically generated ontology.

Existing ontology building methods only partly are built capitalizing the large experience that can be drawn from widely used standards in other areas, like software engineering and knowledge representation. In this project, we shall embrace a methodology for ontology building derived from a well-established and widely used software engineering process, the Unified Software Development Process

This is a novel approach to large-scale ontology building that takes advantage of the Unified Process (UP) and the Unified Modeling Language (UML). This choice makes ontology building an easier task for modellers familiar with these techniques: each phase of the method fits in the UP, providing a number of consolidated steps that guide the process of ontology development. UML has been already shown to be suitable to this end, confirming its nature of rich and extensible language. What distinguishes the UP and the ontology building methodology from the other methodologies, respectively for software and ontology engineering, is their use-case driven, iterative and incremental nature.

The metholdology is use-case driven since it does not aims at building generic domain ontologies, but its goal is the production of ontologies that serve its users, both humans and automated systems (e.g. semantic web services, intelligent agents, etc.), in a well defined application area. Use cases are the first diagrams that drive the exploration of the application area, at the beginning of the ontology building process.

The nature of the process is iterative since each iteration allows the designer to concentrate on part of the ontology being developed, but also incremental, since at each cycle the ontology is further detailed and extended.

Following the UP, in the metholdology we have cycles, phases, iterations and workflows. Each cycle consists of four phases (inception, elaboration, construction and transition) and results in the release of a new version of the ontology. Each phase is further subdivided into iterations. During each iteration, five workflows (described in the next subsections) take place: requirements, analysis, design, implementation and test. Workflows and phases are *orthogonal* in that the contribution of each workflow to an iteration of a phase can be more or less significant: early phases are mostly concerned with establishing the requirements (identifying the domain, scoping the ontology, etc.), whereas later iterations result in additive increments that eventually bring to the final release of the ontology (Figure 5.1). Notice that, as illustrated in the figure, more than one iteration may be required to complete each of the four phases. This scheme follows faithfully the Unified Process.

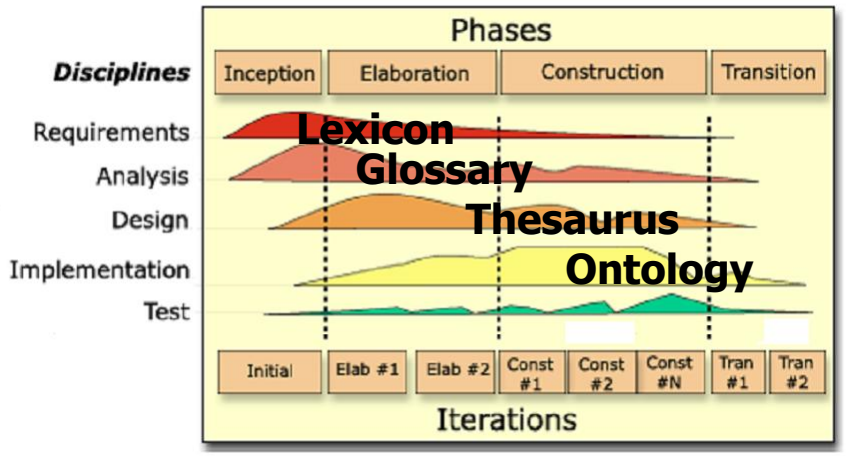

Figure 5.1 Mapping onto workflows and phases of UP

The first iterations (inception phase) are mostly concerned with capturing requirements and partly performing some conceptual analysis. Neither implementation nor test is performed. During subsequent iterations (belonging to the elaboration phase) analysis is performed and the fundamental concepts are identified and loosely structured. This may require some design effort and it is also possible that the modellers provide a preliminary implementation in order to have a small skeletal blueprint of the ontology, but most of the design and implementation workflows pervade iterations in the construction phase. Here some additional analysis could be still required aiming at identifying concepts to be further added to the ontology. During the final iterations (transition phase), testing is heavily performed and the ontology is eventually released. In parallel, the material necessary to start the new cycle, that will produce the next version of the ontology, is collected. As shown in Figure 5.1, and detailed in the next sections, at each iteration different workflows come into play and a richer and more complete version of the target ontology is produced. The incremental nature of the methodology requires first the identification of the relevant terms in the domain, gathered in a lexicon; then the latter is progressively enriched with definitions, yielding a glossary; adding to it the basic ontological relationships allows a thesaurus to be produced, until further enrichments and a final formalization produces the sought reference ontology.

In the following subsections each ontology building workflow is described in detail.

5.2 The Requirements Workflow

Requirements capture is the process of specifying the semantic needs and the knowledge to be encoded in the ontology. The essential purpose of this workflow is to reach an agreement between the modellers, the knowledge engineers, and the final users, represented by the domain experts. During the first meetings, knowledge engineers and domain experts establish the guidelines for building the ontology. The first goal is the identification of the objectives of the ontology users. To this end, it is necessary to:

- determining the domain of interest and the scope, and
- defining the purpose.

These objectives are achieved by:

- writing one or more storyboards
- creating an application lexicon
- identifying the competency questions, and
- the related use cases.

5.2.1 Determining the domain of interest and the scope.

Delimiting the domain of interest is a fundamental step to be performed, aiming at focusing on the appropriate fragment of reality to be modelled. If the domain is large, one or more sub-domains may also be determined. Defining the scope of the ontology consists in the identification of the most important concepts to be represented, their characteristics and granularity. For this purpose, a set of ontological commitments are required, bringing some part of the domain into focus at the (required and expected) expense of other parts. These ontological commitments are not incidental: they provide a guidance in deciding what aspects of the domain are relevant and what to ignore. The ontological commitment can be seen as “a mapping between a language and something which can be called an ontology”. This allows one to preliminarily identify terms as representatives of ontology concepts. Usually at this stage modellers have only a vague idea of the role each concept will play, i.e., their semantic interconnections, within the ontology. If necessary, they can informally annotate these ideas for further development during subsequent iterations.

5.2.2 Defining the purpose (or motivating scenario)

The reason for having an ontology, its intended uses, and the kinds of users must be established. In the pSHIELD, the goal of the ontology is to provide a support for semantic interoperability between entities at different layers.In particular, we envisage three basic uses of the developed ontology:

- Ontology-based search and retrieval of services (discovery);
- Ontology-based reconciliation of data messages exchanged between entities
- Reasoning about configuration and SPD metrics, as defined in the ontology in terms of composition rules, to (dinamically) find new configuratione of the system at run time or design time.

5.2.3 Writing a storyboard.

In this step domain experts are asked to write a panel or series of panels outlining the sequence of the activities that take place in a particular scenario. Storyboards model contexts and situations is a narrative way that can be used in the next steps.

5.2.4 Creating the application lexicon.

The storyboard can be also used to extract the terminology of domain experts, building a preliminary version of the application lexicon. This task can be supported by using some automatic tools to extract knowledge from documents, such as OntoLearn or other methodologies. An application lexicon is more specific than a domain lexicon, but both are necessary to accomplish an effective ontology.

In pSHIELD, the application lexicon has been mainly extracted by the applicable documents listed in section 3.

5.2.5 Identifying the competency questions.

Competency questions are questions an ontology must be able to answer. They are identified through interviews with domain experts, brainstorming, an analysis of the document base concerning the domain, etc. The questions do not generate ontological commitments, but are used during the *test workflow* to evaluate the ontological commitments that have been made. The competency questions are more significant when the use of the ontology will be mainly for querying and discovering rather than for reconciliation.

5.2.6 Use-case identification and prioritization.

In this methodology. competency questions are taken into account through use-case models. A use-case model, that contains a number of use case diagrams, serves as a basis to reach an agreement between the users (i.e., who require the ontology) and the modellers. In the context of ontologies, use cases correspond to *knowledge paths* through the ontology to be followed for answering one or more competency questions. Although they are to be specified during the analysis and design workflows, it is necessary to prioritize and package (i.e. group) them during requirements. The result will help dictate which use cases the team should focus on during early iterations, and which ones can be postponed.

5.3 The Analysis Workflow

The conceptual analysis consists of the refinement and structuring of the ontology requirements identified in previous section. The ontological commitments derived from the definition of scope are extended, by reusing existing resources and through concept refinement. The application lexicon will be enriched through the definition of a more general domain lexicon, then definitions will be added to produce the *Reference Glossary*.

5.3.1 Considering reuse of existing resources: identification of the domain lexicon

The domain lexicon is defined as the terminology used in the domain of interest, extracted by analyzing a corpus of existing resources. The analysis is mainly based on external resources, such as documents, standards, glossaries, thesauri, legacy computational lexicons and available ontologies. This task, like in the case of the application lexicon, can be supported by automatic tools. The description of this activity adheres to the view of linguistic ontology in which concepts, at least the lower and intermediate levels, are anchored to texts, i.e. they have a counterpart in natural language.

In order to build the Embedded Systems domain lexicon, in pSHIELD we have mainly considered as resources: the applicable documents listed in section 3.

A statistical analysis shall be done in a corpus of documents of reference to identify frequently used terms to be included in the domain lexicon. The domain experts shall decide to include, in this lexicon, all the terms present in, for instance, at least two standards. Some other terms, present in only one resource, can be included after approval from a wider panel of experts. After this activity, the domain lexicon shall contain a number of terms (including synonyms).

5.3.2 Modelling the application scenario using UML diagrams.

The goal of this activity is to model the application scenario and better specify the Use Case Diagrams, drawn in the requirement workflow, with the aid of Activity and Class Diagrams. UML diagrams represent a model of the application and will be used for the validation of the ontology. In principle, all the classes, actors, and activities modeled in UML must have a corresponding concept in the ontology.

5.3.3 Building the glossary.

A first version of a glossary of the domain of interest has to be built merging the application lexicon and the domain lexicon. During the merge of the two lexicons we can organize all the concepts in two major areas: the intersection area and the disjoint area (Figure 5.2). Then we use the following “inclusion policy”: the glossary should include all the concepts coming from the intersection area and, after the domain experts approval, some concept belonging to the disjoint area. The output is a reference lexicon that will grow into a glossary by associating one or more definitions to each term. The definitions should be selected from knowledgeable sources and agreed among domain experts.

Refer to “Annex 1 – pSHIELD Glossary” for the pSHIELD glossary built during this phase.

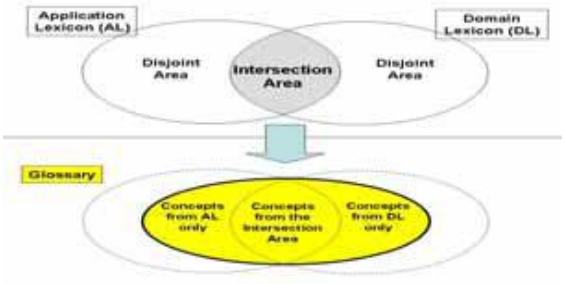

Figure 5.2 Glossary building

5.4 The Design Workflow

The main goal of this workflow is to give an ontological structure to the set of terms gathered in the Glossary. The refinement of entities, actors and processes identified in the analysis workflow, as well as the identification of their relationships, is performed during the design workflow.

5.4.1 Categorising the concepts.

Each concept is categorized by associating a "kind" to it. Such *kinds* should include the major ontological categories, according to proposals of *upper ontologies*, or *meta-ontologies*,

A partial outcome of this phase is provided by Table 1

Concept	Relation	Related concept
Audit	Specializes	SPD Functionality
Authentication	Specializes	SPD Functionality
Availability	Specializes	SPD Attribute
Confidentiality	Specializes	SPD Attribute
Cryptographic	Specializes	SPD Functionality
Dependability	Specializes	SPD Concept
Error	Specializes	Threat
Failure	Specializes	Threat
Fault	Specializes	Threat
Forecasting	Specializes	Mean
Identification	Specializes	SPD Functionality
Integrity	Specializes	SPD Attribute
Maintainability	Specializes	SPD Attribute
Mean	Generalizes	Prevention
Mean	Generalizes	Tolerance
Mean	Generalizes	Forecasting
Mean	Generalizes	Removal
Prevention	Specializes	Mean
Reliability	Specializes	SPD Attribute
Removal	Specializes	Mean
Safety	Specializes	SPD Attribute
Security	Specializes	SPD Concept
SPD Attribute	Generalizes	Availability
SPD Attribute	Generalizes	Reliability
SPD Attribute	Generalizes	Integrity
SPD Attribute	Generalizes	Safety
SPD Attribute	Generalizes	Confidentiality
SPD Attribute	Generalizes	Maintainability
SPD Concept	Generalizes	Dependability
SPD Concept	Generalizes	Security
SPD Functionality	Generalizes	Identification
SPD Functionality	Generalizes	Authentication
SPD Functionality	Generalizes	Audit
SPD Functionality	Generalizes	Cryptographic
Threat	Generalizes	Fault
Threat	Generalizes	Error
Threat	Generalizes	Failure
Tolerance	Specializes	Mean

Table 1 pSHIELD concept categorization

5.4.2 Refining the concepts and their relations.

At this stage, concepts are organised by introducing formal relations among them. between sets of synonyms identified in the previous phase. A first step consists in organizing the concepts in a taxonomic hierarchy through the generalization (i.e., kind-of or is-a) relation. To this end, three main approaches are known in the literature:

- top-down (from general to particular)
- bottom-up (from particular to general)
- middle-out (or combined), which consists in finding the salient concepts and then generalizing and specializing them. This approach is considered to be the most effective because concepts “in the middle” tend to be more informative about the domain.

The resulting taxonomy can be extended with other relations, i.e., part-of and association. The outcome of this step is Thesaurus, structured according the UML class diagram relations: generalization (IsA), aggregation (Part-Of) and association. In parallel, the actual UML diagrams can be built.

The detailed outcome of this phase can be found in the pSHIELD ontology (internal deliverable D5.1, whose access is regulated by the consortium)

5.5 The Implementation Workflow

The purpose of this workflow is to perform the final building step, by formalize defining the actual ontology in a formal language. The structure of the ontology will be the one given in the enriched Thesaurus, but here the different elements will be formally represented. To this end, the Ontology Web Language (OWL) proposed by the W3C shall be adopted.

The outcome of this workflow is the implementation model, i.e., a reference ontology encoded in OWL.

The detailed outcome of this phase can be found in the pSHIELD ontology (internal deliverable D5.1, whose access is regulated by the consortium)

5.6 TheTest Workflow

The test workflow allows to verify that the ontology correctly implements the requirements produced in the first workflow. We envisage two kinds of test.

The first concerns the coverage of the ontology with respect to the application domain. In particular, the domain experts are asked to semantically annotate the UML diagrams, representing the application scenario, with the ontology concepts. (This test is particularly relevant for ontologies used in ontology-based reconciliation of messages).

The second kind of test concerns the competency questions and the possibility to answer them by using concepts in the ontology. Such questions will trigger a traversal of the ontology that will produce proper concepts. Competency questions represent a good test for ontologies to be used in search and discovery of resources

This phase is considered out of the scope of pilot SHIELD project

6 pSHIELD Semantic Models

6.1 Introduction

The described methodology is the most valuable chain to produce ontology and meta-models for a specific scenario (in this case the context of Embedded Systems).
The problem is: given a clear procedure on how to build ontology, what are we supposed to describe in it?
Starting from that, we can affirm that the context is the one of Embedded Systems; in particular the more specific contest are the SPD functionalities provided by their interaction/composition.
The main objective of our approach with semantic models are:

1. the *abstraction* of the *real word* from a technology-dependent perspective into a technology-independent representation.

2. the *representation* of *functional properties* by means of ontology as well

3. the *identification* of the *relations* between real/structural and functional world.

So, as depicted in Figure 6.1, the problem of modeling SPD in the context of ES is reduced to the formulation of three different meta-models describing: i) structure, ii) functions, iii) relations between structure and functions.

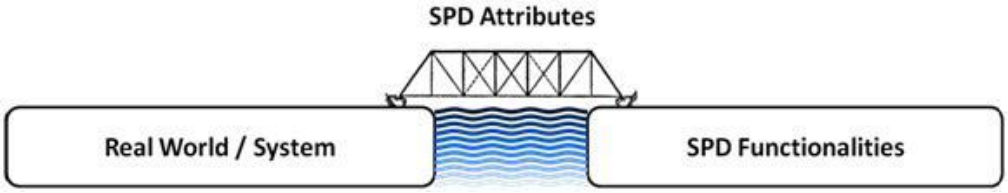

Figure 6.1 Proposed approach to model SPD for ES

The bridge has been built thank to the introduction of a third metamodel taking into account the atomic attributes that are impacted in this context and to map them in these two worlds, thus creating relations.

For each of the three models, a justification and a detailed description is provided in the prosecution of the document, while the global pSHIELD meta-model is depicted in D5.1. All the relevant concepts (classes) are highlighted: they are the basic and exhaustive building blocks by which it is possible to reach the pSHIELD objectives. In particular:

- For the structural ontology these classes have been selected: `System`, `Element`, `Hardware`, `SPD Component`,
- For the functional ontology these classes have been selected: `SPDFunctionality`, `GeneralFunctionality`, `Connector`, `SPDCompositionSpecification`
- For the attribute ontology these classes have been selected: `SPDConcept`, `SPDAttribute`, `SPDThreat`, `SPDMean`

7 Ontology Implementations

The OWL Web Ontology Language is designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with formal semantics. OWL has three increasingly expressive sublanguages: OWL Lite, OWL DL, and OWL Full.

The basic reasons for decision to use of OWL for modelling in pSHIELD are:

1) OWL extends all other languages like XML, RDF, and RDF-S. Actually, OWL has been developed on top of the existing XML and RDF standards, which did not appear adequate for achieving efficient semantic interoperability.

    a) E.g. in XML and XML Schema same term may be used with different meaning in different contexts, and different terms may be used for items that have the same meaning.

    b) E.g. RDF and RDF-S address some problem by allowing simple semantics to be associated with identifiers. With RDFS, one can define classes that may have multiple subclasses and superclasses, and can define properties, which may have subproperties, domains, and ranges. However, in order to achieve interoperation between numerous, autonomously developed and managed schemas, richer semantics are needed, like disjoints and cardinality of relations.

2) OWL adds more vocabulary for describing properties and classes, relations between classes, cardinality, equality, richer typing of properties, characteristics of properties and enumerated classes, and all available in three increasingly expressive and increasingly complex sublanguages (Lite, DL, Full) designed for use by specific communities of implementers and users.

3) OWL is well-known widely used open W3C standard with very good support and promising potential and real usage in several industry applications.

4) OWL has wide support of modelling tools, platforms, and reasoners.

5) Previous languages could express (in most cases) the same things, but for some of them OWL provide direct solution by a predefined type of predicates.

6) There are several well-known mechanisms for expressing OWL-Lite and OWL-DL ontologies to stay on decidable level, where Description Logic (DL) could be used correctly.

7) OWL language has proved its potential to use for modelling of semantic interoperability in several middleware-based applications and domains.

In pSHIELD the same OWL-based framework can be used for representation of context, device descriptions (capabilities), descriptions of middleware components, services, security aspects, with several specific goals such as:

1. Semantic reasoning based on ontology model may carry out a reconciliation of heterogeneous formats of parameters exchanged between different layers (also suitable for interaction with legacy agents).

2. The semantic characterization of the behavioral aspect of components makes it suitable for an agent to determine “what the service does”.

3. The semantic characterization of the composition of functionalities and of the relations among them makes it suitable for an agent to reason about SPD metrics of the current configuration and – if needed - to carry out reconfigurations of the system at run-time, by means of rule-based combination / composition of components and SPD technologies, in order to achieve the new intended values for SPD metrics.

The ontology has many merits, of which the most notable are the excellent extensibility, and high expression power. Many systems in the “ubiquitous” and embedded environments are developed using

DL-based ontologies and used with DL-based reasoning. Usually, ontologies are used for modelling context that the systems should collect and analyze. A pure DL-based approach, however, has certain limitations in a context environment. OWL and other ontology languages based on Description Logic cannot properly handle rules expressed in Horn-Logic. Hence, to ensure syntactic and semantic interoperability on device level (e.g. “low-level” ontologies), *SWRL* (Semantic Web Rule Language) or *Jena* can be used for expressing rules.

The detailed OWL representation of the pSHIELD ontology can be found internal deliverable D5.1, whose access is regulated by the consortium.

Overview

Given that the SPD domain in ES is captured by a (number of) ontologies, automatic reasoning is enabled in order to support several features of the pSHIELD framework

Broadly speaking, inference engines (or reasoners), shall enable interoperability within Middleware Layer and rule based discovery and composition within Overlay Agents (see Figure 7.1)

1. Semantic reasoning based on ontology model may carry out a reconciliation of heterogeneous formats of parameters exchanged between different layers (also suitable for interaction with legacy agents).

2. The semantic characterization of the behavioral aspect of components makes it suitable for an agent to determine “what the service does”.

3. The semantic characterization of the composition of functionalities and of the relations among them makes it suitable for an agent to reason about SPD metrics of the current configuration and – if needed - to carry out reconfigurations of the system at run-time, by means of rule-based combination / composition of components and SPD technologies, in order to achieve the new intended values for SPD metrics.

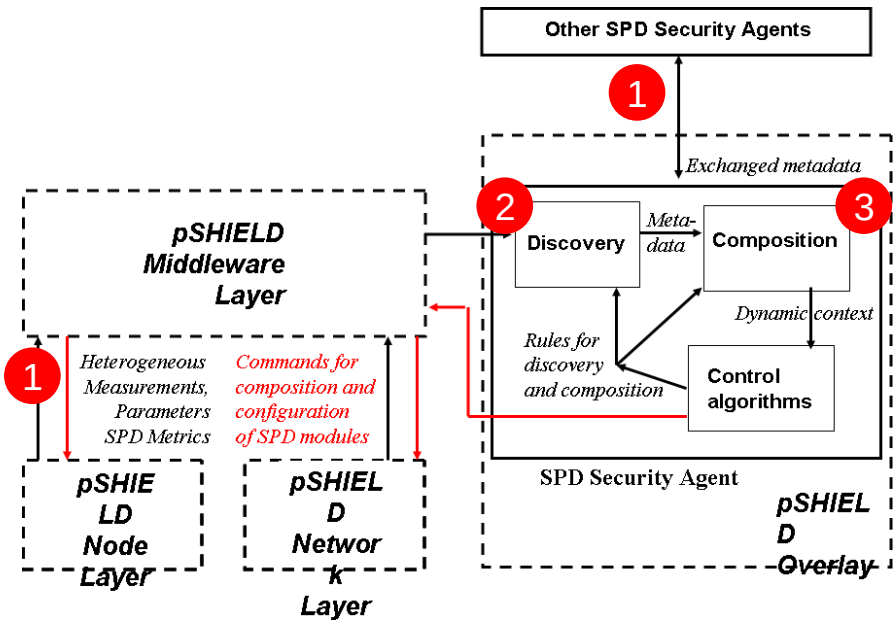

Figure 7.1 Exploitation of semantic inference in the framework

7.1 Semantic reconciliation

7.1.1 Semantic clashes

When nodes have no semantic capabilities due to computational / power limits, semantic reasoning based on ontology model may carry out a reconciliation of heterogeneous formats of parameters exchanged between different layers (also suitable for interaction with legacy agents).

With regards to differences between formats and structures, two types of semantic clashes, i.e. mismatches, are detected in pSHIELD framework:

- Loseless: can be solved without loss of information
- Lossy: any (faithful) transformation will cause a loss of information

Here is a list of identified mismatches for each category

- Loseless mismatches
  - *Naming*: different labels for the same content
  - *Attribute granularity*: the same information is splitted in a different number of attributes
  - *Structure Organization*: different structures and organization of the same content
  - *Subclass-Attribute*: an attribute with predefined value set is represented by a set of subclasses
  - *Schema-Instance*: data hold schema information
  - *Encoding*: different format of data or unit of measure
- Lossy mismatches
  - *Content*: different content denoted by the same concept (typically expressed by enumeration)
  - *Coverage*: the presence/absence of information
  - *Precision*: the accuracy of information
  - *Abstraction*: level of specialisation refinement of the information

7.1.2 Clash removal

For a subset of identified semantic clashes, we create rules templates that let us reconcile, i.e.remove, the clashes between a Source and Destination that exchange messages

- Naming structuring template
  - Allows the path from the Destination to be instantiated with the value carried by the path from the Source
  - No manipulation of the value is needed (only type casting is allowed)
  - Solved clashes: Naming, Structuring organization
- GranularitySplit rule template

- Allows the string value of a path from the Source to be decomposed into several strings in order to instantiate paths from the Destination. Substrings are found by identifying separators in the Source path.
- Solved clashes: Attribute Granularity when the Source is less structured than the Destination

- GranularityMerge rule template
  - Allows the value of a path from the Destination to be instantiated with the concatenation of the values of paths from the Source and the specified string separators
  - No manipulation of the value is needed (only type casting is allowed)
  - Solved clashes: Attribute Granularity when the Source is more structured than the Destination
- SubClass-Attribute rule template
  - Allows to instantiate a structure (Class) from the Destination organized into n paths with values of paths from the Source, if a given path from the Source is true.
  - Solved clashes: SubClass-Attribute
- Encoding rule template
  - Allows the path from the destination to be instantiated with the value carried by the path from the Source converted by using a conversionRate
  - Solved clashes: Encoding

7.1.3 Candidate implementation architecture

A semantic engine (reasoner) acts as semantic hub between interacting entities, based on the shared ontology and a set of reconciliation rules (see Figure 7.2)

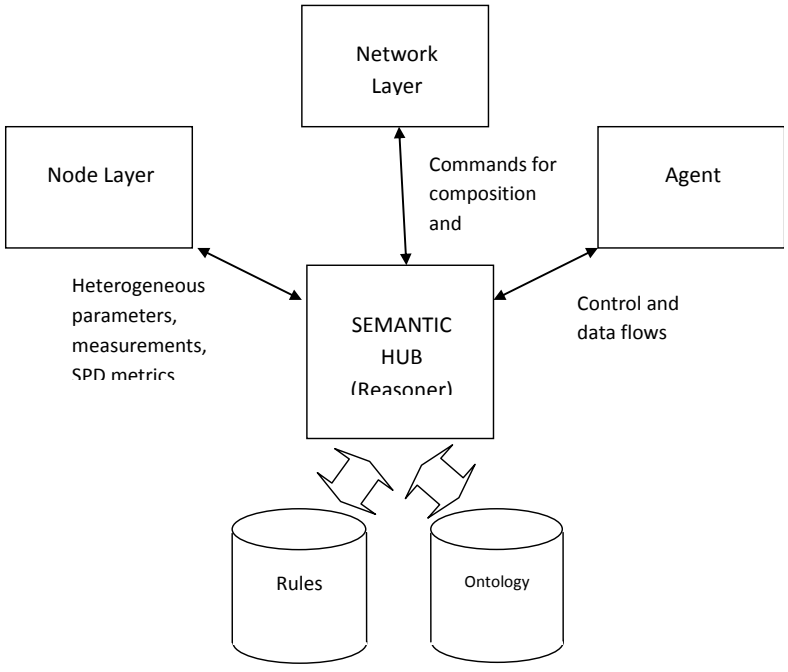

Figure 7.2 Semantic hub architecture

The implementation of the semantic hub is based on an inferential engine provided by the Jena framework, in a general purpose rule engine configuration (Figure 7.3)

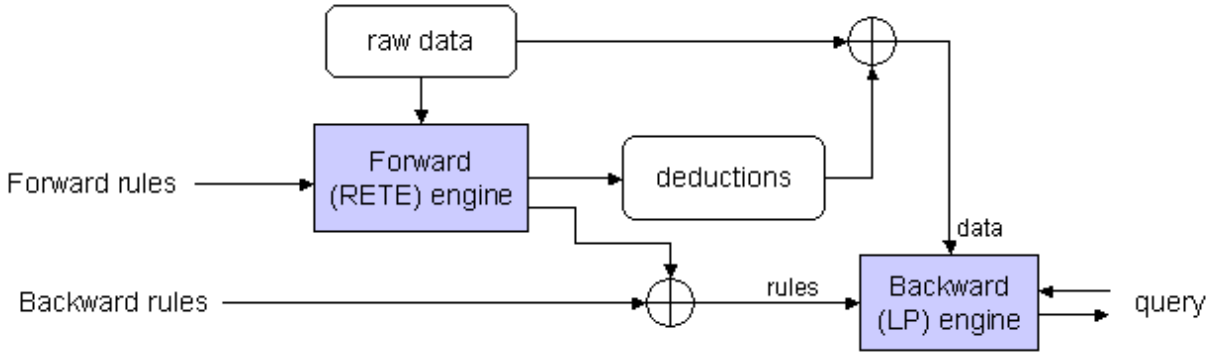

Figure 7.3 Jena hybrid rule engine

7.2 Semantic discovery & composition

While semantid discovery will be addressed in D5.2, now we will focus on semantic composition.

Semantic (inferential) engines, enabled by SPD ontologies, may be profitably exploited to carry out a number of basic functionalities supporting composability based on SPD metric in the pSHIELD framework. In this novel approach to the determination of semantic enabled SPD composition, an automatic reasoner can infer the overall level of SPD metrics resting upon model axioms and declarative rules.

First of all, the pSHIELD framework can leverage the semantic support to composability during the two following stages:

- Analysis: at run time (online), the semantic engine oversees the current value of the overall SPD level as the state of the system evolves in time
- Synthesis:at design time (offline) the semantic engine helps along the configuration of a system architecture, by discovering proper combinatons of SPD modules, according to the corresponding semantic model of modules and composability rules picked out from an offline repository (catalogue); at run time (online), changes in the state of the system trigger the semantic engine to devise new compositions, based on knowledge of modules that at the moment are active in the system (possibly discovered at run time), in order to guarantee the prearranged overall SPD level.

With regards to the operations that have been identified in the proposal for the aggregation of SPD metrics, and to the requirements of ontological SPD modeling, a number of suitable mixes of rules and ontology axioms can be used to develop the aggregation features.

MIN operation: can be modeled after the concept of ontological functional features (behaviour of the SPD module) and ontological composition features (all the SPD modules are present at the same time, with weak correlation between the two). Antecedent and consequent of the declarative rule look like the following, given that we are assessing the composite SPD level “L” of a component “X” which exposes 2 distinct SPD functionalities S1 and S2:

$$hasSPDFunct(X,S1) \wedge hasSPDFunct(X,S2) \wedge hasSPDLevel(\mathrm{S1},\mathrm{L1}) \wedge hasSPDLevel(\mathrm{S2},\mathrm{L2}) \wedge MINOp(L1,L2,L) \Rightarrow hasCompositeSPDLevel(X,L)$$

OR and MEAN operation: can be modeled in a similar fashion, but the ontological composition feature conveys a stronger relation between the SPD modules; as a matter of fact, every SPD function combine to bring a higher value of overall SPD level. The relation between several SPD functionalities is captured at different extent of complexity by an ontological composition feature, in order to render OR operation (one SPD function “includes” a second one) or MEAN operation (SPD functions act as a sort of alternative)

$$hasSPDFunct(X,S1) \wedge hasSPDFunct(X,S2) \wedge hasSPDLevel(\mathrm{S1},\mathrm{L1}) \wedge hasSPDLevel(\mathrm{S2},\mathrm{L2}) \wedge includesSPD(S1,S2) \wedge OROp(L1,L2,L) \Rightarrow hasCompositeSPDLevel(X,L)$$

$$hasSPDFunct(X,S1) \wedge hasSPDFunct(X,S2) \wedge hasSPDLevel(\mathrm{S1},\mathrm{L1}) \wedge hasSPDLevel(\mathrm{S2},\mathrm{L2}) \wedge alternativeSPD(S1,S2) \wedge MEANOp(L1,L2,L) \Rightarrow hasCompositeSPDLevel(X,L)$$

Redundancy: can be modeled by means of ontological axioms of subclassing or (application) ontological features of equivalence. In our system model, we can state that a functionality S2 is the equivalent of another S1, or that a functionality S2 is a specialization of S1 (concept of inheritance), and consequently that S2 can act as S1. Of course, a composite value of SPD metric can be linked to the redundancy as it’s recognized by the reasoner

$$hasSPDFunct(X, S1) \wedge hasSPDFunct(X, S2) \wedge hasSPDFunctionalEquivalence(S1, S2)...$$
$$\Rightarrow hasSPDRedundancy(X, S1) \wedge hasCompositeSPDLevel(X, L)$$

MINOp, MEANOp, OROp are examples of function objects (functors), that are customizable operators, possibly partially based on operators provided as built-in by the inferential engine. We can think of them as application ready-for-use functions, automatically made available by meta-relations between SPD functions inside the model, further extensible to fit new instances of meta-relations.

The clauses inside the rules may be rendered by means of automatic expansion of predicates, based on the constraints that are stated in the model.

At last, if the variables in the clauses are bound, the reasoner works as analyzer, and the compoiste level appear in the consequent clauses since it's unknown. If the variables are unbound, the reasoner acts as synthesizer: in this case, the composite level is a target (known) value, so it must appear in the antecedent clauses, whereas composition of modules and functionalities are the unknown data to find out.

8 Conclusions

In order to address SPD composability in embedded systems, we think that we can benefit the ability of semantic technologies to process explicit knowledge of a domain in an effective way. SPD composability can be reformulated in a problem of interoperability, and for this purpose we leverage the features of validation, analysis and harmonization provided by ontologies and semantic reasoning

We argue that interoperability issues can be divided onto three levels of integration, the syntactic, structural, and semantic level. And we try out a holistic approach in which all the three levels of integration are working together by means of a unifying ontology
The framework we propose is able to address the first two issues by means of reconciliation, and the latter by means of the semantic annotation of the behavioral aspect of components in an ES
Further work involves several areas, among which:

- Automatic translation of constraints of the model in reasoner's rules (clauses)
- support to the customization of generic SPD metrics
- Investigation on completeness issues in reasoner's solution
- Assessment of performance of inference process

8.1 References

[Aben, 1993] Aben, M. (1993). Formally specifying reusable knowledge model components. Knowledge Acquisition Journal, 5:119–141.

[Bergamashi et al., 1999] Bergamashi, Castano, Vincini, and Beneventano (1999). Intelligent techniques for the extraction and integration of heterogeneous information. In Workshop Intelligent Information Integration, IJCAI 99, Stockholm, Sweden.

[Borgida and Patel-Schneider, 1994] Borgida, A. and Patel-Schneider, P. (1994). A semantics and complete algorithm for subsumption in the classic description logic. JAIR, 1:277–308.

[Brachman, 1977] Brachman, R. (1977). What's in a concept: Structural foundations for semantic nets. International Journal of Man-Machine Studies, 9:127–152.

[Brickley and Guha, 2000] Brickley, D. and Guha, R. (2000). Resource description framework (rdf) schema specification 1.0. Technical Report PR-rdf-schema, W3C. http://www.w3.org/TR/2000/CR-rdf-schema- 20000327/.

[Chawathe et al., 1994] Chawathe, S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y.,

Ullman, J., and Widom, J. (1994). The TSIMMIS Project: Integration of Heterogeneous Information Sources. In Proceedings of IPSJ Conference, pages 7– 18.

[Collet et al., 1991] Collet, C., Huhns, M. N., and Shen, W.-M. (1991). Resource integration using a large knowledge base in carnot. IEEE Computer, 24(12):55–62.

[Genesereth and Fikes, 1992] Genesereth, M. and Fikes, R. (1992). Knowledge interchange format version 3.0 reference manual. Report of the Knowledge Systems Laboratory KSL 91-1, Stanford University.

[Gruber, 1991] Gruber, T. (1991). Ontolingua: A mechanim to support portable ontologies. KSL Report KSL-91-66, Stanford University.

[Gruber, 1993] Gruber, T. (1993). A translation approach to portable ontology specifications. Knowledge Acquisition, 5(2).

[Guarino, 1998] Guarino, N. (1998). Formal ontology and information systems. In Guarino, N., editor, FOIS 98, Trento, Italy. IOS Press.

[Guarino and Giaretta, 1995] Guarino, N. and Giaretta, P. (1995). Ontologies and knowledge bases: Towards a terminological clarification. In Mars, N., editor, Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing, pages 25–32. Amsterdam.

[Horrocks, 1999] Horrocks, I. (1999). FaCT and iFaCT. In (Lambrix et al., 1999), pages 133–135.

[Jasper and Uschold, 1999] Jasper, R. and Uschold, M. (1999). A framework for understanding and classifying ontoogy applications. In Proceedings of the 12th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop. University of Calgary/Stanford University.

[Kim et al., 1995] Kim, W., Choi, I., Gala, S., and Scheevel, M. (1995). Modern Database: The Object Model, Interoperability, and Beyond, chapter On Resolving Schematic Heterogeneity in Multidatabase Systems, pages 521–550. ACM Press / Addison- Wesley Publishing Company.

[Kim and Seo, 1991] Kim, W. and Seo, J. (1991). Classifying schematic and data heterogeinity in multidatabase systems. IEEE Computer, 24(12):12–18.

[Lambrix et al., 1999] Lambrix, P., Borgida, A., Lenzerini, M., M¨oller, R., and Patel-Schneider, P., editors (1999). Proceedings of the International Workshop on Description Logics (DL'99).

[Lenat, 1998] Lenat, D. (1998). The dimensions of context space. Available on the web-site of the Cycorp Corporation. (http://www.cyc.com/publications).

[Mena et al., 1996] Mena, E., Kashyap, V., Illarramendi, A., and Sheth, A. (1996). Managing multiple information sources through ontologies: Relationship between vocabulary heterogeneity and loss of information. In Baader, F., Buchheit, M., Jeusfeld, M. A., and Nutt, W., editors, Proceedings of the 3rd Workshop Knowledge Representation Meets Databases (KRDB '96).

[Mitra et al., 1999] Mitra, P., Wiederhold, G., and Jannink, J. (1999). Semi-automatic integration of knowledge sources. In Fusion '99, Sunnyvale CA.

[Mitra et al., 2000] Mitra, P., Wiederhold, G., and Kersten, M. (2000). A graph-oriented model for articulation of ontology interdependencies. In Proc. Extending DataBase Technologies, EDBT 2000, volume Lecture Notes on Computer Science, Konstanz, Germany. Springer Verlag.

[Naiman and Ouksel, 1995] Naiman, C. F. and Ouksel, A. M. (1995). A classification of semantic conflicts in heterogeneous database systems. Journal of Organizational Computing, pages 167–193.

[OMG, 1992] OMG (1992). The common object request broker: Architecture and specification. OMG Document 91.12.1, The Object Management Group. Revision 1.1.92.

[Papakonstantinou et al., 1996] Papakonstantinou, Y., Garcia-Molina, H., and Ullman, J. (1996). Medmaker: A mediation system based on declarative specifications. In International Conference on Data Engineering, pages 132–141, New Orleans.

[Schreiber et al., 1994] Schreiber, A., Wielinga, B., Akkermans, H., Velde, W., and Anjewierden, A. (1994). Cml the commonkads conceptual modeling language. In et al., S., editor, A Future of Knowledge Acquisition, Proc. 8th European Knowledge Acquisition Workshop (EKAW 94), number 867 in Lecture Notes in Artificial Intelligence. Springer.

[Stuckenschmidt and Wache, 2000] Stuckenschmidt, H. and Wache, H. (2000). Context modelling and transformation for semantic interoperability. In Knowledge Representation Meets Databases (KRDB 2000). to appear.

[van Harmelen and Fensel, 1999] van Harmelen, F. and Fensel, D. (1999). Practical knowledge representation for the web. In Fensel, D., editor, Proceedings of the IJCAI'99 Workshop on Intelligent Information Integration.

[W3C, 1998] W3C (1998). Extensible markup language (xml) 1.0. W3C Recommendation. [W3C, 1999] W3C (1999). Resource descrition framework (rdf) schema specification. W3C Proposed Recommendation.

[Wache et al., 1999] Wache, H., Scholz, T., Stieghahn, H., and K¨onig-Ries, B. (1999). An integration method for the specification of rule–oriented mediators. In Kambayashi, Y. and Takakura, H., editors, Proceedings of the International Symposium on Database Applications in Non-Traditional Environments (DANTE'99), pages 109–112, Kyoto, Japan.

[Wiederhold, 1992] Wiederhold, G. (1992). Mediators in the architecture of future information systems. IEEE Computer, 25(3):38–49. standard reference for mediators. [Wiederhold, 1999] Wiederhold, G. (1999). Mediation to deal with heterogeneous data sources. In Vckovski, A., editor, Interop99, volume 1580 of Lecture Notes in Computer Science, Zurich, Switzerland. Springer.

[Wiener et al., 1996] Wiener, J., Gupta, H., Labio, W., Zhuge, Y., Garcia-Molina, H., and Widom, J. (1996). Whips: A system prototype for warehouse view maintenance. In Workshop on materialized views, pages 26–33, Montreal, Canada.

---

[1] Corcho, O., Fernandez-Lopez, M., Gomez-Perez, A., "Methodologies, tools and languages for building ontologies. Where is the meeting point?" Data&Knowledge Engineering 46, pp. 41-64, 2003.

[2] Genesereth, M.R., Fikes, R.E., "Knowledge Interchange Format. Version 3.0. Reference Manual." Stanford University, 1992.

[3] MacGregor, R., Bates, R., "The Loom Knowledge Representation Language". Technical Report ISIRS-87-188, USC Information Sciences Institute, Marina del Rey, CA, 1987.

[4] Connolly, D., van Harmelen, F., Horrocks, I., McGuinness, D., Patel-Schneider, P.F., Stein, L.A., DAML+OIL (March 2001) Reference Description. W3C Note, 2001.

[5] McGuinness, D., van Harmelen, F., "OWL Web Ontology Language Overview". W3C Recommendation, 2004.

[6] Kifer, M., Lausen, G., F-Logic: "A Higher-Order language for Reasoning about Objects, Inheritance, and Scheme", in Proc. of SIGMOD Conference 1989, pp. 134-146, 1989.

[7] Fensel, D., Decker, S., Erdmann, M., Studer, R., Ontobroker: "The Very High Idea", in Proceedings of the 11th International Flairs Conference (FLAIRS-98), Florida, 1998.

[8] Knutilla, A., et. al., “Process Specification Language: Analysis of Existing Representations”, NISTIR 6133, National Institute of Standards and Technology, Gaithersburg, MD, 1998.

[9] Gruber, T.R., A Translation Approach to Portable Ontology Specifications, in Knowledge Acquisition, 5(2), 1993.

[10] Motta E., Reusable Components for Knowledge Modelling. IOS Press, Amsterdam, Netherlands, 1999

[11] Chaudhri, V., Farquhar, A., Fikes, R., Karp, P., Rice, J. (1998), OKBC: A programming foundation for knowledge base interoperability, in Proceedings of AAAI’98, pp. 600-607, 1998.

[12] Karp, R., Chaudhri, V., Thomere, J., XOL: An XML-Based Ontology Exchange Language, Technical report, 1999.

[13] Sowa, J.F., Semantic networks, available online http://www.jfsowa.com/pubs/semnet.htm, 2002.

[14] Sowa, J.F., Knowledge Representation: Logical, Philosophical, and Computational Foundations, Brooks Cole Publishing Co., Pacific Grove, CA, 2000.

[15] R.E. Kent., Conceptual Knowledge Markup Language: The Central Core’, in: Twelfth Workshop on Knowledge Acquisition, Modeling and Management, 1999.

[16] TopicMaps.Org Authoring Group, XML Topic Maps (XTM) 1.0 TopicMaps.Org specification, 2001

[17] Yergeau, F., Bray, T., Paoli, J., Sperberg-McQueen, J.M., Maler, E., Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation, 2004.

[18] Manola, F., Miller, E., McBride, B., RDF Primer. W3C Recommendation, 2004.

[19] Fensel, D., Horrocks, I., van Harmelen, F., McGuiness, D., Patel-Schneider, P.F., OIL: Ontology Infrastructure to Enable the Semantic Web. IEEE Intelligent Systems, 16 (2), 2001.

[20] Stein, L.A., Connolly, D., McGuinness, D.,Annotated DAML Ontology Markup. Initial draft, http://www.daml.org/2000/10/daml-walkthru, 2000.

[21] Connolly, D., van Harmelen, F., Horrocks, I., McGuinness, D., Patel-Schneider, P.F., Stein, L.A., DAML+OIL (March 2001) Reference Description. W3C Note, 2001.

[22] Brickley, D., Guha, R.V., McBride, B., RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, 2004.

[23] Martin, D., et al., Bringing Semantics to Web Services: The OWL-S Approach, in Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), July 6-9, 2004, San Diego, California, USA.

[24] Berners-Lee T., Hendler J., Lassila O.: “The Semantic Web. A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities”. Scientific American, 284 (5), pp. 34-43, May 2001. http://www.sciam.com/

[25] Semantic Technology. TopQuadrant Technology Briefing, March 2004, http://www.topquadrant.com/tq_white_papers.htm

[26]. The World Wide Web Consortium. http://www.w3.org/

[27]. Gruber, Th. (1993) ‘A Translation Approach to Portable Ontology Specifi cations’ *Knowledge Acquisition* **5**(2), 199–220. http://citeseer.ist.psu.edu/viewdoc/download;jsessionid=E4FFFE129063D3DABCBA9822014E43E7?doi=10.1.1.101.7493&rep=rep1&type=pdf

[28]. Staab, S. and Studer, R. (eds) (2004) *Handbook on Ontologies*. Springer.

[29]. Baader, F., Calvanese, D., McGuinness, D., Nardi, D. and Patel-Schneider, P. (2003) *The Description Logic Handbook – Theory, Implementation and Applications*. Cambridge University Press.

[30]. Horrocks, I. (1998) 'Using an Expressive Description Logic: Fact or Fiction?' In Cohn, A.G., Schubert, L. and Shapiro, S.C. (eds), *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pp. 636–647, San Francisco, CA, June. Morgan Kaufmann. http://www.cs.ox.ac.uk/ian.horrocks/Publications/download/1998/kr98.pdf

[31]. Haarslev, V. and Möller, R. (2001) 'Racer System Description' *International Joint Conference on Automated Reasoning (IJCAR'01)* 18–23 June, Siena, Italy. Springer. http://users.encs.concordia.ca/~haarslev/publications/ijcar-2001-racer.pdf

[32]. Sirin, E. and Parsia, B. (2004) 'Pellet: An OWL DL Reasoner' In Haarslev, V. and Möller, R. (eds), *International Workshop on Description Logics (DL'04)*, pp. 212–213. Whistler, British Columbia, Canada, June. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.100.7584&rep=rep1&type=pdf

[33]. Tsarkov, D. and Horrocks, I. (2004) 'Efficient Reasoning with Range and Domain Constraints. In Haarslev, V. and Möller, R. (eds), *International Workshop on Description Logics (DL'04)*, pp. 41–50. Whistler, British Columbia, Canada, June. http://www.cs.ox.ac.uk/ian.horrocks/Publications/download/2004/TsHo04a.pdf

[34]. Sattler, U. (2005) Description Logic Reasoners. Information Management Group, School of Computer Science, University of Manchester. http://www.cs.man.ac.uk/~sattler/reasoners.html

[35]. Lutz, C., Areces, C., Horrocks, I. and Sattler, U. (2004) 'Keys, Nominals, and Concrete Domains *Journal of CEUR Workshop Proceedings* **49**, 170–179. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.2375&rep=rep1&type=pdf

[36]. Berners-Lee, T., Hendler, J. and Lassila, O. (2001) 'The Semantic Web' *Scientific American* **284**(5), 34–43. http://www-sop.inria.fr/acacia/cours/essi2006/Scientific%20American_%20Feature%20Article_%20The%20Semantic%20Web_%20May%202001.pdf

[37]. Horrocks, I. (2002) 'DAML+OIL: a Description Logic for the Semantic Web' *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* **25**(1), 4–9. http://www.cs.man.ac.uk/~horrocks/Publications/download/2002/ieeede2002.pdf

[38]. Patel-Schneider, P.F., Hayes, P. and Horrocks, I. (2004) *OWL Web Ontology Language Semantics and Abstract Syntax* W3C recommendation. The Worldwide Web Consortium. http://www.w3.org/TR/owl-semantics/

[39]. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D., Patel-Schneider, P.F. and Andrea Stein, L. (2004) *OWL Web Ontology Language Reference* W3C Recommendation. The Worldwide Web Consortium. http://www.w3.org/TR/owl-ref/

[40]. Horrocks, I., Patel-Schneider, P. and van Harmelen, F. (2003) 'From SHIQ and RDF to OWL: The Making of a Web Ontology Language' *Journal of Web Semantics*, **1**(1). http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.2.7039&rep=rep1&type=pdf

[41]. Noy, N.F., Sintek, M., Decker, S., Crubezy, M., Fergerson, R.W. and Musen, M.A. (2001) 'Creating Semantic Web contents with Protégé-2000' *IEEE Intelligent Systems* 1**6**(2), 60–71. http://icc.mpei.ru/documents/00000829.pdf

[42]. Kalyanpur, A., Parsia, B. and Hendler, J. (2005) 'A Tool for Working with Web Ontologies' *International Journal on Semantic Web and Information Systems* **1**(1), 36–49. http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.77.8076&rep=rep1&type=pdf

[43]. Liebig, Th. and Noppens, O. (2004) 'OntoTrack: Combining Browsing and Editing with Reasoning and Explaining for OWL Lite Ontologies' *Proceedings of the 3rd International Semantic Web Conference (ISWC'04)*, Hiroshima, Japan, November. http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.139.3270&rep=rep1&type=pdf

[44]. Knublauch, H., Fergerson, R.W., Noy, N.F. and Musen, M.A. (2004) 'The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications' *Proceedings of the Third International Semantic Web Conference (ISWC'04)*, Hiroshima, Japan. http://protege.stanford.edu/plugins/owl/publications/ISWC2004-protege-owl.pdf

[45]. McBride, B. (2001) 'Jena: Implementing the RDF Model and Syntax Specification' *Proceedings of the 2nd International Workshop on the Semantic Web (SemWeb'01)*, Hongkong, May.

[46]. Grau, B., Parsia, B., Sirin, E. and Kalyanpur, A. (2005) 'Automatic Partitioning of OWL Ontologies using *E*connections' In Horrocks, I., Sattler, U. and Wolter, F. (eds) *International Workshop on Description Logics (DL'05)*, Edinburgh, Scotland, July. National e-Science Centre. http://www.mindswap.org/2004/multipleOnt/papers/DLpartitioningRevised.pdf

[47]. Grau, B., Parsia, B. and Sirin, E. (2006) 'Combining OWL Ontologies using E-connections' *Journal of Web Semantics* **4**(1), 40–59. http://www.mindswap.org/2004/multipleOnt/papers/EconnJWS.pdf

[48]. Seaborne, A. (2004) *RDQL – a Query Language for RDF* W3C member submission. The Worldwide Web Consortium. http://www.w3.org/Submission/RDQL/

[49]. Kalyanpur, A., Parsia, B. and Sirin, E. Black box techniques for debugging unsatisfiable concepts' In Horrocks, I., Sattler, U. and Wolter, F. (eds) *International Workshop on Description Logics (DL'05)*, Edinburgh, Scotland, July. National e-Science Centre. http://ceur-ws.org/Vol-147/14-Kalyanpur.pdf

[50]. Parsia, B., Sirin, E. and Kalyanpur, A. (2005) 'Debugging OWL Ontologies' *Proceedings of the 14th International World Wide Web Conference (WWW'05)*, May. http://www.mindswap.org/papers/debuggingOWL.pdf

[51]. Liebig, Th. and Halfmann, M. (2005) 'Explaining Subsumption in ALEHFR+ Tboxes. In Horrocks, I., Sattler, U. and Wolter, F. (eds) *International Workshop on Description Logics (DL'05)*, Edinburgh, Scotland, July. National e-Science Centre.

[52]. McGuinness, D. and van Harmelen, F. (2004) OWL *Web Ontology Language Overview* W3C Recommendation. The Worldwide Web Consortium. http://www.w3.org/TR/owl-features/

[53]. Horrocks, I., Sattler, U. and Tobies, S. 'Practical Reasoning for Very Expressive Description Logics' *Logic Journal of the IGPL* **8**(3), 239–263. http://www.cs.man.ac.uk/~horrocks/Publications/download/2000/HoST00.pdf

[54]. Horrocks, I. and Sattler, U. (2005) A Tableaux Decision Procedure for SHOIQ' *Proceedings of 19th International Joint Conference on Artifi cial Intelligence (IJCAI'05)*. http://www.cs.man.ac.uk/~sattler/publications/shoiq.pdf

[55]. Dr. Berndt H. (2008) 'Towards 4G Technologies: Services with Initiative' ISBN: 978-0-470-01031-0, Wiley Chapter 7 Ontologies 141 – 164.

[56]. Semantic Web: Concepts, Technologies and Applications, NASA Monographs in Systems and Software Engineering, 2007, Part III Semantic Web Software Tools, 201-215, DOI: 10.1007/978-1-84628-710-7_10 Springer (2007).

[8a] Alonso, G., Casati, F., Kuno, H., & Machiraju, V. (2003). *Web services*. Springer.

[9a] Berners-Lee, T., Hendler J., &. Lassila, O. (2001). The semantic web. *Scientific American*, 284(5), 34–43.

[10a] Dumitru R., Uwe K., Holger L., Jos de B., Rub´en Lara, Michael S., Axel P., Cristina F., Cristoph B. and Dieter F. (2005) Web Service Modeling Ontology, Applied Ontology, 77–106. http://vsr.informatik.tu-chemnitz.de/edu/2011/webe-seminar-ws/material/12/wsmo.pdf

[11a] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[12a] Pedrinaci, C., Domingue, J. and Sheth, A. (2011) **Semantic Web Services**, in eds. John Domingue, Dieter Fensel and James Hendler, Handbook of Semantic Web Technologies, 2, Springer

[13a] Schulte, Stefan **:** *Web Service Discovery Based on Semantic Information - Query Formulation and Adaptive Matchmaking.* TU Darmstadt [Ph.D. Thesisa] , (2010) http://tuprints.ulb.tu-darmstadt.de/2293/

[14a] George Baryannis: The Frame Problem in Web Service Specifcations, University of Crete Computer Science Department [Master's Thesisa] (2009). http://ics.forth.gr/_publications/Baryannis_master.pdf

[15a] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. Mcdermott, S. Mcilraith, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, K. Sycara. OWL-S: Semantic Markup for Web Services. W3C Member Submission. (2004) http://www.w3.org/Submission/OWL-S

[16a] P. F. Patel-Schneider, P. Hayes, I. Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. W3C Recommendation. (2004)

[17a] Chris Metcalf C, and Grace LewisModel Problems in Technologies for Interoperability: OWL Web Ontology Language for Services (OWL-S), Technical Note CMU/SEI-2006-TN-018, *April 2006.* http://www.sei.cmu.edu/reports/06tn018.pdf

[18a] Klusch, M. (2008): Semantic Web Service Description. In: M. Schumacher, H. Helin, H. Schuldt (Eds.) CASCOM - Intelligent Service Coordination in the Semantic Web. Chapter 3. Birkhäuser Verlag, Springer. http://www-ags.dfki.uni-sb.de/~klusch/i2s/cascom_book_ch3-4.pdf

[19a] Martin, David, et al. *OWL-S: Semantic Markup for Web Services*. W3C Member Submission: November 22, 2004. http://www.w3.org/Submission/OWL-S/.

[20a] Martin, David, et al. *Bringing Semantics to Web Services: The OWL-S Approach*.b(2004). http://www-2.cs.cmu.edu/~softagents/papers/OWL-S-SWSWPC2004-final.pdf

[21a] World Wide Web Consortium. *OWL Web Ontology Language Guide: W3C Recommendation* (2004). *February 10, 2004*. http://www.w3.org/TR/owl-guide/

[22a] World Wide Web Consortium. *OWL Web Ontology Language Reference: W3C Recommendation February 10, 2004*. http://www.w3.org/TR/owl-ref/

[23a] World Wide Web Consortium. *OWL Web Ontology Language for Services (OWL-S)*. (2004).http://www.w3.org/Submission/2004/07/

[24a] World Wide Web Consortium. *Web Service Modeling Ontology (WSMO) Submission*. (2005). http://www.w3.org/Submission/2005/06/

[25a] World Wide Web Consortium. *WSDL-S Submission Request to W3C*. / (2005) http://www.w3.org/Submission/2005/10

[26a] World Wide Web Consortium. *Semantic Web Services Framework (SWSF)*. (2005) http://www.w3.org/Submission/2005/07/

[27a] CASCOM Project Deliverable D3.2: Conceptual Architecture Design. September 2005. www.ist-cascom.org

[28a] D. Fensel, H. Lausen, A. Polleres, J. De Bruijn, M. Stollberg, D. Roman, J. Domingue. Enabling Semantic Web Services: the Web Service Modeling Ontology. Springer (2007).

[29a] D. Roman, H. Lausen, U. Keller. Web Service Modeling Ontology (WSMO). WSMO Working Draft. (2006) http://www.wsmo.org/TR/d2/v1.3/

[30a] G. Schreiber, H. Akkermans, A. Anjewierden, R. De Hoog, N. Shadbolt, W. v. De Velde, B. Wielinga. Knowledge Engineering and Management: the CommonKADS Methodology. Mit Press. (1999)

[31a] D. Fensel, E. Motta, F. Van Harmelen, V. R. Benjamins, M. Crubezy, S. Decker, M. Gaspari, R. Groenboom, W. Grosso, M. Musen, E. Plaza, G. Schreiber, R. Studer, B. Wielinga. The Unified Problem-Solving Method Development Language UPML.Knowledge and Information Systems. **5,** 83--131 (2003)

[32a] Fensel, D., & Bussler, C. (2002). The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2).

[33a] Domingue, J., Cabral, L., Hakimpour, F., Sell, D. and Motta, E.: IRS-III: A Platform and Infrastructure for Creating WSMO-based Semantic Web Services. In Workshop on WSMO Implementations (WIW 2004) Frankfurt, Germany. CEUR Workshop Proceedings, Vol. 113 (2004)

[34a] WSMO Working Group. Deliverable D2v1.2 Web Service Modeling Ontology (WSMO) (2005).

http://www.wsmo.org/TR/d2/v1.2

[35a] L. Cabral, J. Domingue, S. Galizia, A. Gugliotta, V. Tanasescu, C. Pedrinaci, B. Norton, IRS-III: A Broker for Semantic Web Services based Applications. In proceedings of the 5 th International Semantic Web Conference (ISWC 2006) http://projects.kmi.open.ac.uk/dip/resources/ISWC2006/ISWC06CabralFinal.pdf

[36a] Cabral, L., Domingue, J., Motta, E., Payne, T. and Hakimpour, F. (2004). Approaches to Semantic Web Services: An Overview and Comparisons. In proceedings of the First Euro-pean Semantic Web Symposium, ESWS 2004, Heraklion, Crete, Greece. LNCS 3053 http://eprints.ecs.soton.ac.uk/12688/1/cabralESWS04.pdf

[37a] G. Kapitsaki, D.A. Kateros, I.E. Foukarakis, G. N. Prezerakos, D.I. Kaklamani and I.S. Venieris, Service Composition: State of the art and future challenges, http://www.ist-sms.org/Documents/C8.pdf

[38a] D. Berardi et al., “Reasoning about Actions for e-Service Composition”, ICAPS Workshop on Planning for Web Services (P4WS 2003), 2003

[39a] S. McIlraith and T.C. Son. Adapting golog for composition of semantic web services. In Proc. Of KR-02, pages 482{496, 2002. Morgan Kaufmann.

[40a] K. Erol, J. Hendler and D. S. Nau, “Semantics for Hierarchical Task Network Planning”, UMIACS Technical Report, 1994.

[41a] D. Wu, E. Sirin, J. Hendler, D. Nau, and B. Parsia, “Automatic Web Services Composition Using SHOP2”, Workshop on Planning for Web Services, 2003.

[42a] A. Patil, S. Oundhakar, A. Sheth, K. Verma, “METEOR-S Web Service Annotation Framework”, The Proceedings of the Thirteenth International World Wide Web Conference, May 2004 (WWW2004)

[43a] Semantic Annotations for Web Services Description Language, http://www.w3.org/2002/ws/sawsdl/

[44a] NetBeans.org. (2007, 11.05.2010). NetBeans IDE 6.1 SOA Pack Documentation. Available: http://netbeans.org/kb/61/soa/index.html

[45a] JOpera. (2010, 12.05.2010). JOpera for Eclipse. Available: http://www.jopera.org

[46a] F. Benedikt, et al. (2008, 02.02.2010). Hybrid OWL-S Web Service Matchmaker. Available: http://www-ags.dfki.uni-sb.de/~klusch/i2s/SMR2-2009-owlsmx3.pdf

[47a] H. Li, et al., "Automatic Composition of Web Services Based on Rules and Meta-Services," presented at the 11th International Conference on CSCW in Design, Melbourne, Australia, 2007.

[48a] S. Narayanan and S. A. McIlraith, "Simulation Verification and Automated Composition of Web Service," presented at the 11th International World wide Web conference, Hawaii, USA 2002.

[49a] I. Paik and D. Maruyama, "Automatic Web Services Composition Using Combining HTN and CSP," presented at the 7th IEEE International Conference on Computer and Information Technology (CIT 2007), Aizu-Wakamatsu City, Fukushima, Japan, 2007.

[50a] X. Li and C. Wu, "Research on OWLS Service Automatic Composition Based on Planning," Wuhan, China 2009.

[51a] W3C. 09.05.2010). OWL-S: Semantic Markup for Web Services. Available: http://www.w3.org/Submission/OWLS/

[52a] LSDIS. (2005, 06.03.2010). METEOR-S: Semantic Web Services and Processes. Available: http://lsdis.cs.uga.edu/projects/meteor

[53a] C. Feier, D. Roman, A. Polleres, J. Domingue, M. Stollberg, and D. Fensel, Towards Intelligent Web Services: The Web Service Modeling Ontology (WSMO), International Conference on Intelligent Computing (ICIC) (2005) http://axel.deri.ie/publications/feie-etal-2005.pdf

[54a] Berners-Lee, T., Hendler, J., Lassila, O.: The SemanticWeb. Scientific American 284 (2001)

[55a] Klyne, G., Carroll, J.J.: Resource Description Framework (RDF): Concepts and abstract syntax. Recommendation 10 February 2004, W3C (2004)

[56a] Brickley, D., Guha, R.V., eds.: RDF Vocabulary Description Language 1.0: RDF Schema. (2004) W3C Recommendation 10 February 2004.

[57a] Dean, M., Schreiber, G., eds.: OWL Web Ontology Language Reference. (2004) W3C Recommendation 10 February 2004.

[58a] WSML working group: WSML homepage (since 2004) http://www.wsmo.org/wsml/

[59a] McIlraith, S., Son, T.C., Zeng, H.: SemanticWeb Services. IEEE Intelligent Systems, Specia Issue on the Semantic Web 16 (2001) 46–53

[60a] Martin, D., ed.: OWL-S 1.1 Release. (2004) http://www.daml.org/services/owl-s/1.1/

[61a] Lara, R., Roman, D., Polleres, A., Fensel, D.: A Conceptual Comparison of WSMO and OWL-S. In: Proc. of the European Conf. on Web Services. (2004)

[62a] WSMO working group: WSMO homepage (since 2004) http://www.wsmo.org/

[63a] Fensel, D., Bussler, C., Ding, Y., Omelayenko, B.: The Web Service Modeling Framework WSMF. Electronic Commerce Research and Applications 1 (2002)

[64a] WSMX working group: WSMX homepage (since 2004) http://www.wsmx.org/

[65a] The OWL-S Coalition. OWL-S 1.1 draft release, 2004. http://www.daml.org/services/owl-s/1.1/

[66a] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau. HTN planning for web service composition using SHOP2. Journal of Web Semantics, 1(4):377-396, 2004.

[67a] R. Reiter. Knowledge in Action. MIT Press, 2001.

[68a] H.J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R.B. Scherl. Golog: a logic programming language for dynamic domains. Journal of Logic Programming, 31(1-3):59-83, 1997.

[69a] D. S. Nau, T. C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman. SHOP2: An HTN planning system. JAIR, 20:379-404, 2003.

[70a] M. Thielscher. Introduction to the Fluent Calculus. ETAI, 2(3{4):179{192, 1998.

[71a] F. Baader , C. Lutz , M. Miličić , U. Sattler , F. Wolter, A description logic based approach to reasoning about web services, In Proceedings of the WWW 2005 Workshop on Web Service Semantics (WSS2005) http://www.informatik.uni-bremen.de/~clu/papers/archive/wss05.pdf

[72a] Michael Stollberg , Cristina Feier , Dumitru Roman , Dieter Fensel, Semantic Web Services – Conceprts and Technology, Language Technology, Ontologies, and the Semantic Web (2006) http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.87.5391

[73a] D. Martin, editor. *OWL-S 1.1 Release*. 2004. http://www.daml.org/services/owl-s/1.1/

[74a] D. Martin, editor. *OWL-S 1.2 Release*. 2008. http://www.ai.sri.com/daml/services/owl-s/1.2/

[75a] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan. Automated Discovery, Interaction and Composition of SemanticWeb services. *Journal of Web Semantics*, 1(1):27{46, September 2003.

[76a] D. Roman, H. Lausen, and U. Keller (eds.). Web Service Modeling Ontology (WSMO). Deliverable D2, final version 1.2, 2005. Available from http://www.wsmo.org/TR/d2/

[77a] A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler. WSMX - A Semantic Service-Oriented Architecture. In *Proceedings of the International Conference on Web Service (ICWS 2005)*, 2005.

[78a] J. de Bruijn (ed.). The Web Service Modeling Language WSML. WSML Deliverable D16.1 _nal version 0.2, 2005. available from http://www.wsmo.org/TR/d16/d16.1/v0.2/

[79a] OMG: The Object Management Group. Meta-object facility, version 1.4, 2002.

[80a] S. Staab and R. Studer, editors. *Handbook on Ontologies in Information Systems*. International Handbooks on Information Systems. Springer, 2004.

[81a] D. Fensel. *Ontologies: A Silver Bullet for Knowledge Management and ECommerce*. Springer, Berlin, Heidelberg, 2 edition, 2003.

[82a] G. Wiederhold. Mediators in the architecture of the future information systems. *Computer*, 25(3):38{49, 1994.

[83a] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Proceedings of the First International Semantic Web Conference, Springer*, 2002

[84a] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proceedings of the 12th International Conference on the World Wide Web, Budapest, Hungary.*, 2003.

[85a] U. Keller, R. Lara, and A. Polleres (eds.). WSMO Web Service Discovery. Deliverable D5.1, 2004. available at: http://www.wsmo.org/TR/d5/d5.1/

[86a] S. Grimm, B. Motik, and C. Preist. Variance in e-business service discovery. In *Proc. of the ISWC 2004 workshop on Semantic Web Services: Preparing to Meet the World of Business Applications, Hiroshima, Japan, Nov. 2004*, 2004.

[87a] M. Kifer, R. Lara, A. Polleres, C. Zhao, U. Keller, H. Lausen, and D. Fensel. A Logical Framework for Web Service Discovery. In *Proc. of the ISWC 2004 workshop on Semantic Web Services: Preparing to Meet the World of Business Applications, Hiroshima, Japan, Nov. 2004*, 2004.

[88a] M. Stollberg, U. Keller, and D. Fensel. Partner and Service Discovery for Collaboration Establishment on the SemanticWeb. In *Proceedings of the Third International Conference on Web Services, Orlando, Florida*, 2005.

[89a] J. de Bruijn, A. Polleres, R. Lara, and D. Fensel. OWL DL vs. OWL Flight: Conceptual Modeling and Reasoning for the Semantic Web. In *Proceedings of the 14th International World Wide Web Conference (WWW2005), Chiba, Japan, 2005.*, 2004. http://www2005.org/cdrom/docs/p623.pdf

[90a] J. Cardoso and A. Sheth, "Introduction to Semantic Web Services and Web Process Composition," in Semantic Web Process: powering next generation of processes with Semantics and Web Services, Lecture Notes in Computer Science, Springer, 2005

[91a] SWSI, Semantic Web Services Initiative (SWSI). http://www.swsi.org/, 2004.

[92a] OWL-S, OWL-based Web Service Ontology. http://www.daml.org/services/owl-s/, 2004.

[93a] WSMO, Web Services Modeling Ontology (WSMO). http://www.wsmo.org/, 2004.

[94a] WSML, Web Service Modeling Language (WSML). http://www.wsmo.org/wsml/index.html, 2004.

[95a] WSMX, Web Services Execution Environment (WSMX). http://www.wsmx.org/, 2004.

[96a] LSDIS, METEOR-S: Semantic Web Services and Processes. http://lsdis.cs.uga.edu/Projects/METEOR-S/index.php, 2004.

[97a] SWWS, Semantic Web Enabled Web Service. http://swws.semanticweb.org/, 2004, Digital Enterprise Research Institute (DERI).

[98a] DERI, Digital Enterprise Research Institute (DERI). http://www.deri.ie/, 2004.

[99a] Michael C. Jaeger , Gregor Rojec-goldmann , Christoph Liebetruth , Gero Mühl , Kurt Geihs, Ranked Matching for Service Descriptions using OWL-S, In KiVS 2005, Informatik Aktuell (2005

[100a] Marja-Riitta Koivunen and Eric Miller. W3c semantic web activity. In Semantic Web Kick- Off in Finland, pages 27–44, November 2001. http://www.w3.org/2001/12/semweb-fin/w3csw

[101a] Anupriya Ankolenkar et al. Daml-s: A semantic markup language for web services. In Proceedings of 1st Semantic Web Working Symposium (SWWS' 01), pages 441–430, Stanford, USA, August 2001. Stanford University.

[102a] Frank Manola et al. RDF Primer. Technical report, W3C, http://www.w3.org/TR/rdf-primer/ 2004.

[103a] Deborah L. McGuinness and Frank van Harmelen. Owl web ontology language overview. Technical report, W3C, http://www.w3.org/TR/owl-features/ 2004.

[104a] Sheila A. McIlraith and David L. Martin. Bringing semantics to web services. IEEE Intelligent Systems, 18:90–93, January/February 2003.

[105a] The OWL Services Coalition. OWL-S: Semantic Markup for Web Services. Technical report, http://www.daml.org/services/ 2004.

[106a] Jos De Bruijn, Dieter Fensel, Uwe Keller, Ruben Lara, Using the Web Service Modelling Ontology to enable Semantic eBusiness, Volume: 48, Issue: 12, Publisher: ACM, Pages: 43 Communications of the ACM, (2005)

[107a] Robert Allen and David Garlan. A Formal Basis for Architectural Connection. ACM Transactions on Software Engineering and Methodology, 6(3):213{249, July 1997.

[108a] Dieter Fensel and Christoph Bussler. The Web Service Modeling Framework WSMF. Electronic Commerce Research and Applications, 1(2):113137, 2002.

[109a] Ruben Lara, Axel Polleres, Holger Lausen, Dumitru Roman, Jos de Bruijn, and Dieter Fensel. A Conceptual Comparison between WSMO and OWL-S. Final Draft D4.1v0.1, WSMO, 2005. http://www.wsmo.org/TR/d4/d4.1/v0.1/

[110a] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, Katia Sycara, OWL Web Ontology Language for Services (OWL-S). W3C Member Submission, 22 November 2004. http://www.w3.org/Submission/OWL-S/.

[111a] World Wide Web Consortium. *Semantic Web Services Interest Group*. http://www.w3.org/2002/ws/swsig/ (2003)

[112a] Intelligent Software Agents Lab in the Robotics Institute at Carnegie Mellon I University. *The Intelligent Software Agents Lab*. http://www-2.cs.cmu.edu/~softagents/index.html (2001)

Annex 1 – pSHIELD Glossary

Concept	Description
Application Processor (AP)	Main processing unit
Atomic pSHIELD SPD Component	Is a generic atomic SPD Functionality (innovative or legacy) provided by a pSHIELD device at node, network or middleware level.
Audit	Involves recognizing, recording, storing, and analyzing information related to SPD relevant activities. The resulting audit records can be examined to determine which SPD relevant activities took place.
Authorized User	A user who possesses the rights and/or privileges necessary to perform an operation
Automatic Web Service Composition and Interoperation	This task involves the automatic selection, composition, and interoperation of Web services to perform some complex task, given a high-level description of an objective.
Automatic Web Service Discovery	Is an automated process for location of Web services that can provide a particular class of service capabilities, while adhering to some client-specified constraints.
Automatic Web Service Invocation	Is the automatic invocation of an Web service by a computer program or agent, given only a declarative description of that service, as opposed to when the agent has been pre-programmed to be able to call that particular service.
Availability	Readiness for correct service. The correct service is defined as delivered system behaviour that is within the error tolerance boundary.
Awareness	Capability of the Cognitive Radio to understand, learn, and predict what is happening in the radio spectrum, e.g., to identify the transmitted waveform, to localize the radio sources, etc.
Bridge	Is a network device that is at the link layer of the ISO / OSI model and translates from one physical media to another within the same local network.
Class and Family	The CC has organized the components into hierarchical structures: Classes consisting of Families consisting of components. This organization into a hierarchy of class - family - component - element is provided to assist consumers, developers and evaluators in locating specific components
Cognitive Radio	Is an intelligent wireless communication system that is aware of its surrounding environment (i.e., outside world), and uses the methodology of understanding-by-building to learn from the environment and to adapt its internal states to statistical variations in the incoming Radio-Frequency (RF) stimuli by making corresponding changes in certain operating parameters (e.g., transmit-power, carrier-frequency, and modulation strategy) in real-time, with two primary objectives in mind: (i) highly Reliable and Dependable communications whenever and wherever needed and (ii) efficient utilization of the radio spectrum.
Common Criteria	The Common Criteria for Information Technology Security Evaluation (abbreviated as Common Criteria or CC) is an international standard (ISO/IEC 15408) for computer security certification. It is currently in version 3.1. Common Criteria is a framework in which computer system users can specify their security functional and assurance requirements, vendors can then implement and/or make claims about the security attributes of their products, and testing laboratories can evaluate the products to determine if they actually meet the claims. In other words, Common Criteria provides assurance that the process of specification, implementation and evaluation of a computer security product has been conducted in a rigorous and standard manner.
Common Criteria Approach	Is approach based in three fundamental part: • Assets to protect and in particular SPD attribute of these assets definition

Concept	Description
	• Threats identifications (Fault Errors Failures); in our approach faults are grouped in HMF (FUA, NFUA) and NHMF. • SPD functionalities (whose purpose is to mitigate threats) are implemented to meet pSHIELD SPD objectives.
Composability	Is the possibility to compose different (possibly heterogeneous) SPD functionalities (also referred to as SPD components) aiming at achieving in the considered system of Embedded System Devices a target SPD level which satisfies the requirements of the considered scenario.
Confidentiality	Property that data or information are not made available to unauthorized persons or processes.
Contiki Operating System	Contiki is also an open source, highly portable, multi-tasking operating system for memory-efficient networked ESDs and WSNs
Control Algorithms	Retrieves the aggregated information on the current SPD status of the subsystem, as well as of the other interconnected subsystems, by the pS-CA interface connected to the Semantic Knowledge Representation; such retrieved information is used as input for the Control Algorithms. The outputs of the Control Algorithms consist in decisions to be enforced in the various ESDs included in the pSHIELD subsystem controlled by the Security Agent in question; these decisions are sent back via the pS-MS interface, as well as communicated to the other Security Agents on the Overlay, through the pS-OS interface.
Control system	A system that uses a regulator to control its behaviour
Core SPD Services	The core SPD services are a set of mandatory basic SPD functionalities provided by a pSHIELD Middleware Adapter in terms of pSHIELD enabling middleware services. The core SPD services aim to provide a SPD middleware environment to actuate the decisions taken by the pSHIELD Overlay and to monitor the Node, Network and Middleware SPD functionalities of the Embedded System Devices under the pSHIELD Middleware Adapter control.
Correct service	Delivered system behaviour is within the error tolerance boundary.
Cryptographic Algorithms	Algorithms to hiding the information, to provide security and information protection against different forms of attacks
Dependability	Is a composite concept that encompasses the following attributes: Availability, Reliability, Safety Integrity, Maintainability
Desired objectives	Desired objectives normally specified by the user.
Desired service	Delivered system behavior is at or close to the desired trajectory.
Discovery	Provide to the pSHIELD Middleware Adapter the information, raw data, description of available hardware resources and services in order to allow the system composability
Discovery Engine	it is in charge to handle the queries to search for available pSHIELD components sent by the Composition service.
Discovery Protocol	it is in charge to securely discover all the available SPD components description stored in the Service Registry, using a specific protocol
Disturbance	Anything that tends to push system behavior off the track is considered a disturbance. A disturbance can occur within a system or from the external environment.
Error	Deviation of system behavior/output from the desired trajectory.
Error tolerance boundary	A range within which the error is considered acceptable by a system or user. This boundary is normally specified by the user.
Evaluator	An independent person involved in the judgment about the measure of the SPD functions.
Failure	An event that occurs when system output deviates from the desired service and is beyond the error tolerance boundary.
Fault	Normally the hypothesized cause of an error is called fault. It can be internal or external to a system. An error is defined as the part of the total state of a system that may lead to subsequent service failure. Observing that many errors do not reach a system's external state and

Concept	Description
	cause a failure, Avizienis et al. have defined active faults that lead to error and dormant faults that are not manifested externally.
Fault injection	This block has the responsibility to inject a fault into Demodulator
Faults with Unauthorized Access	The class of Faults with unauthorized access (FUA) attempts to cover traditional security issues caused by malicious attempt faults. Malicious attempt fault has the objective of damaging a system. A fault is produced when this attempt is combined with other system faults.
Feedback	Use of the information observed from a system's behavior to readjust/regulate the corrective action/control so that the system can achieve the desired objectives.
Feedback control system	A control system that deploys a feedback mechanism. This is also called a closed-loop control system.
Filter Engine	It is in charge to semantically match the query with the descriptions of the discovered SPD components
Flash Memory	It stores the bit-stream and system status information
Forecasting (Fault)	Mechanism that predicts faults so that they can be removed or their effects can be circumvented
Functional Component	Describes a functional entity that, in general, does not have necessarily a physical counterpart (e.g. a software functionality, a middleware service, an abstract object, etc.).
Gateway	Is a network device that operates at the network layer and above the ISO / OSI model. Its main purpose is to transmit network packets outside a local area network (LAN).Functional Component
Grounding	Provides details on how to interoperate with a service, via messages.
Health Status Monitoring (HSM)	Monitoring for checking the status of each individual component.
Hub	Is a concentrator, a network device that acts as a routing node of a data communications network
Human-Made Faults	Human-made faults result from human actions. They include absence of actions when actions should be performed (i.e., omission faults). Performing wrong actions leads to commission faults. HMF are categorized into two basic classes: faults with unauthorized access (FUA), and other faults (NFUA).
Hybrid Automata	Is composed by automaton formulation hybrid formulation that Permit to choose the most suitable configuration rules for components that must be composed on the basis of the Overlay control algorithms.
HYDRA Middleware	Middleware for Heterogeneous Physical Devices
I/O Interface	(I/O) to connect to any peripheral and to the rest of the pSHIELD embedded functionalities.
Innovative SPD Functionalities	Reside in the pSHIELD Middleware, Network and Node Adapters. They are constituted by a variety of pSHIELD-specific components. Each pSHIELD-specific component. represents an innovative SPD functionality ad hoc developed for the pSHIELD project which is included in the pSHIELD Node, Network or Middleware Adapter.
Integrity	Absence of malicious external disturbance that makes a system output off its desired service
Legacy Device Component	i.e. the SPD functionalities already present in the legacy devices which can be accessed through the pSHIELD Adapters; they can be classified in Node, Network and Middleware Component according to whether they are included in a legacy Node/Network/Middleware which can be accessed through the corresponding pSHIELD Adapter.
Legacy Embedded System Device (L-ESD)	It represents an individual, atomic physical Embedded System device characterized by legacy Node, Network and Middleware functionalities.
Legacy Functionalities of L-ESD	Is a functionality partitioned into three subsets: - Node layer functionalities: hardware functionalities such as processors, memory, battery, I/O interfaces, peripherals, etc. - Network layer functionalities: communication functionalities such as connectivity, protocols stack, etc.

Concept	Description
	- Middleware layer functionalities: firmware and software functionalities such as services, functionalities, interfaces, protocols, etc.
Legacy Middleware Services	Includes all the legacy middleware services (i.e. messaging, remote procedure calls, objects/content requests, etc.) provided by the Legacy Embedded System Device which are not pSHIELD-compliant.
Legacy Network Services	Includes all the legacy network services (protocol stacks, routing, scheduling, Quality of Service, admission control, traffic shaping, etc.) provided by the Legacy Embedded System Device which are not pSHIELD-compliant.
Legacy Node Capabilities	Component includes all the legacy node capabilities (i.e. battery, CPU, memory, I/O ports, IRQ, etc.) provided by the Legacy Embedded System Device which are not pSHIELD compliant.
Life-Cycle support elements	It is the set of elements that support the aspect of establishing discipline and control in the system refinement processes during its development and maintenance. In the system life-cycle it is distinguished whether it is under the responsibility of the developer or the user rather than whether it is located in the development or user environment. The point of transition is the moment where the system is handed over to the user.
Maintainability	Ability to undergo modifications and repairs.
Mean	All the mechanisms that break the chains of errors and thereby increase the dependability of a system
Memory	Memory RAM, SRAM, DRAM,
Metadata	Information that describe set of data
Micro/Personal nodes	Nodes that are richer than Nano Nodes in terms of hardware and software resources, network access capabilities, mobility, interfaces, sensing capabilities, etc.
Middleware Layer	Includes the software functionalities that enable the discovery, composition and execution of the basic services necessary to guarantee SPD as well as to perform the tasks assigned to the system (for example, in the railway scenario, the monitoring functionality)
Nano Nodes	Are typically small ESD with limited hardware and software resources, such as wireless sensors.
Net Device	Components used to connect computers or other electronic devices
Network CAN	Control Area Network
Network LAN	Local Area Network
Network Layer	Includes the communication technologies (specific for the rail transportation scenarios) that allow the data exchange among pSHIELD components, as well as the external world. These communication technologies, as well as the networks to which pSHIELD is interconnected can be (and usually are) heterogeneous.
Network MAN	Metropolitan Area Network
Network VPN	Virtual Private Network
Network WAN	Wide Area Network
Node Layer	Includes the hardware components that constitute the physical part of the system.
Node Metrics / Health Status	It receives periodic health messages and metrics from the other blocks. This block contains the information about the full node configuration, metrics and health status. If e.g. the “Assertions” block detects some erroneous output, “Node Metrics / Health Status” block receives this information and must act accordingly.
NonHuman-Made Faults	NHMF refers to faults caused by natural phenomena without human participation. These are physical faults caused by a system’s internal natural processes (e.g., physical deterioration of cables or circuitry), or by external natural processes. They can also be caused by natural phenomena
Non-Volatile Memory (NVM)	Memory ROM, EEPROM, FLASH, Hard Disk
Not Faults with	Human-made faults that do not belong to FUA. Most of such faults are

Concept	Description
Unauthorized Access	introduced by error, such as configuration problems, incompetence issues, accidents, and so on.
Open-loop control system	A control system without a feedback mechanism.
Overlay	Consists of a set of SPD Security Agents, each one controlling a given pSHIELD subsystem.
Overlay Layer	The “embedded intelligence” that drives the composition of the pSHIELD components in order to meet the desired level of SPD. This is a software layer as well.
Physical Component	Describes an entity that can be mapped into a physical object (e.g. a hardware component).
Plant	A system that is represented as a function P(.) that takes its input as a functional argument
Power Management (PM)	Module for managing power sources, monitoring power consumption, etc.
Power nodes	Is an ES node that offers high performance computing in one self-contained board offering data storage, networking, memory and (multi-)processing.
Prevention (Fault)	Mechanism that deals with preventing faults incorporated into a system
Privacy	The right of an entity (normally a person), acting in its own behalf, to determine the degree to which it will interact with its environment, including the degree to which the entity is willing to share information about itself with others.
pSHIELD Adapter	Is a technology dependent component in charge of interfacing with the legacy Node, Network and Middleware functionalities (through the MS, NS and NC interfaces). The legacy functionalities can be enhanced by the pSHIELD Adapter in order to make them pSHIELD-compliant, i.e. they become SDP legacy device components. In addition, the pSHIELD Adapter includes Innovative SPD functionalities which are SPD pSHIELD-specific components, which can be composed with other SPD components. The pSHIELD Adapter exposes the technology independent pSHIELD Middleware layer functionalities that are used by the Security Agent component.
pSHIELD Communication	This block interfaces SPD Node to pShield Network. It is composed by: Ethernet interface: it allows a communication data interface based on a TCP/IP protocol Message encoding/decoding: it receives the commands from pShield network, decodes them, and acts accordingly. It also sends messages to the network.
pSHIELD Demonstrator	Demonstrator for the project that Have the task To monitor on-carriage environment; To integrate the sensors at the wagon with the M2M platform; To allow secure interoperability of sensor information (between railway infrastructure and third party service provider).
pSHIELD Embedded System Device (pS-ESD)	It is a L-ESD equipped at least with the minimal set of pSHIELD functionalities at Middleware Layer. The pS-ESD exposes the same functionalities as the L-ESD plus an additional interface: the pSHIELD Middleware layer services.
pSHIELD Middleware Adapter	Is a component partitioned in the Core SPD services and in the Innovative SPD functionalities. These two components are linked through the pS-MS interface. The pSHIELD Middleware Adapter should also carry into operation the decisions taken by the Overlay and communicated via the pS-MS interface by actually composing the discovered SPD functionalities. The pSHIELD Middleware Adapter includes a set of Innovative SPD functionalities interoperating with the legacy ESD middleware services (through the MS interface) in order to make them discoverable and composable SPD functionalities.

Concept	Description
pSHIELD Multi-Layered Approach	The pSHIELD multi-layered approach considers the partition of a given Embedded System into three technology-dependent horizontal layers: the node layer (meaning the hardware functionalities), the network layer (meaning the communication functionalities) and the middleware layer (meaning thesoftware functionalities).
pSHIELD Network Adapter	Includes a set of Innovative SPD functionalities interoperating with the legacy ESD network services (through the NS interface) and the pSHIELD Node Adapter (through the pS-NC interface) in order to enhance them with the pSHIELD Network layer SPD enabling technologies (such as Smart Transmission). This adapter is also in charge to provide (through the pS-NS interface) all the needed information to the pSHIELD Middleware adapter to enable the SPD composability of the Network layer legacy and Network pSHIELD-specific functionalities. Moreover, the pSHIELD Network Adapter translates the technology independent commands, configurations and decisions coming from the pS-NS interface into technology dependent ones and enforce them also to the legacy Network functionalities through the NS interface.
pSHIELD Node	Is an Embedded System Device (ESD) equipped with several legacy Node Capabilities and with a pSHIELD Node Adapter. A pSHIELD Node is deployed as a hardware/software platform, encompassing intrinsic, Innovative SPD functionalities, providing proper services to the other pSHIELD Network and Middleware Adapters to enable the pSHIELD Composability and consequently the desiredsystem SPD
pSHIELD Node Adapter	Includes a set of Innovative SPD functionalities interoperating with the legacy ESD node capabilities (using the NC interface) in order to enhance them with the pSHIELD Node layer SPD enabling technologies (such as FPGA Firmware and Lightweight Asymmetric Cryptography). This adapter is in charge to provide (through the pS-NC interface) all the needed information to the pSHIELD Middleware adapter to enable the SPD composability of the Node layer legacy and Node pSHIELD-specific functionalities. Moreover, the pSHIELD Node Adapter translates the technology independent commands, configurations and decisions coming from the pS-NC interface into technology dependent ones and enforce them also to the legacy Node functionalities through the NC interface.
pSHIELD Proxy (pS-P)	Is a technology dependent component of a pS-SPD-ESD that, interacting with the available legacy Node, Network and Middleware capabilities and functionalities (through the NC, NS and MS interfaces, respectively), provides all the needed pSHIELD enhanced SPD functionalities.
pSHIELD SPD Embedded System Device (pS-SPD-ESD):	It is a pS-ESD equipped at least with the minimal set of pSHIELD Overlay functionalities. The pS-SPD-ESD exposes the same functionalities as the pS-ESD plus an additional interface: the pSHIELD Overlay layer SPD services provided by a so-called Service Agent operating in that ESD.

Concept	Description
pSHIELD Subsystem (pS-S)	Is an architecture of a set of Embedded System Devices including several L-ESD, connected to several pS-ESD and one and only one pS-SPD-ESD. Connections between two generic ESDs (L-ESD, pS-ESD or pS-SPD-ESD) can be performed, by means of legacy functionalities at Node, Network and/or Middleware layer, through the so-called NC, NS and MSfunctionalities, respectively.
pSHIELD-Specific Components	It is i.e. the innovative SPD functionalities ad hoc developed for the pSHIELD project which are included in the pSHIELD Adapters. They can be classified in Node Network and Middleware pSHIELD-specific components according to whether they are included in the pSHIELD Node, Network or Middleware Adapter. They can be directly accessed by pSHIELD Middleware Core SPD Services through the pSNC, pS-NS and pS-MS interfaces.
Query Preprocessor	It is in charge to enrich the query sent by the Composition service with semantic information related to the peculiar context.
Reconfigurability	Provide self-configuration of some internal parameters according to the observed radio spectrum.
Reconfiguration / Recovery	This block runs at the PPC static core. It must receive periodically health status information, otherwise it restarts the system
Reconfiguration/Recovery Controller	This is a hard processor or microcontroller, responsible for either reconfiguring the node or recovering in case of an error.
Recovery Watchdog Timer (RWDT)	Timer for restarting recovery if no activity is detected from the SHSM.
Regulator	A control function whose role is to regulate the input of the plant, under the influence of the environment such as instructions of the user, observed error, input disturbance, etc. to achieve the desired objective.
Reliability	Continuity of correct service even under a disturbance.
Removal (Fault)	mechanism that permits to the system to record failures and remove them via a maintenance cycle
Repeater	A digital device that amplifies, reshapes, retimes, or performs a combination of any of these functions on a digital input signal for retransmission
Router	Is a device that forwards data packets between telecommunications networks, creating an overlay internetwork. A router is connected to two or more data lines from different networks.
Rules for Discovery, Configuration and Composition of the SPD Components.	Design and implementation of the Control Algorithms which, on the basis of the sensed metadata (i.e) on the basis of the ontological description (possibly semantically enriched) of the SPD Components provide Rules for discovery, configuration and composition of the SPD components.
Safety	Absence of catastrophic consequences on the users and the environment.
Service failure	An event that occurs when system output deviates from the desired service and is beyond the error tolerance boundary.
SDP Network	Is a Network implementable and interoperable with standard networks to comply the main business cases of the application scenarios.
Seamless Approach	Common approach which leaves out of consideration the specificity of the underlying technologies providing enriched SPD functionalities to heterogeneous Embedded Systems
Secure Service Discovery	Allows any pSHIELD Middleware Adapter to discover in a secure manner the available SPD functionalities and services over heterogeneous environment, networks and technologies that are achievable by the pSHIELD Embedded System Device (pS-ESD) where it is running.

Concept	Description
Secure/Privacy (SP)	Module to perform security and privacy actions, such as encryption, decryption, key generation, etc.
Security	Is a composite of the attributes of confidentiality, integrity (in the security context, “improper” means “unauthorized"), and availability (for authorized actions only),
Security Agent	Is a technology-independent component in charge of aggregating the information coming from the pSHIELD Middleware Services provided by the internal pSHIELD Adapter or by other pSHIELD Proxies located in the same subsystem. The Security Agent is also in charge of gathering the information coming from other Security Agents connected on the same Overlay (through the pS-OS interface). The Security Agent includes proper control algorithms working on the basis of the available information; the decisions taken by these Control Algorithms are enforced through the pS-MS and the pS--OS interfaces.
Semantic Database	It holds any semantic information related to the pSHIELD components (interface, contract, SPD status, context, etc.).
Semantic Engine (Reasoner)	Enable interoperability within Middleware Layer and rule based discovery and composition within Overlay Agents.
Semantic Knowledge Repository	Is (i.e. a database) that storing the dynamic, semantic, enriched, ontological aggregated representation of the SPD functionalities of the pSHIELD subsystem controlled by the SPD Security Agent;
Semantic Knowledge Representation	Is in charge of bi-directionally exchanging technology independent (and semantic enriched) information from the pS-MS and the pS-OS interfaces. It is also in charge to provide such information via the pS-SKR interface to the Control Algorithms component.
Semantic Model	It is a conceptual model in which semantic information is included.
Sensor/Actuator	Are represented by the Core SPD Services lying at the pSHIELD Middleware layer.
Sensors/Actuators	Represent the Core SPD Services lying at the pSHIELD Middleware layer.
Service Composition	Is in charge to select atomic SPD services that, once composed, provide a complex and integrated SPD functionality that is essential to guarantee the required SPD level. The service composition is a pSHIELD Middleware Adapter functionality that cooperates with the pSHIELD Overlay in order to apply the configuration strategy decided by the Control Algorithms residing in the pSHIELD Security Agent.
Service Grounding	Specifies the details of how an agent can access a service-details having mainly to do with protocol and message formats, serialization, transport, and addressing
Service Orchestration	Deploy, execute and monitoring SPD services.
Service Profile	Tells "what the service does", in a way that is suitable for a service-seeking agent (or matchmaking agent acting on behalf of a service-seeking agent) to determine whether the service meets its needs.
Services Model	Tells a client how to use the service, by detailing the semantic content of requests, the conditions under which particular outcomes will occur, and, where necessary, the step by step processes leading to those outcomes.
Services Registry	It acts as a database to store the service entries
Smart SPD Driven Transmission	New advances signal processing techniques.
SOA	Service-oriented architecture

Concept	Description
Software Agent	Permit a computer-interpretable description of the service.
Software Defined Radio (SDR)	Software programmable Components
Software Wireless Sensor Networks (WSN)	Software part that can be layered into three levels: sensor software, node software and sensor network software.
SPD	Security Privacy Dependability
SPD Component	Is defined as a functionality which (i) offers a given SPD service through an interface which can be semantically described according to the provided SPD Metrics, (ii) can be accessed through the pSHIELD Middleware Core SPD Services for being configured (if necessary) and activated (or deactivated).
SPD Metrics	Is the possibility to identify and quantify the SPD properties of each component, as well as the SPD properties of the overall system. SPD Metrics allow (i) users to define in an univocal way the requirements for the specific application, (ii) to describe the SPD level provided by the components, and (iii) to compute the SPD level achieved by the system through the Composability mechanism.
SPD Node	It is composed by the following sub- blocks: FM Signal Acquisition: this blocks principally handles the receiving of data samples from “FM Signal Generator” and pre-processes the data to feed to the “Demodulation Processing” block. This block provides also periodic status & metrics information to the “Node Metrics/Health Status” block. Demodulation Processing: it is responsible for the demodulation processing of the data coming from the “FM Signal Acquisition“
SPD Security Agent	Consists of two key elements: (i) the Semantic Knowledge Repository (i.e. a database) storing the dynamic, semantic, enriched, ontological aggregated representation of the SPD functionalities of the pSHIELD subsystem (ii) the Control Algorithms generating, on the grounds of the above representation, key SPD-relevant decisions (consisting, as far as the Composability feature is concerned, in a set of discovery, configuration and composition rules).
SPD Status	It represents the current SPD level associated to the function.
Special Purpose Processor	(SPP) module for any pre- or post-processing, such as compression/decompression, conversion, etc.
Persistant Storage	Used for storing the status of the system, a bit stream to program an FPGA, and/or the software for system start-up, operating system and application. it receives from each block check-pointing data. It is able to perform a stable write with this data (write on a circular buffer on flash memory, and then validate the just written data). On system restart, this module is able to recover the last valid data.
Switch	Is a computer networking device that connects network segments.
System	A composite constructed from functional components. The interaction of these components may exhibit new features/functions that none of the composite components possess individually.
System output	System behavior perceived by the user.
System Health Status Monitoring (SHSM)	Monitoring for checking the status of the whole system.
The Security/Privacy controller	Consists of one or more modules able to perform different security-related functionalities, such as Data Encryption, Data Decryption, Generation of Cryptographic Keys, etc

Concept	Description
Threat	Include faults, errors and failures, as well as their causes, consequences and characteristics.
TinyOS	This operating system (OS) is a free and open source operating system and platform that is designed for WSNs.
Tolerance (Fault)	System architecture that deals with putting mechanisms in place that will allow a system to still deliver the required service in the presence of faults, although that service may be at a degraded level
User session	A period of interaction between users and SPD functional components.
WSDL	Web Services Description Language

Table 2 pSHIELD Glossary