UNIVERSITY OF OSLO
Department of informatics

Semantic Web Services and the Traveller scenario

Master thesis
60 credits

Marianne Rustad

01. August 2006
**Background information**

I (Marianne Rustad) started on my bachelor degree at Bodo Graduate School of Business in the program “Bachelor in Information Technology”. The first year was integrated with the course in business administration. The following two years focused in depth on system and application development, also including some elective courses giving me in depth knowledge of application development and the technology of the telephone network.

I started my master’s degree at the Brunel University of West London (through The Norwegian School of Information Technology) in the Distributed Information System and Computing program. This program focused on information technology in distributed systems including a course in semantic integration frameworks which also included the development of OWL ontology. I switched to The University of Oslo for the second year where I have continued my master degree at the University Graduate Centre (UNIK) under the supervision of Prof. stip. Dr. Josef Noll.

My starting point and background for the master theses in Semantic Web Services was the semantic integration frameworks course from Brunel University of West London. My knowledge was the Web Ontology Language and through this theses I have learned WSMO, WSDL-S, OWL-S and how they are defined, how they differs and completes each other. Other technology I have focused on as a result of OWL-S, WSMO and WSDL-S is SOA, UDDI, WSDL and WSML. The framework initiatives are also evaluated against criteria I have found important for such technology. Then real world evaluations of the use of these initiatives are shown in the context of a traveller scenario.

This year of working and writing this theses has given me an understanding of the initiatives and the background for them. This has resulted in my work of defining basic requirements for semantic frameworks for Semantic Web Services. Further has my work been a factor used in an analysis of cost effectiveness in comparison of conventional Web Services. The work cumulates in the real world implementation of the traveller scenario through examples from the different initiatives.
Acknowledgements

I would like to thank Josef Noll for his valuable advice and shearing of knowledge. It has been a pleasure to work at this theses and other small projects and papers with him supervising the work. His ability to include me as a student in projects as travelling to Telenor's lab in Trondheim, a paper submitted to Communication magazine and the involvement in the Adaptive Service Grid project has provided me with knowledge beyond theoretical knowledge.

I would also like to thank Erik Lillevold for his involvement in my master theses the first semester. The meetings with him and Josef Noll resulted in discussions that contributed to my analysis of the subjects at hand. Also his involvement in the technical part of setting up a server to Telenor in Trondheim had great value for this thesis. Unfortunately he had to back out of the co-supervisor role in the spring semester due to the workload of being retired and a doctorates student.

Others I need to thank for their help includes Gyorgy Kalman with his help in the VPN setup to Trondheim and discussions about UDDI deployment.

Further I would like to thank my family and friends for supporting my decision of doing the master degree and all the work this has included. Especially I want to thank my live-in boyfriend Jon Brannsten for the support and advices along the way as well as all the dinners he had to make.
# Table of Contents

Background information ...................................................................................................................................... 2
Acknowledgements........................................................................................................................................ 4
List of Abbreviations ................................................................................................................................. 10
Abstract......................................................................................................................................................... 12
Introduction.................................................................................................................................................. 14

**PART 1 Background for Semantic Web Services** ..................................................................................... 16

1. Web technology ............................................................................................................................................... 16

2. Web Services ................................................................................................................................................ 16
   2.1. Advantages of Web service ...................................................................................................................... 17
   2.2. Deficiencies of Web service .................................................................................................................... 18

3. Service-Oriented Architecture (SOA) ........................................................................................................... 18
   3.1. Advantages of SOA ................................................................................................................................. 20
   3.2. Deficiencies of SOA ............................................................................................................................... 20

4. Semantic Web .............................................................................................................................................. 20
   4.1. Advantages of Semantic Web .................................................................................................................. 22
   4.2. Deficiencies of Semantic Web ................................................................................................................ 22

5. Semantic Web Services ............................................................................................................................. 22
   5.1. Advantages of Semantic Web Services ................................................................................................... 23
   5.2. Deficiencies of Semantic Web Services ................................................................................................ 23

**Part 2 – The Semantic Web Services initiatives** .......................................................................................... 25

6. Requirements for Semantic Web Services .................................................................................................. 25
   6.1. Standardisation of Semantic Web Services .............................................................................................. 25
   6.2. Issues in Web Ontology Language- Service (OWL-S) ........................................................................... 26
   6.3. Issues in Web Service Modelling Ontology (WSMO) ............................................................................ 27
   6.4. Issues in Web Service Description Language –Service (WSDL-S) ........................................................... 28
   6.5. High-level requirements ......................................................................................................................... 29

7. Web Service Modelling Ontology (WSMO) ............................................................................................... 30
   7.1. Background information on WSMO ......................................................................................................... 30
   7.2. Goals of WSMO development ............................................................................................................... 30
   7.3. Tools for WSMO ..................................................................................................................................... 30
   7.4. Conceptual elements of WSMO ............................................................................................................. 30
   7.5. Implementation details of WSMO through WSML
      7.5.1. Defining the web service, goal and mediators ................................................................................... 33
      7.5.2. Defining the ontology ....................................................................................................................... 35
      7.5.3. Grounding in WSDL ....................................................................................................................... 36
8. Web Ontology Language – Service ontology (OWL-S) ........................................................................ 37
  8.1. Background information of OWL-S ......................................................................................... 37
  8.2. Goals of OWL-S development ............................................................................................... 37
  8.3. Tools for OWL-S ...................................................................................................................... 37
  8.4. Conceptual elements of OWL-S ............................................................................................. 38
  8.5. Implementation details of OWL-S .......................................................................................... 39
    8.5.1. The service class .................................................................................................................. 39
    8.5.2. The Service Profile ............................................................................................................ 39
    8.5.3. The Service model .............................................................................................................. 40
    8.5.4. The Grounding model ....................................................................................................... 41
    8.5.5. The OWL ontology ............................................................................................................ 42

9. Web Service Description Language – Service (WSDL-S) ............................................................... 46
  9.1. Background information of WSDL-S ..................................................................................... 47
  9.2. Goals of WSDL-S development ............................................................................................ 47
  9.3. Tools for WSDL-S .................................................................................................................. 47
  9.4. Conceptual elements of WSDL-S .......................................................................................... 47
  9.5. Implementation details of WSDL-S ....................................................................................... 48
    9.5.1. WSDL basics ....................................................................................................................... 48
    9.5.2. WSDL-S extensions .......................................................................................................... 49

10. Comparison of the initiatives ..................................................................................................... 50
   10.1. WSMO vs. OWL-S ............................................................................................................... 50
   10.2. OWL-S vs. WSDL-S .......................................................................................................... 52
   10.3. WSDL-S vs. WSMO .......................................................................................................... 53

11. Evaluation of the initiatives to the high-level requirements ....................................................... 54

PART 3 – Semantic Web Services for location services .................................................................. 57

12. The context of prototype implementation .................................................................................. 57

13. The traveller scenario ................................................................................................................. 58
   13.1. Requirements for the scenario .............................................................................................. 59
   13.2. Actors in the scenario ............................................................................................................ 59
   13.3. Participating services in the scenario .................................................................................... 60

14. Implementation of real world service for the scenario ............................................................... 60
   14.1. Services from Telenor PATS lab .......................................................................................... 60
   14.2. WSDL file for location service ........................................................................................... 62
   14.3. Ontology description for location services ......................................................................... 65
     14.3.1. Using editors for domain ontology development ............................................................ 67
   14.4. Services description for location services .......................................................................... 70
   14.5. Comparison of development through editors ................................................................... 73

15. Costs of semantics for Web Services .......................................................................................... 73

16. Conclusion and further work ...................................................................................................... 75
References.......................................................................................................................................................... 77

APPENDIX A – WSML, OWL-S WSDL and WSDL-S files ...................................................................................... 81

I. WSDL for location.............................................................................................................................................. 81

II. WSMO Domain Ontology.................................................................................................................................. 83

III. WSMO Location Service file .......................................................................................................................... 85

IV. OWL location ontology file ............................................................................................................................ 86

V. OWL-S location service file including the ontology .......................................................................................... 89

VI. WSDL-S location service file .......................................................................................................................... 93
List of Abbreviations

ASG  Adaptive Service Grid
DERI  Digital Enterprise Research Institute
DNS  Domain name system
GB  Gigabyte
GSM  Global System for Mobile Communications
HTTP  Hypertext Transfer Protocol
IP  Internet Protocol
LAN  Local area network
OASIS  Organization for the Advancement of Structured Information Standards
OWL  Web Ontology Language
OWL-S  Web Ontology Language – Service ontology
PATS  Program for Advanced Telecom Services
RDF  Resource Description Framework
RPC  Remote Procedure Call
SEKT  Semantically Enabled Knowledge Technology
SOA  Service-oriented architecture
SOAP  Simple Object Access Protocol
TCP  Transmission Control Protocol
UDDI  Platform independent Universal Description, Discovery and Integration
UMTS  Universal Mobile Telecommunications System
URI  Uniform Resource Identifier
RAM  Random access memory
VPN  Virtual private network
WAN  Wide area network
WSDL  Web Services Description Language
WSDL-S  Web Service Description Language – Service
WSMF  Web Service Modelling Framework
WSML  Web Service Modelling Language
WSMO  Web Service Modelling Ontology
W3C  World Wide Web Consortium
XML  Extensible Markup Language
Abstract

To see if Semantic Web Services is a viable solution for the future of Web Services I take into consideration the existing technologies their advantages and deficiencies. From this analysis I recognised the need for machine understandable semantics in Web Service technology. For adding such semantics to services there needs to be requirements set. Since there are several initiatives under development for implementing such services I start by setting up a list of requirements. Then the different initiatives are introduced, evaluated against each other followed by an evaluation against the requirements. For a full evaluation of the initiatives I have implemented a location service from a traveller scenario in the different languages and evaluated them against each other. I consider the usability for developers in terms of available tools and the syntactical language the developers have to deal with. To finalise the viability analysis a cost effect analysis of implementation effort is included and shows that the Semantic Web Services is a viable solution for future Web Services.

In summary I see the three main outcomes of this paper as the evaluation of Semantic Web Service, requirements the different initiatives has to comply with and a real world example of Semantic web services. The initiatives considered are WSMO, OWL-S and WSDL-S.
**Introduction**

Traditionally the web has been a large information deposit. With the introduction of services available over the web it has been altered and a need for more efficient discovery and interoperability between different Web Services is emerging. Web services can exist as atomic services or as more complex services where several services are combined to offer a specific service. Services are registered in different deposits but have the problem of only locating services by keyword searches that have to be evaluated by a human user. This is why it would be beneficial to add semantics to such services for machine interpretation of the service content. This is where Semantic Web and Semantic Web Services come in to play. To make sure the evolution is in a direction appropriate for the future, there is need for criteria set so all research pull in the right direction. Different interest influences the criteria, both industrial and research interests.

This thesis starts in part 1 by introducing the background of semantics and adds this for use in web technology. In part 2 the criteria are set as a guide line of what is needed for Semantic Web Services language/framework. We evaluate the technology today and look at deficiencies in semantics for web technology. Then in part 3 three different language/framework initiatives are introduced through, WSMO, OWL-S and WSDL-S. These initiatives are then evaluated in comparison to each other and the requirements set. In part 4 we realise real world services in a semantic environment in form of a specific scenario of telematics, with appropriate WSDL-S, WSMO and OWL-S syntax for a part of the telematics scenario- a location service.
PART 1 Background for Semantic Web Services

We will start this chapter by introducing the World Wide Web (web), its technology and its deficiencies. Then Web Services and the implications this have on the web and its technology is introduced followed by the basics for the evolution to better web is introduced by the basics of semantics and ontologies.

1. Web technology

To start the evaluation of Semantic Web Services we start by looking at the basics for all Internet technology. This is to some degree based on the documentation provided by Wikipedia [1].

In 1969 the first node of ARPANET, that will later evolve into the Internet, is launched at the University of California, Los Angeles (UCLA). In 1983 the first TCP/IP WAN was operational through a university network backbone. This net was later merged with commercial nets as the TCP/IP protocol gained its popularity. Tim Berners-Lee incorporated his idea of hypertext which was an idea to facilitate information shearing between researchers adjusted it to TCP and DNS technology and then the world wide web was a fact in 1989. The www started as a world wide accessible information storage where users where able to share digital information over great geographical areas.

As the web initially can be categorised as a large information repository, it has evolved to also include application to application communication through what is called Web Services. In the next chapter we shall take a closer look at Web Services and the advantages and disadvantages of Web Services.

2. Web Services

As mentioned the traditional use of the web is information sharing. The introduction of application to application communication has evolved the web into not only a repository for information but also a repository of services. This application to application communication is enabled through interfaces referred to as Web services. Issues and definitions of Web services are found in [2], [3], [4], [5] and [6]:

Web services use open protocols for the applications to communicate over networks. The interface is described in machine language and other systems interact with the Web Services through messages. A web service provides interoperability between different software applications which can run on different platforms/frameworks, and they can be combined loosely to achieve complex operations. Your applications/services can be published and found in registries and by this made available to the online community in UDDI registries both to users and other applications. The technology basis is XML and SOAP/HTTP for the data transportation. SOAP is an exchange protocol for XML based messages and uses normally HTTP for the transport. This implies that when a service is discovered it uses SOAP/HTTP for communication. Web services are simply built on the web browser standards and are available for all platforms that are able to use web browsers. The service is described in a common language (WSDL) so others can “understand” both the function offered and how to
The functionality can be advertised through depositing descriptions in a register much like the Yellow Pages.

The architecture of Web Services and its components is described in the figure:

![Figure 1 Web service model](image)

The elements shown in figure 1 are used in Web service deployment and consist of:

- **Service Broker** – Service providers advertise their services, and service requesters find services they need. This is done in a UDDI.
- **UDDI** - is as the name implies a “Platform independent Universal Description, Discovery and Integration”.
- **Service Provider** – Provides a service registered in the Service Broker. The service is described in a WSDL document.
- **WSDL** – The “Web Service Description Language” describes the Web service and how to communicate with the service. It is a public interface to the web service
- **Service requester** – is the application or user in need of a service.
- **SOAP** – protocol for exchanging messages over a network using HTTP. It is the transport protocol for interaction with the service wanted.

Web services and the companying technology have several advantages which makes web service an attractive solution. But it also has some disadvantages which imposes the need for further development.

### 2.1. Advantages of Web service

The prime advantage of Web services is the application to application communication available over the web and the interoperability this imposes across different platforms. It facilitates repositories of services accessible through a common language which makes the services easy to understand. The open standards and protocols which are based on text also imply that developers find it easy to deploy. The possibility of combining different services from different locations allows reuse and provides possible integrated services. By this it facilitates a distributed approach to application integration. Since the Web Services use HTTP it will function without changes to possible firewalls which can both be an advantage as well as a disadvantage, the disadvantage is the security issue this imposes.
2.2. Deficiencies of Web service

The major deficiency of Web services is how the services are made available to the public. The interface to a service through WSDL lacks machine understandable semantics. Services are not automatically accessed by client applications. It depends on human understanding of the service and is usually hard coded in the applications and needs manual alterations if changes in the landscape occur. Solutions today can not find services with the same semantic abilities and it does not support automatic service composition without human interaction. By adding machine understandable semantics we gain automatic service discovery and composition.

If we compare the Web service standards to more mature distributed standards it has the disadvantage that vendors are committed to the existing standards. Existing distributed standards for computing includes approaches as:

- **CORBA** – Common Object Request Broker Architecture which enable application written in various languages on various platforms to interoperate. Further details of CORBA are found in [7].
- **RMI** – Java Remote Method Invocation for applications performing remote procedure calls. Further details of RMI are found in [8].
- **DCOM** – Distributed Component Object Model from Microsoft for communication between distributed software components. Further details are found in [9].

In comparison to these approaches, Web Services may suffer from poor performance. This is connected to text-based formats as XML, but binary representations such as SOAP will increase the efficiency.

Web services as a basic functional implementation need an infrastructure for advertisement and enactment as shown in figure 1. The following chapter describes a specific implementation for Web services through the SOA framework.

3. Service-Oriented Architecture (SOA)

SOA is an architecture which may be implemented as Web services. In this chapter the implications of using the SOA framework and its components are introduced. Issues and descriptions of SOA are found in [10], [11], [12], and [13]. The following will correspond to the previous chapter.

As the use of heterogeneous technologies in businesses is a reality, it needs to be a way to still be able to use these systems. SOA is a framework for utilizing distributed services and it is a popular architecture since it offers interoperability across different platforms and by this loose coupling of services. It is not Web Services, but an architectural pattern and popular use of the SOA framework is Web services which in contrast to the architectural pattern are implemented services. The services may be owned by several providers and SOA helps users build composite applications from several sources. The platform and programming language independence for services through the use of common interfaces hides the implementation details and implies that applications are reusable by other service parts. A service written in C# can be used by a service developed in Java. To deploy Web services through the SOA framework we publish the interfaces in a service directory and assure the platform independence through the use of XML and SOAP. The interfaces can be used to combine
services to complex services without regard for the underlying platforms and programming languages.

The value of SOA is that the framework matches the different users’ needs to the available capabilities. Some key concepts of SOA are:

- **Visibility** – this includes descriptions of capabilities in syntax understood by the users. The service description includes interaction details, input, output, and associated semantics and conditions for using the service. The service description lets the user decide if the service matches their needs and decide if the user satisfies the requirements set by the service provider.

- **Interaction** – This includes the actual way of using the capabilities. It describes message exchange description, information exchange and invoked actions.

- **Effect** - is the result of the interaction. We refer to them as real world effects. The real world effects are described in the interaction and establish the expected effect of a capability. A service includes the capability to do work, the specification of the work and the offer to perform the work. In SOA they refer to the services as the mechanism that brings needs and capabilities together. SOA is a way of organising the capabilities in services and promotes reuse, growth and interoperability between different “owners” to utilize the full potential of building services as a result of different capabilities and needs for full utilization.

Figure number 2 gives a conceptual model of the SOA architecture. It complies with the figure shown in the Web service definition (figure 1) as Web services is a popular way of implementing SOA.

![Figure 2 The service-oriented architecture (SOA) [11]](image)

In figure 2 as for the Web service model (figure 1) it includes a service directory for publishing and finding published services. The service is registered at the service directory and the service consumer finds the needed service in the service directory. When the wanted service is located, the service consumer invokes the service at its location. The standards used for Web services in SOA are explained in the Web service chapter and includes WSDL, UDDI and SOAP.

SOA has several advantages and some disadvantages. The disadvantages drive the further development of Web service standards in the Semantic Web service standardisation.
3.1. **Advantages of SOA**

As I see it the main advantage of SOA for Web services is that it offers a solution to interoperability problems across different platforms and gives the ability to combine services to achieve the request from users through standard interoperable interfaces. Through this it is possible for businesses with already large investments in services development based on one platform to also deploy them as Web services and by this include the possibility to integrate them with other services and let users use the service from other systems.

3.2. **Deficiencies of SOA**

My analysis of SOA indicates the following deficiencies. The first deficiency of SOA is the lack of semantic search possibilities. When a service is requested the search for it is keyword based. It includes the basics of combining atomic services to facilitate the needs of service consumers but lacks the ontology semantics base needed for machine interpretation of the service capabilities.

If the capability of the services is to be semantically understood we need to add semantics that is machine understandable. The first step towards this is the evolution of the World Wide Web to the “World Wide Semantic Web”.

4. **Semantic Web**

The World Wide Web as it is today is primarily based on documents written in HTML which is basically for visual presentation of context. The explicit meaning is not feasible for machines and they do not have the ability to understand the meaning of the context. The goal of the Semantic Web is to have the machines “understand” the content of web elements. It is not a new web but we can say that the traditional web will by this evolve to a Semantic Web. Issues and descriptions of Semantic web are found in [5], [6], [14], [15], [16] and [17].

By my evaluation the Semantic Web is easiest explained by the use of search engines and the results they give. If the user searches for a specific topic the results may include several non relevant hits since it has no “understanding” of the context. This problem is addressed by the Semantic Web solution by using descriptive technologies to provide descriptions of web pages. This way the search engine could narrow the search to the semantic correct pages. The Semantic Web has structured collections of information and sets of rules that they can use to conduct automated reasoning. Semantics gives structure for software agents to perform sophisticated tasks for the user and makes the information on the web machine comprehensible.

To reach this the Semantic web uses the current web and adds semantics to information to give a defined meaning. This is done by using ontologies as vocabulary and through this computers and users work in cooperation. To define the ontology vocabulary we need to take a look at what semantics and ontology is in this context.

Semantics is defined as the study of meaning, and it is described by ontology which is the study of existence. We can see semantics as a contrast to syntax which gives the combination of letters to denote a word. The semantics of a word is the meaning the word gives us. Ontology is based on questions like “What exists?” Ontology is a concept from philosophy
which denotes how to describe all existing things. In computer science we split this concept of one definition of all existing thing to several ontologies to define different domains. Ontology is a formal specification that describes a domain of terms and relations between the terms. The terms can be described as classes of objects in the domain. The relations are typically hierarchies of classes, with classes as parent classes and subclasses. A mere classification of all existing things is called taxonomy. To extend it so it gives meaning of the classifications we describe the classes by information as properties (cats’ eats fish) and value restrictions (only animals living in the sea are fish). We can also take the description further by adding disjointness between classes (dogs are not cats) and logical relations between the objects (Dogs have tails). By adding this additional information the taxonomy equals an ontology which gives semantics to the meaning of the different objects of the ontology.

In the following figure we can see an example of an ontology/taxonomy for a part of the “Animal” domain.

Figure 3 Example ontology for the “Animal” domain

Figure 3 shows the hierarchical definition of classes in the domain Animal. Both fish and mammals are animals. Horses, cats and dogs are mammals and by this also Animals. This ontology includes relations between the elements and can through syntax for ontology description be used to define semantics usable for machine interpretation.

Figure 4 The model for Semantic web [17]

Figure 4 describes the layering of the Semantic Web as described by Tim Berners Lee. The different layers include the basic data format for interoperability between applications. Modelling of classes, properties, domain, range, comments labels etc is covered by the stack to include ontology information. There are different rules to represent relations and logic for logical expressions. The Semantic Web has the top layer of trust which implies that the
information gained form the web might not be truthful but applications have to consider the truthfulness and build a web of trust.

The Semantic Web has its advantages and deficiencies. And it indicates also the use of semantics as basis for Web Services.

4.1. Advantages of Semantic Web
My analysis of advantages for the Semantic web set the main advantage to be the shift from human to machine consumable content of the information on the web. By adding content understandable for machines to search for information would become more efficient. As it is today the web is overflowing with information and you as the user have to eliminate the hits that are not of interest. Also if you search for one thing, others might call it something else and you would not get those hits. By adding semantics these hits would automatically be included and eases the user’s effort to get what he wants.

4.2. Deficiencies of Semantic Web
Even though the Semantic web has clear advantages I identify some deficiencies. The vide deployment of including ontology development as a part of the development process takes some time. Adding semantics to the development process adds extra cost. The benefits of doing this might not be clear to the developers, and still if they recognize the advantages they might not be the ones defining the budget for the development. Why mess with something that works? But the use of ontologies is getting more and more interest from the industry as standards are developed.

The Semantic Web indicates specific meaning for information on the web. One thing missing is the possibility to add ontologies to Web Services which brings us to Semantic Web Services.

Efforts to combine the Semantic Web with the world of Web Services have clear advantages and several initiatives are under development (OWL-S, WSMO and WSDL-S). By this we can avoid the keyword search problems and service composition and discovery problems of SOA.

5. Semantic Web Services
As the name indicates, Semantic Web Services are Web Services that combines Semantic Web and Web Services. Ontologies are used to make the Web Services understandable for computers and thus support automatic discovery, invocation and composition of Web Services as described in [18].

The following figure gives a visual description of how this works:
From figure 5 the elements of the architecture for Semantic Web Services are visualised. A service provider describes the services it offers. This service description is expressed in terms of ontology and is published in a Service Registry for users to find. It is described through semantics which is a part of the Semantic web. The description includes this semantics and is added as an addition to the Semantic web or from already described domains placed in the Semantic web. The service requestor creates a service request describing the service it needs which also includes semantic description of what service is needed/wanted. The request of the Service requestor is sent to a service discovery engine that locates the service wanted. In the Web service Integration platform there is in addition to the registry and the Discovery services a Service Invocation engine that communicate with the requestor’s client and the provider’s server. The Web service Integration Platform understands and uses the ontologies in the semantic Web to fulfil the requests from the requestor. As we see in figure 5 the ontologies used by the requestor, the provider and the Web Service integration platform understands the semantics and the semantics with its ontologies are located in the Semantic Web.

5.1. Advantages of Semantic Web Services
By adding semantics to Web Services we give the service explicit meaning, the discovery and composition of services is automatic. The semantics describing the services gives the machines the ability to understand the services, build complex services from user needs and if one service is not available another might be located without user assistance.

5.2. Deficiencies of Semantic Web Services
The deficiencies of Semantic Web Services are the maturity of the initiatives to solve the requirements. There are several initiatives (OWL-S, WSMO, WSDL-S and others) that all
pull in the same direction but with some what shifting focus on what’s the most important aspect of a full Semantic Web for Web services.

In the next part I will take a closer look at what the requirements are for Semantic Web Services from the different initiatives and combine them into a fully set of requirements to make this function as best as possible.
Part 2 – The Semantic Web Services initiatives

To achieve a well functioning Semantic Web with Semantic Web Services there has to be a set of requirements that the different initiatives has to follow. There are several views of what is important and not. The maturity of the initiatives also influences what the requirements are. The older and more mature initiatives may be some what set in their ways and stays true to the requirements they already have set and might not be willing to change them with the notion of what works does not have to be changed. The newer initiatives have some altered views of what is necessary requirements and bases the “new” requirements on deficiencies they see in the older initiatives. Newer initiatives have learned from the mistakes made by older initiatives and some what shifted their focus on their way. So in this part of the paper I have taken all set requirements in to consideration and evaluated them.

Then the different initiatives are introduced and then I evaluate them to each other and to the high-level requirements set in chapter 6.

6. Requirements for Semantic Web Services

The definitions of Semantic Web and Semantic Web Services are explained in the previous chapters. Different initiatives for semantics in web technology exist and to evaluate the initiatives there needs to be set some requirements to evaluate them. The background for the initiatives plays an important role in the evolution of Semantic Web Services frameworks, both industrial and research efforts. As a step in gaining the online community to use the initiatives they are dependent on international standardisation organisations.

6.1. Standardisation of Semantic Web Services

W3C describes in [19] them selves as an international standardisation organisation with member organisations and public contributors located all over the world for the creation of vendor neutral web standards. Their aim is to lead the Web to its full potential by setting standard protocols and guidelines to provide what they call “Web interoperability”. The standards are called W3C Recommendations and by this tries to avoid web fragmentation.

Some of the achievements of W3C include the standards HTML, XML, RDF and OWL [20]. W3C’s Semantic Web Services Interest group provides an open forum to discuss Semantic Web Services. Submissions by W3C members include [22]:

- Web Service Modelling Ontology (WSMO).
- Web Ontology Language for Services (OWL-S).
- Web Service Semantics (WSDL-S).

The acknowledgement of a submission does not imply an endorsement by W3C. It only states that a submission has been made by a member.

In [21] other standardisation bodies involved in the Semantic Web Service area is included. One of them is the standardisation consortium Organization for the Advancement of Structured Information Standards (OASIS). It aims at driving development, convergence and adoption of e-business standards. It includes 500 members from 100 countries. OASIS has established a Technical Committee Process to govern the technical work. Several technical
committees exist that work on specific standards. In 2002, OASIS joined in a co-operation with several international standards organisations for coordinating the various standards activities related to electronic business. The organisations are the International Electrotechnical Commission (IEC), the International Organisation for Standardization (ISO), the International Telecommunication Union (ITU) and the United Nations Economic Commission for Europe (UN/ECE). Standards from OASIS are expected to have a high impact as the web community deploy these standards quickly.

The different initiatives includes there interpretation of the requirements need for Semantic Web services. In the following three chapters the issues in OWL-S, WSMO and WSDL is considered.

6.2. Issues in Web Ontology Language- Service (OWL-S)

OWL-S is the most mature initiative for Semantic Web Services and it is still evolving. Traditionally the OWL has made no effort to connect to other initiatives but is now working on this. OWL-S is described to support the following task types as described in [23]:

- **Automatic Web service discovery** for automatic localisation of services that can solve the users required task. As an example: As it is today, if a user wants to buy an airline ticket we would need a human using a search engine to locate a service, then read the information, execute the service manually and determine if it satisfies the buyers requirements. With OWL-S the requirements could be specified as semantic mark up at the service web sites and a registry could locate the service automatically. OWL-S enables advertisements of properties and capabilities for automatic service discovery.

- **Automatic Web service invocation** for automatic invocation of a service given the description of the wanted service rather than having an agent pre-programmed to call one particular service. The user can then choose to buy a ticket from the wanted airline and a specific flight. OWL-S mark up of Web Services provides a computer-interpretable API that includes the semantics of the input and output. OWL-S and the ontologies in OWL provide standard means of specifying such APIs for automated web service execution.

- **Automated web service composition and interpretation**. Given a description of the users needs, this task selects, compose and interpret the services to perform a complex task. If a user wants to make all travel arrangements for a trip this has to be done manually with current solutions. With OWL-S the information necessary to select and compose services will be encoded at the service Web sites and software can be written to manipulate these representations together with a specification of the objectives of the task to achieve the task automatically. To support this, OWL-S provides specifications of input and output of the individual service, and a language for describing service compositions and dataflow interactions.

The requirements set for the development of the task types listed above include in [24] and is described here as:

- **Interoperability**. To ensure loosely-coupled components that work together at different platforms and with different programming language.

- **Reliability**. The web service architecture has been reliable to ensure further evolution in the Semantic Web Services evolution.

- **Integration**. Semantic Web Services technology should be integrated with the web and the Semantic Web. All conceptual elements should be addressed through a URI.
Security. Security is an important part for online processes in consideration to denial of service, authentication, authorisation, confidentiality, integrity, non-repudiuation, auditing and privacy protection.

Scalability. By requiring modular components scalability is ensured. It also reduces complexity and eases the development of such components through logical, consistent and easy to understand architectural basis.

Transportation. Requires transportation of data and access to data to and from the Web services.

Architecture. The architecture has to meet the need of the user community through standard definition languages and standard terms.

6.3. Issues in Web Service Modelling Ontology (WSMO)

[25] and [26] describes issues in WSMO and states that WSMO requirements are based on the requirements set by Semantic Web Service Language (SWSL) committee. The SWSL committee aims to identify and develop technology to provide a foundation for the future Web services consistent with the Semantic Web. It is to be consistent with emerging commercial work on Web services but also place much emphasis on developing a long-term foundation for the future of Web services. The committee aims at developing a formal language for specification of Web services, which will support automatic discovery, selection, composition, negotiation, invocation, monitoring, and recovery from failure. It is based on the elements of OWL-S and revises some new requirements based on the deficiencies in OWL-S. The requirements are:

- **Advertising and matchmaking (Automatic discovery).** Services are discovered based on the users goals/needs. This is done by a matching of the user goals and the capability of the service. Discovery is done through semantic descriptions and the goals of the user have to be formally described.

- **Negotiation and contracting.** When the service is discovered we have to make sure that the user actually can use the service. Negotiation and contracting is enabled by formally expressed capabilities.

- **Process modelling.** For describing processes in Semantic Web Services and focus on the support of reasoning. It has a role in describing behaviour in contracting. Service composition is another aspect of modelling. Composition of services can be created by hand or as an automatic process. A Semantic Web Services description describes the activity that web service performs. It may be composed of several sub activities. A set of constraints may include ordering and temporal constraints like simple workflows, iterated processes, duration constraints, concurrency and conversations. Further we may have State constraints which include input, output, states etc. Also resource constraints and composition is a part of the possible constraints.

- **Process enactment.** This is for the user to monitor the execution of a composite service. A user might want to cancel a service. It also includes constraint checking during operation, authorisation, exception handling etc.

- **Formalisation.** To support the functionality requirements, the formalism is to support external language support and analysis and verification of properties.

- **Declarative semantics.** The language should have declarative semantics, language constructs may be order independent, and it should be extensible do it can evolve further.

- **Interfaces.** The language should be capable of expressing interfaces, functional and behaviour specification of software, simple workflow descriptions, incomplete service
specifications, constraints, temporal constraint satisfaction, exceptions, plans and planning domains, scheduling, security issues, policies, what-if reasoning, sub-activity compositions.

- **Meta properties.** The language should have the following meta properties: understandable by a broad range of users, usable tools should be available, integrated with standards as OWL-S, RDF, XML, SOAP, WSDL etc, address knowledge representation and programming language aspects of web service, interoperable with automatic reasoning techniques, capable of linking to remote service specifications and supporting analysis. Capability to interoperate with other programs, support multi-party activities, support taxonomies of services and service description, importing external ontologies, a general extendible annotation mechanism.

- **Design principles.** WSMO as any other framework for Semantic Web Services descriptions needs to integrate the basic Web design principle, the Semantic Web design principles and design principles for distributed service oriented computing for the Web. The design principles for WSMO includes by this in [27]:
  - **Web Compliance** through the use of URI for unique identification of resources and adopts the namespace concept. It supports the use of XML and decentralisation as recommended by W3C.
  - **Ontology-Based** with ontologies as the data model for all resource descriptions and data interchange. This allows semantic information and interoperability.
  - **Strict Decoupling** of resources for possible usage and interaction with other resources.
  - **Centrality of Mediation** for addressing the heterogeneity in open environments.
  - **Ontology Role Separation** user requests are separated from the available Web services.

### 6.4. Issues in Web Service Description Language –Service (WSDL-S)

WSDL-S takes a different approach to the requirements for service description. Requirements for web service semantics in WSDL-S [28]:

- **Use existing standards.** Build on existing Web Service standards. To take advantage of their interoperability abilities and wide spread use.

- **Support different semantic languages.** Mechanism for supporting different semantic languages like OWL and WSML. By keeping the semantics from the representation we gain flexibility for the developers.

- **Reference different ontologies.** Allow references to different domain ontology languages. It should allow multiple annotations for one service to allow a service to be discovered by multiple discovery engines.

- **Semantic annotation.** Support semantic annotation of Web Services with XML Schema data types since documents using XML Schema is wide spread. Parameters should support XML Schema definitions.

- **Mapping.** Provide mapping between Web Service Schema types and ontologies. By mapping XML Schema types to ontological concepts.

The requirements for WSDL-S take a simple and reasonable approach. By extending the WSDL standard which is widely used and importing ontologies from other initiatives to describe the services.
6.5. **High-level requirements**

In this section I have evaluated the requirements from the initiatives mentioned in the previous subchapters. They are combined with other requirements I have found to be both necessary and important for all initiatives. The requirements are prioritised as follows:

I. **Automatic Web Service discovery** for automatic localisation of services that may solve the requester's task from a semantic description. The service needs a means to be discovered and the Semantic Web Services language needs to include the advertisement in Semantic Web Services registries (UDDI, ASG….)

II. **Automatic service invocation and access** of a service through the semantic description of the wanted service. Computer-interpretable description of the service. The basic for Semantic Web Services and the Semantic Web is the possibility of computer understandable descriptions of services. To access service Semantic Web Services need interfaces for enabling access.

III. **Service composition** through selection of service and interpretation given a semantic service description.

IV. **Semantic Web Services Interoperability.** Interoperability between different environments through loosely coupled components. In addition the languages should enforce the possibility to connect several initiatives. A service language should have the possibility of including ontologies from other initiatives.

V. **Standardisation** well described. Vendor neutral to keep the technology available for all users. The Semantic Web Services language needs to be/become a standard for availability to users and for community deployment of the standard. Through standardisation the community takes technology in use, a joint force to keep the development of Semantic Web Services in the same direction is important.

VI. **Tools** are important to have automated tool support for development of Semantic Web Services. Through tool support companies can take the technology in use with ease. Therefore a tool with a simple interface will be of high importance. For example HTML got popular when tools for conversion of normal text files where in place.

VII. **KISS – Keep it simple stupid.** It is important to keep the technology simple to use and understand.

VIII. **Public available and reusable ontologies** so the users do not have to invent the wheel over and over again. The reuse of already existing ontologies is a big factor in usability and also in the form of cost effectiveness since the costs of developing Semantic Web Services drastically is reduced by using existing ontologies (see chapter 15 about cost/effect).

IX. **Ontology evolution.** For a Semantic Web Services system to be appropriate for development there should be no problems of scalability of the systems by adding more ontologies to existing ontologies.

X. **Unambiguous descriptions.** The description of the service and the semantics for the service has to be unambiguous.

XI. **Separation of consumer and provider.** By separating the consumer from the provider there is logical separation. This will hinder possible misunderstandings.

XII. **Clear business vision.** For the community to deploy the technology in their business, the benefits of employing Semantic Web Services have to be of value. The initiatives needs to include business directed scenarios for the industry to want to include it in their development process.

XIII. **Maturity** of the implementation language.
By these requirements I have a basis of evaluating the different initiatives. But first I introduce the different initiatives through background information, goals for the development, available tools and the conceptual elements and the implementation languages.

7. Web Service Modelling Ontology (WSMO)
In this chapter the details of WSMO is investigated. I start by looking at the origins and motivation of the initiative. Followed by a look at the high-level goals of WSMO and the conceptual elements WSMO consists of. To fully describe WSMO the implementation details are summarized.

7.1. Background information on WSMO
WSMO is described in [29]. The work with WSMO is founded by the European Commission under the projects DIP (Data, Information and Process integration with Semantic Web Services), Knowledge Web, Ontoweb, SEKT, SWWS, Esperanto, COG and h-TechSight. It is also founded by Science Foundation Ireland under the DERI-Lion project and by the Vienna city government under the CoOperate programme. WSMO is an initiative with focus on both industrial efforts and the aim of delivering a long-term standard for Semantic Web Services with more research background.

7.2. Goals of WSMO development
The mission of WSMO is to describe various aspects of Semantic Web Services. The main focus is to solve the integration problem through the simplest solution possible, solve all aspects of the integration problem and include semantics for execution. This is summarised as simplicity, completeness and executability and is stated to possibly provide a world wide standard through interoperation between industrial partners and research groups. Another goal is to align the efforts of WSMO to other current available initiatives and standards that focus on similar problems and eliminate problems in existing approaches. This is manifested in the SWSL requirements WSMO uses as a basis. The WSMO initiative uses case studies for showcasing the usability through application areas as banking, marketplace, supply chains etc. The WSMO ontology is to be easy to use by research groups, research projects, software developers and user communities in the Semantic Web area.

7.3. Tools for WSMO
There are several tools developed for implementation of Semantic Web Services through WSMO. This includes WSMO Studio from [61] which is a Semantic Web Services editor available as a set of Eclipse plugins, Protégé with the WSMO Tab plug-in from [30], WSML Rule Reasoner for WSML implementation from [31], WSMO4J from [32] which is an API and a reference implementation for building WSMO Semantic Web Services application. There are also further tools developed. I have used both the WSMO studio and the Protégé plug-in WSMO Tab. The Protégé plug-in did not function as well as anticipated. Experiences of using the WSMO Editor are shown in chapter 14.3.1.

7.4. Conceptual elements of WSMO
WSMO uses Web Service Modelling Framework (WSMF) as a starting point and refines and extends it as described in [33] and [34]. WSMF provides a model for describing Web Services
and the composition through ontologies for terminology, goal repositories for defining problems the Web services can solve Web Services descriptions to describe the aspects of a web service and mediators to solve interoperability problems. The framework is described by the following figure:

![WSMF (Web Service Modelling Framework) Diagram](image)

Figure 6 WSMF (Web Service Modelling Framework) [35]

Figure 6 shows the top-level elements of WSMO for describing Semantic Web Services. They are described in [35] and [36] as:

- **Ontologies.** To provide the terminology used by other WSMO elements. It describes the domains of interest. It defines a common terminology through concepts and relations between the concepts. To define semantic properties of relations and concepts it also provides a set of axioms defined in some logical language. WSMO ontology design is based on modularisation and de-coupling. This issues the probability of combining ontologies and resolving heterogenity problems between them.

- **Web services.** A Web service is described by functional, non-functional and behaviour aspects of a service. The Web service is a computational entity that can solve user goals. It includes capabilities, interfaces and the internal process of the Web service. The capabilities describe what functionality the service provides. The interface describes how it is used through choreography describing how the service is used and orchestration for realisation of service through several Web services. A Web service can have several interfaces and only one capability.

- **Goals.** For defining the problems that should be solved by a Web service. It is the representation of the objective the Web service solves. It represents the goals that potentially satisfy the users needs. The use of goals for fulfilling users requests are based on the area of Artificial intelligence, where intelligent mechanisms are used for finding appropriate services according to a goal. The requester defines a goal and through discovery mechanisms a suitable service is located automatically.

- **Mediators,** to overcome interoperability problems between different elements, WSMO includes the following four types of mediators:
  - **ggMediators.** A mediator for linking a source goal and a target goal (source goal to target goal linking).
  - **ooMediators** import and resolves any mismatch between the ontologies (ontology to ontology linking).
  - **wgMediators** links Web services to goals. The Web service (totally or partially) fulfils the goal it is linked to (Web service to source/target goal linking).
**wwMediators** links two Web services (Web service to Web service linking).

Mediators resolve mismatches, and aim at realising de-coupling of resources. WSMO components are always linked with a mediator.

These top-level elements support the goal of Semantic Web Services through this conceptual framework and use a formal language for semantically describing all relevant aspects of Web Services. This facilitates the automation of discovering, combining and invoking services.

WSMO aims at compatibility towards web design principals through namespaces from XML with all resources including a URI id and data types as XML Schema types. WSMO also include non-functional properties to specify all non functional aspects of all information items which is not logical descriptions. Core properties from the Dublin Core Metadata elements are e.g. title, creator, subject, description. All WSMO elements can include such core properties.

In addition to the conceptual model we need a formal language for writing down the annotations of the Web Services according to the conceptual model and the Web Service Modelling Language (WSML) is a family of languages which formalizes WSMO as described by [33].

### 7.5. Implementation details of WSMO through WSML

The implementation details of WSMO in [37] states that WSMO provides a conceptual model of Ontologies, Semantic Web services, Goals and mediators which give the conceptual grounding for Ontology and Web service description. WSML provides means to formally describe all the elements defined in WSMO (ontology/goals/Web Services/mediators).

WSML provides a formal syntax and semantics for WSMO to describe the elements defined in WSMO. It is based on formal logic and WSML consists of a number of variants based on the formal logic. WSML-Core corresponds to Description logic and Horn Logic while the rest of the WSML-variants are more expressive and is based on Description logic and Logic Programming. The WSML language has the different levels (just like for OWL):

- **WSML-Core** is the basic language and is the base for the other variants. It has the least expressive power and the main features are for modelling classes, attributes, binary relations and instances. The language also supports class and relational hierarchies. It supports data types and data type predicates.
- **WSML-DL** captures a major part of OWL-DL with a data type extension.
- **WSML-Flight** Features meta-modelling, constraints and no monotonic negation. It is based on F-Logic and is a powerful rule language.
- **WSML-Rule** It is an extension of the WSML-Flight and includes Logic Programming.
- **WSML-Full** It unifies WSML-DL and WSML-Rule with extensions to support the no monotonic negation of WSML-Rule.

WSML uses XML and RDF based syntax to exchange over the web and for interoperability with RDF-based applications. It also provides a mapping for interoperability between WSML ontologies and OWL ontologies through a common semantic subset of OWL and WSML [33].

The WSML document has the following structure:
WSML = wsmlvariant? namespace? definition?
Definition =  goal
          ontology
          web service
          mediator

The implementation description of the service file and the ontology file are separated. The following subchapters describe the details for the two parts and are based on the specification in [38]:

7.5.1. **Defining the web service, goal and mediators**

All elements of WSML include the header element. This is defined as:

\[
\text{header} = \text{nfp}, \text{importontology}, \text{usesmediator}
\]

This specifies non-functional properties, which can be used by the whole WSML document or for the individual elements.

\[
\text{header} = \text{nfp}, \text{importontology}, \text{usesmediator}
\]

We import ontologies to use in the web service specification.

\[
\text{header} = \text{nfp}, \text{importontology}, \text{usesmediator}
\]

Mediators link ontologies, goal, Web Services. We have different mediators for the different elements. ooMediator, wwMediator, ggMediator, wgMediator where the first letters (like oo for ontology to ontology mediation) indicates what it mediates.

The service specification follows the structure:

\[
\text{WSML} = \text{wsmlvariant}? \text{namespace}? \text{definition}?
\]

We start by denoting which WSML variant we use. This includes Core, Flight, DL etc. and what namespaces is used for unique identification.

\[
\text{WSML} = \text{wsmlvariant}? \text{namespace}? \text{definition}?
\]

The namespace consists of the name and the local name. This is equal to the namespace mechanism as in RDF. The namespace is an International Resource Identifier (IRI from the URI standard).

\[
\text{WSML} = \text{wsmlvariant}? \text{namespace}? \text{definition}?
\]

The different parts includes:

\[
\text{Definition} = \text{goal}, \text{ontology}, \text{web service}, \text{mediator}
\]

The elements of goal are capability and interfaces of the services:

\[
\text{goal} = \text{'goal' id}? \text{header}? \text{capability}? \text{interfaces}?
\]

A goal specification can couple to a namespace. This identifies where the service is deployed.

\[
\text{goal} = \text{'goal' id}? \text{header}? \text{capability}? \text{interfaces}?
\]

The header is described in the first part of this chapter.
This is a formal description of what functionality is requested and provided by the web service. It includes preconditions, post conditions to describe what is needed for a successful service delivery and what effect this will have.

A goal and a web service may request and offer multiple interfaces for where the service is to be accessed.

**Definition**

\[ \text{Goal} = \text{goal}, \text{ontology}, \text{web service}, \text{mediator} \]

(The specification of the ontology follows in the next chapter)

\[ \text{WebService} = \text{\textquote{webService} id? Header* capability? Interface*} \]

The web service includes an id consisting of a name and an IRI which serves as an identifier.

\[ \text{WebService} = \text{\textquote{webService} id? Header* capability? Interface*} \]

The header is described in the first part of this chapter.

\[ \text{Capability} = \text{\textquote{capability} id? header* sharedvardef? pre_post_ass_or_eff*} \]

The capability starts with an optional name and IRI as an identifier for the element.

\[ \text{Capability} = \text{\textquote{capability} id? header* sharedvardef? pre_post_ass_or_eff*} \]

The header is described in the first part of this chapter.

\[ \text{Capability} = \text{\textquote{capability} id? header* sharedvardef? pre_post_ass_or_eff*} \]

The variables used for preconditions, postconditions, assumption and effect for the capability.

\[ \text{Capability} = \text{\textquote{capability} id? header* sharedvardef? pre_post_ass_or_eff*} \]

Pre condition, postcondition, assumption and effect are defined through axiom definitions.

\[ \text{WebService} = \text{\textquote{webService} id? Header* capability? Interface*} \]

\[ \text{Interface} = \text{\textquote{interface} id? header* choreography* orchestration?} \]

The interface starts with an optional name and id.

\[ \text{Interface} = \text{\textquote{interface} id? header* choreography* orchestration?} \]

The header is described in the first part of this chapter.

\[ \text{Interface} = \text{\textquote{interface} id? header* choreography* orchestration?} \]

An optional choreography is an identifier for the choreography

\[ \text{Interface} = \text{\textquote{interface} id? header* choreography* orchestration?} \]

An optional orchestration is an identifier for the orchestration
Definition = goal, ontology, web service, mediator

oomediator = 'oomediator' id? nfp? importedontology? sources target? use_service
The oomediator starts with in optional name and id for the mediator.

oomediator = 'oomediator' id? nfp? importedontology? sources target? use_service
The mediator can include non-functional parameters for non-functional elements of the mediator.

oomediator = 'oomediator' id? nfp? importedontology? sources target? use_service
The oomediator imports ontologies and resolve heterogeneity issues. It is more powerful than the importsOntology statement used for importing ontologies into other WSML specifications.

oomediator = 'oomediator' id? nfp? importedontology? sources target? use_service
An ooMediator can include identifiers of ontologies and other ooMediators as a source.

oomediator = 'oomediator' id? nfp? importedontology? sources target? use_service
The ooMediator can only have one target, which might be the identifier of an ontology, a goal, a web service, a web service or an other mediator.

oomediator = 'oomediator' id? nfp? importedontology? sources target? use_service
Uses_service is used to identify a goal that describes a service that implements the mediation or a wwMediator that links to such a web service.

As for the ooMediator the other mediators include an optional name and identifier. And they include headers.

wwmediator = 'oomediator' id? header* source? Target? Uses service?
The wwMediator connects Web services and resolves heterogeneity problems between data, process and protocol. It can only have one source which may be the identifier of a web service or another wwMediator. The mediator can only have one target indicating a web service or another wwMediator. The uses service is the same as for the ooMediator.

ggmediator = 'ggmediator' id? header* source? Target? Uses service?
The ggMediator is described as the wwMediator but solves heterogeneity problems between Web Services.

wgmediator = 'wgmediator' id? header* source? Target? Uses service?
The wgMediator is described as the wwMediator but solves heterogeneity problems between Web Services and goals.

7.5.2. Defining the ontology
I have taken out the ontology specification in its own section to separate it from the service definition. It still follows the rules as mentioned above and therefore we include variant,
namespaces etc. and is in itself a valid WSML document. But I find it more understandable and logical to separate them in two parts. This also facilitates the reuse of ontologies to some extent. The elements not discussed above for the ontology definition is as follows:

\[
\text{ontology} = \prescript{}{?}\text{ontology}', \text{header}*, \text{ontology element}*
\]

The specification is identified by the ontology keyword that can be followed by an IRI.

\[
\text{ontology} = \prescript{}{?}\text{ontology}', \text{header}*, \text{ontology element}*
\]

The header is described in the first part of chapter 7.5.1.

\[
\text{ontology} = \prescript{}{?}\text{ontology}', \text{header}*, \text{ontology element}*
\]

The ontology element includes:

\[
\text{ontology element} = \text{concept}, \text{relation}, \text{instance}, \text{relation instance}, \text{axiom}
\]

Here we define concepts of the ontology. What real world objects exist in the domain of interest.

We have concept, sub concept and super concept. The concepts are defined by name, non-functional properties and attributes.

\[
\text{ontology element} = \text{concept}, \text{relation}, \text{instance}, \text{relation instance}, \text{axiom}
\]

The relation couples the concepts and identifies the relation between them.

\[
\text{ontology element} = \text{concept}, \text{relation}, \text{instance}, \text{relation instance}, \text{axiom}
\]

Instances specified in an ontology are those shared as part of the ontology. It is instances of the concepts defined.

\[
\text{ontology element} = \text{concept}, \text{relation}, \text{instance}, \text{relation instance}, \text{axiom}
\]

We can also specify instances of relations.

\[
\text{ontology element} = \text{concept}, \text{relation}, \text{instance}, \text{relation instance}, \text{axiom}
\]

Axioms includes logical expressions

---

7.5.3. **Grounding in WSDL**

The importance of grounding WSMO to WSDL is described in [39] and states that it is very important that Semantic Web service infrastructure is able to communicate with existing Web services described in WSDL. WSDL specifies as mentioned a syntactical contract of message formats of what is legal to send. WSMO describes the functional and behavioural aspects of the service. Grounding of WSMO in WSDL describes the relation between a WSMO description and a WSDL description of a service. WSMO descriptions will be used for discovering and automate composition of services. The grounding to WSDL will then specify which WSDL interface to use. The grounding also includes how to compose proper XML messages for the WSDL described service. It will also reverse the interpretation from XML to ontology instances.

In WSMO execution environments, only Web Services with a WSMO description are available for the operations of discovery, composition, invocation, etc. The first step in making a service with only a WSDL description available is to create its corresponding
WSMO description. This makes the service description available to execution environments but does not solve the problem of how the implementation of the service can be invoked.

When a service has both WSMO and WSDL descriptions we need to ground the WSMO description in the WSDL description so that a WSMO execution environment (following the choreography of the service) will be able to know which actual messages to send and/or expect to receive. The WSDL description will provide the networking details such as serialization by SOAP XML messages, or that the data can be sent as RDF triples as described in [40].

8. Web Ontology Language – Service ontology (OWL-S)

As for WSMO in the previous chapter I investigate the details of OWL-S. I start by looking at the origins and motivation of the initiative. Followed by a look at the high-level goals of OWL-S and the conceptual elements OWL-S consists of. To fully describe OWL-S the implementation details are summarized. The issues, descriptions and details of OWL-S used for this chapter are described in [23], [28], [41], [42], [43], [44] and [45].

8.1. Background information of OWL-S

OWL-S is the most mature and probably most used semantic service description language. It is a submission to W3C from France Telecom, University of Maryland, National institute of Standards and Technology, Network Inference, Nokia, SRI International, Stanford University, Toshiba Corporation and University of South Hampton.

8.2. Goals of OWL-S development

OWL-S builds on top of OWL and is by this an OWL based Web Service Ontology and follows the layered approach of markup language development as OWL-S builds on OWL. It originates from the DARPA Agent Markup Language (DAML) program. The service ontology aims supporting discovery, composition and invocation of semantic Web Services by adding semantic descriptions for agents to handle this functionality. It provides a core set of markup language constructs for describing Web service in an unambiguous, computer interpretable form. There design decision is to add semantics and still comply with the industry standards of WSDL and UDDI.

8.3. Tools for OWL-S

Tools for OWL-S include the OWL-S Protégé-based Editor from [46] which provides capabilities to create and maintain OWL-S service descriptions. It is implemented as a plug-in for the Protégé editor. The OWL-S Editor that includes a wizard for building OWL-S description. Other tools include OWL-S Matcher from [47], OWL-S plug-in for Axis from the Technische Universitaet Berlin, Semantic Web Author from [48]. There are several more tools available for OWL-S implementation.

I have used the Protégé tool and my experiences are evaluated in the chapter 14.3.1.
8.4. Conceptual elements of OWL-S

OWL-S is used together with OWL to define Semantic Web Services. OWL-S is a Semantic Web markup Language to establish a framework for service descriptions, whilst OWL is an ontology description language. OWL uses XML and XML Schema for syntax and data type structuring. RDF is a data model for objects and relations between the objects and a RDF Schema as a vocabulary for describing properties and classes. OWL provides three languages for increasing expressiveness [43]:

- **OWL Lite** which gives the syntax for classification hierarchy and simple constraints. It is less complex than the following languages.
- **OWL DL** is somewhat more expressive but still includes all the features from Lite.
- **OWL Full** is the most expressive language and would prove difficult for any reasoning software to support complete reasoning. It includes the basics from Lite and DL.

The service ontology in OWL-S helps organise the three main ontologies for OWL-S and the service and the three parts of this includes [23]:

- **Service**. The service provides organisation of the parts of a Web service. It ties the parts together by using properties like presents, describedBy, and supports.
  - **Service Profile**. (presents) Describes what the service provides for the user. It is a profile used for advertising the service.
  - **Service Model**. (describedBy) The process describes how the service is used.
  - **Service Grounding**. (supports) Describes how to interact with the service with details of transport protocols.

![Figure 7 The OWL-S service model [23]](image)

In the figure 7 you can see how the service includes the profile, the grounding and the model for describing a service.

The parts of OWL-S have been described, and now the language descriptions of how this is implemented are shown in the following chapter.
8.5. Implementation details of OWL-S

The issues, description and details of OWL-S are found in [23] and [36]. Descriptions of Owl are found in [44].

8.5.1. The service class

One instance of service exists for each service. The Service profile provides information for an agent to find a service and evaluate if it meets the needs of the end user. It describes the ability of the service, its limitations as well as requirements of the service requester to be able to use the service. The ServiceModel and Grounding gives information about how to use the service when it is located. The service model describes how to ask for the service and what the service does. The grounding specifies the access details like port numbers etc. To link the service with its elements of Service Profile, Service Model and Service Grounding the service class uses the:

- **Presents** for linking it to the Service Profile.
- **DescribedBy** for linking it to the ServiceModel.
- **Supports** for linking it to Service Grounding.

Each instance of Service will present a ServiceProfile, be describedBy a ServiceModel and support a Service Grounding. (This is shown in figure 7).

8.5.2. The Service Profile

The primary role of the Service Profile is to assist discovery of services. It does not specify how it is used or invocation details. A service includes the three parties of requester, provider and infrastructure. The service requester is the user of the service, the provider is the party who deliver a service and the infrastructure is what the provider depends upon for the requester to locate the service. This is registries of services requester uses to find the appropriate service. The Service Profile defines a way to describe the services offered. An OWL-S Profile describes who provides the service, what functionality it offers and the service characteristics. The profile includes the description of the service, when this is located the user uses the process model to control the interaction with the service. The two should have a strong connection since they represent the same thing, but in OWL-S this is not defined so in practise the two may be inconsistent without affecting the validity of the other. If the two is mismatched the service will not function as anticipated.

OWL-S acknowledges the fact that other registries than UDDI is possible and by using declarative representation of the service in the service profile it can be used by different registries. In [23] they state it can be used by 28 types of registries from UDDI to P2P with no registry at all.

The profile model has four main parts:

1) A serviceProfile class with a high-level description of the service where the service is linked to the profile by presents and presentedBy.
   - **presents** describes that the service is described by the profile.
   - **presentedBy** specifies that the profile describes the service.

2) Human readable information about
   - **ServiceName** the name of the service that can be used as an identifier
   - **textDescription** a textual description of what the service offers and how it works.
• **contactInformation** as a referral to the provider responsible for the service.

3) Specification of the service functionality and the conditions for a successful result. Including expected and unexpected results of the service. It includes input and output to the service and the effect of the state in preconditions and effects. The profile ontology does not provide the schema to describe the parameters, preconditions and effect. They are defined in the process ontology, but the specification published in the profile is subset of those published by the process. Then the profile can just point to the process for the schema of the parameters. The profile ontology defines the following properties:

• **hasParameter** this class will not be instantiated but only states the domain knowledge explicit.

• **hasInput** input range as defined in the process ontology.

• **hasOutput** output range as defined in the process ontology.

• **hasPrecondition** specifies the preconditions defined in the process ontology.

• **hasResult** range of results as defined in the process ontology.

4) The attributes of the service other than the technical as quality guarantees provided by the service, classification of the service and additional parameters

• **serviceParameter** a list of properties. The properties include the **serviceParameterName** which simply is the name of the parameter followed by the **sParameter** that points to the value of the parameter within an OWL ontology.

• **serviceCategory** refers to an entry in a non-OWL ontology of services. It is denoted by a **categoryName**, the **taxonomy** pointer, a **value pointer in the taxonomy** and a **code** associated to a type of service.

• **serviceClassification** defines mapping to an OWL ontology of services.

• **serviceProduct** defines mapping from a profile to an OWL ontology of products.

8.5.3. **The Service model**

For details on how to interact with a service we include a subclass of **serviceModel**, Which is the **ProcessModel**. The process for interaction is not a program, but a specification on how a client can interact with a service. It is a detailed description on how the service is executed to for delivering the expected results. It dictates the input, output, preconditions and effects which is indicated in the service profile but is here defined explicitly. The service profile can point to the ProcessModel.

For a process to execute properly the preconditions have to be true, and a result of the execution results in various effects. The approach is to treat them as literals, like RDF literals which are defined in [49].

The Service Model is linked to the service class through the describes and describedBy properties.

• **describes** indicates that the service is described by the Service model.

• **describedBy** specifies that service model describes the service.

A processes is a choreography and includes three levels:
• **Atomic processes** are the evocable steps where an input gives an output. It corresponds to the WSDL operations.

• **Simple processes** are an technical abstraction of the process and can be used to denote processes that can be expressed in different ways.

• **Composite process** is the complete workflow and specifies how processes are combined to deliver a complex service. It includes a control flow of the processes included and a data flow of how the result of one process is used in other processes.

These can be described as a tree structure where the composite services are the internal nodes and the atomic processes are the leafs as shown in figure 8:

![Figure 8 Processes in OWL-S](image)

The definition of a process is characterised by four parameters:

- **Inputs** the service expect from the requester in the process execution.
- **Preconditions** which is required for the process to execute correctly.
- **Outputs** which is results of the execution of the process.
- **Results** are an conditional effect for handling exception handling, constraints on output and it also includes the real world effect.

The process is connected to the input, precondition, output and results following parameters:

- **hasParticipants** the process has at least two participants, the client and provider.
- **hasInput** in parameter to the process.
- **hasOutput** out parameter from the process.
- **hasLocal** local parameter in the process.
- **hasPrecondition** this has to be true for execution.
- **hasResult** is the effects as described.

**8.5.4. The Grounding model**

The grounding implies the access of the service through protocol and message formats, serialisation, transport and addressing. The grounding show how inputs and outputs are grounded in WSDL. The mapping is shown in figure 9:
Figure 9 Describes the grounding of OWL-S to WSDL. WSDL describes the service endpoints with appropriate communication protocols as HTTP and SOAP-RPC which aims to automate communication between Web services through abstract descriptions. SOAP is the basic for Web service message exchange. OWL-S Grounding supports SOAP message exchange through WSDL. In WSDL messages are normally described by a XML Schema. OWL-S implicitly defines message types in terms of OWL-classes for more semantically descriptions. So WSDL is purely a description of what services are available, where they are available and how they can be accessed. OWL-S includes a semantic description of the Web Services beyond their interfaces. It is a complementary that WSDL and the grounding in OWL-S provides the means to how they work together. So the OWL-S processes are abstract service specifications and the grounding specifies the interoperation details as communication protocol by mapping ontology elements to elements in a WSDL specification.

8.5.5. The OWL ontology

The OWL ontology is used to build ontologies to provide a language for describing classes and relations between them. It formalizes a domain through classes, and individuals and defines the properties of them so it is possible for machines to interpret. Whilst XML and XML Schema is a message format it does not provide a knowledge representation to support reasoning outside the context of a transaction. The exchange syntax for OWL is RDF/XML. OWL is designed to be compatible to RDF and RDF Schema, they are a part of the OWL standard. OWL is an extension of RDF.

The initial component of the ontology is the namespace to provide identifiers and the rest of the ontology representation is more readable. This might look like:

```xml
<rdf:RDF
    xmlns=http://www.marekatt.net/master/location#
    xmlns:location=http://www.marekatt.net/master/location#
    xmlns:base=http://www.marekatt.net/master/location#
    xmlns:owl="http://www.w3.org/2002/07/owl#"
```
This includes the namespaces for this ontology. The first indicates the default of unprefixed elements indicates the current ontology. Then a namespace for this ontology is added followed by the base URI for this document. The xmlns:owl indicates the OWL vocabulary. Since OWL depends on RDF, RDF Schema and XML Schema data types, they are also included to define elements drawn from this constructs.

By using the namespaces one could indicate the whole namespace by using only location:oslo.

Then through the owl:Ontology tag we include comments, version control and inclusion of other ontologies. It includes meta-data for the document:

```
<owl:Ontology rdf:about="">
  <rdfs:comment>An OWL location ontology </rdfs:comment>
  <owl:priorVersion rdf:resource="http://www.marekatt.net/master/old"/>
  <owl:imports rdf:resource="http://www.resource.org/Time"/>
  <rdfs:label>Location Ontology</rdfs:label>
...
</owl:Ontology>
```

The rdf:about="" indicates the empty element from the namespace that refer to the location ontology. The comment annotates the ontology. Prior version is for version control for systems using the ontology. Import includes the possibility to use other ontologies in this ontology. This might not always function if the imported ontology is removed or changed. Tools will respond to such changes. The Dublin Core metadata tags could be imported and includes e.g. Title, Creator, Description and Publisher.

Then the defining of the ontology starts. There are several elements used to denote the ontology. The first stage is to include the simple classes and the individuals. We need classes that individuals belong to and the properties they inherit as class members. We call individual members of the class the extension of the class. Class based reasoning is the basic for ontology reasoning. All classes in OWL are subclasses of owl:Thing. Domain specific classes are defined implicitly as a subclass of owl:Thing and domain specific root classes are defined by declaring a named class.

**Simple named classes:**

```
<owl:Class rdf:ID="LocationInfo"/>
<owl:Class rdf:ID="Accuracy"/>
```

The rdf:ID indicates that the class exists by name but they still do not give any meaning in the concept of the ontology. There will be added more information to the classes as we progress. The classes may now be referred in the document through rdf:resource="#LocationInfo", or by other ontologies through the complete form “http://www.marekatt.net/master/location#”. By using the rdf:about syntax you may also extend existing ontologies without modifying the original ontology document externally. This support the construction of a larger ontology.

**SubClasses:**

By using the rdfs:subClassOf we define the class further. If High is a subclass of Accuracy then every instance of High is also an instance of Accuracy:
We can see that the High class is restricted to be a subclass of the Accuracy class. The label is optional and is a human readable name of this class and the lang (language) includes the possibility of adding human readable names of the class in different languages. The label is only a comment and does not contribute to the logical interpretation.

In addition to describing the classes we also want to describe their members. This is a specific thing in the universe and is declared as a member of the class.

Or:

The rdf:type denotes the individual to be a member of a class.

The classes are described and now we need to extend this taxonomy by adding binary relations, also known as properties, to tie it together and add facts about the members, classes and individuals.

**Simple Properties:**

We have two kinds of properties:

- **ObjectProperty**: indicates relations between instances of two classes.
- **DatatypeProperty**: indicates relations between instances of a class and the RDF literals/XML Schema data types.

Where the domain is the Location and the range is Coordinate. From this we can infer that a coordinate is a location. As for classes also properties may be defined as hierarchies through the subPropertyOf element.

It is possible to expand the definition further through the owl_Restriction tag. Through this you can set the e.g. min and max cardinality as e.g.:

Where the domain is the Location and the range is Coordinate. From this we can infer that a coordinate is a location. As for classes also properties may be defined as hierarchies through the subPropertyOf element.
We can also relate individuals to data types through data type properties as RDF literals or XML Schema data types. The data types are defined in [50].

We also include properties of individuals. And we further describe the properties through:

- **Transitive property** to indicate that if a property is set for X and Y and for Y and Z this implies that the property also is set for X and Z. This is set by:
  
  ```
  <rdf:type rdf:resource="&owl;TransitiveProperty" />
  ```

  Figure 10 describes the transitive property:

  ![Figure 10 The transitive property](image)

- **Symmetric property** is to indicate that the properties define “both” ways. It is defined through:
  
  ```
  <owl:ObjectProperty rdf:ID="Property">
  <rdf:type rdf:resource="&owl;SymmetricProperty" />
  <rdfs:domain rdf:resource="#X" />
  <rdfs:range rdf:resource="#Y" />
  </owl:ObjectProperty>
  ```

  Figure 11 describes the symmetric property:

  ![Figure 11 The symmetric property](image)

- **Functional Property** indicates the same value. If the functional property is used this implies that a element connected through the same property but to different (Y and Z) elements also implies that the elements are the same.

  ```
  <owl:Class rdf:ID="X" />
  <owl:ObjectProperty rdf:ID="hasX">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  ```
Figure 12 describes the functional property:

- **InversOf** \((X \implies Y)\) and \((Y \implies X)\)
- **InversFunctional Property** \((Y \rightarrow X)\) and Property\((X \rightarrow Y)\) implies \(Y=Z\)

Properties may also be further constrained on the range. This is done through property restrictions. This includes:

- **allValuesFrom** indicates that all instances of a class using this property are members of the denoted class.
- **someValuesFrom** indicates that at least one of the properties points to the denoted class.
- **Cardinality** states cardinality constraints as min, max and equals.
- **hasValue** indicates that a member of a class denoted by a hasValue property at least has one property value from the denoted resource.

**Ontology mapping:**
Ontologies need to be reused. Much effort is devoted to link classes and properties together and adoption of other ontologies are preferable. Merging the collection of the ontologies is difficult but very useful. This is done through setting classes to be equivalent and also properties to be equivalent. In OWL Full we can use the owl:sameAs to indicate classes to describe the same thing. The sameAs mechanism can be used for defying two individuals to be identical. As an opposite of sameAs we have the differentFrom and AllDifferent to indicate values to be distinct.

**Complex Classes and class expressions:**
To form classes OWL provides operations as union, intersection and complement. It also includes enumerated classes and disjoint classes.

9. **Web Service Description Language – Service (WSDL-S)**
Just as it is done for OWL-S and WSMO in the previous chapters I investigate the details of WSDL-S. I start by looking at the origins and motivation of the initiative then followed by the high-level goals and the conceptual elements. To fully describe WSDL-S the implementation
details are examined. The details of WSDL are based on the information from [28],[51] and [52].

9.1. **Background information of WSDL-S**
The WSDL-S initiative is from IBM and the University of Georgia. It is based on the WSDL standard. WSDL is an XML-based language used to describe Web services, their location, and how to access them through their operations.

9.2. **Goals of WSDL-S development**
WSDL-S builds on the WSDL standard. WSDL-S is an extension of the syntactical level of WSDL, and includes semantic capabilities for Semantic Web Services. In WSDL-S we assume that an ontology model already exists and that they are maintained on the outside of the WSDL documents and is only referred to through extensibility elements. The ontologies referenced can be of any kind including WSMO and OWL. By building on the standard they assume faster adaptation from the user community.

9.3. **Tools for WSDL-S**
From the METEOR-S project at the LSDIS Lab at the University of Georgia the Radiant plug-in to eclipse offers a tool for producing WSDL-S documents. The Radiant plug-in provides a user interface for creating WSDL-S documents through an OWL ontology. The tool is located at http://lsdis.cs.uga.edu/projects/meteor-s/downloads/index.php?page=1. Otherwise there should not be great difficulty of extending the existing WSDL tools already available as they only need to include the extended elements from WSDL-S. In my implementation I have used the Radiant plug-in, and my experiences with this are shown in the chapter 14.3.1.

9.4. **Conceptual elements of WSDL-S**
From a WSDL description we can syntactically see what type the parameters are, but there is no further meaning of what effect this has on the real world. Semantic markup as in WSMO and OWL-S has this functionality. To avoid some of the problems with the other initiatives WSDL-S is a expansion of the WSDL standard. By building on the descriptive capability of WSDL we reference the input, output, preconditions and effects to semantic concepts in one or more external domain model.
Figure 13 Association between WSDL and ontology elements [28]

Figure 13 shows how the WSDL file associates the elements of the WSDL file to an external domain model. The external model can be of both WSMO and OWL-S specification.

9.5. Implementation details of WSDL-S

When starting the development of a WSDL-S file there are often available WSDL documents for the service. If not we always need to describe it as a WSDL file and then extend it to WSDL-S. Therefore I start by giving an overview of WSDL elements and then add to this through the details of the WSDL-S extensions.

9.5.1. WSDL basics

The WSDL document defines services as a collection of ports. It abstracts it from the concrete network deployment. Messages are abstract definitions of the data being exchanged and the port type abstracts the operations. The basics of WSDL implementation include the following language constructs:

The WSDL document uses the following elements included in the definitions tag:

```
<wsdl: definition name=“”? targetNamespace=“uri“>
  <types>
    <xsd:schema .... />*
    Defines the data types the web service uses. WSDL uses XML Schema syntax to define the data types so its platform neutral.
  </types>
</definitions>
```

```
<wsdl: message name=“msgname“>
  It defines the data element of an operation used by a web service. Each message has one ore more parts. We compare them to parameters in function calls in traditionally programming language
</message>
```

```
<wsdl: portType name=“portname“>
  The portType defines the entrance point to a web service. It describes the interfaces to the web service. This is the most important element as it describes the service, the operations and the messages involved. It is compared to a function library as in traditional programming languages. The operation types are defined as:
  • **One-way** – The operation receives a message but will not return a response. There is no output defined.
  • **Request-response** – In this case the operation can also return a response. Both input and output is defined.
  • **Solicit-response** – the operation can send a request and waits for a response.
  • **Notification** – the operation only notifies by sending a message and will not wait for a response.
</portType>
```
<binding type="type" name="name">  
Defines the communication protocols and message format the web service uses for a  
web service. The binding element has two attributes, name and type. The type points  
to the port of the binding. The name defines the name of the binding and is up to your  
discretion to choose an appropriate name of the binding. For example you can bind it  
to the SOAP protocol for communication. After the binding and its attributes is set  
you bind it to SOAP by:  
<soap:binding style="document/rpc"  
transport=http://schemas.xmlsoap.org/soap/http />  
Here the attributes are style and transport. The style attribute can be rpc-remote  
procedure call or document. The transport attribute defines in this case the SOAP  
protocol and HTTP.

In a WSDL document an endpoint (port) is defined for specifying an address. And a service  
groups a set of related ports together:  
<wsdl:definitions .... >  
   <wsdl:service name="nmtoken"> *  
      <wsdl:port name="nmtoken" binding="qname">  
      </wsdl:port>  
   </wsdl:service>  
</wsdl:definitions>

The WSDL-S uses the WSDL basics and extends it to include semantics.

9.5.2. WSDL-S extensions

The WSDL basic elements are described and now the extensions to add semantics to the  
WSDL file will be described. WSDL 2.0 includes service descriptions by interface, operation,  
message constructs deal with the abstract definition of the service with service and endpoint  
constructs deal with service implementation. The extensible elements and attributes are:

- **modelReference** for specifying a reference between a WSDL entity and a concept in a  
  semantic model.
- **schemaMapping** is for handling structural differences between schema elements of a  
  Web Service and semantic model concepts.
- **precondition** and **effect** are new elements used for service discovery. Precondition  
defines a set of conditions that must be met for an operation to be invoked. An effect  
defines the result of an operation. To keep it simple, each operation may only have  
one precondition and one effect. It is implied that the underlying semantics expresses  
complex preconditions and effects.
- **category** is an interface element for semantic categorisation to be used in an UDDI  
  registry. It aids service discovery when published in a registry. The semantics of the  
categorisation is the responsibility of the registry holder.

10. Comparison of the initiatives

The initiatives are described in the previous chapters and now the initiatives are evaluated  
against each other.
10.1. WSMO vs. OWL-S

Both the WSMO and OWL-S initiatives aim at realizing Semantic Web Services and improving cost effectiveness, scalability and robustness of current solutions. They take different approaches to fulfil the same goal as described in [36], [45] and [53]. They share the vision of the use of ontologies for automatic discovery, interoperability, composition etc. of Web services. But they differ in the way this is achieved. Both initiatives include, although different syntax for declaration of input, output, precondition and results associated with a service. OWL-S is more mature but WSMO presents advantages in the separation of requester and provider and includes orchestration of Web Services for dynamic use of services for more complex services. Also the mediators in WSMO are used to subvert the heterogeneity problems of distributed environments which are by OWL-s seen as a architectural and composition question not be handle internally in the ontology.

The major differences of OWL and WSMO are how they interpret reasoning for the logic behaviour. WSMO offers a rule language for Semantic Web Services as a result of what they see as shortcomings in other initiatives. As the WSMO initiative sees it the ontology language for OWL-S lacks in proper language layering. The OWL-Lite and OWL-DL is practically the same and the OWL-Full has no real reason for it other than complying with RDF Schema. The logic languages in WSMO Core maps to the logic language in OWL-Lite and is applicable to ontologies and logical expressions. The Core extends by rule language to WSML-Rule and WSML Flight. WSML-Flight is based on OWL-Flight and corrects what is seen by WSMO as problems in the model constructs. WSML-Flight and WSML-Rule is combined in WSML-Full and is not possible to map to OWL.

To look at the differences and the commonalities we have the WSMO elements:

- **Ontologies.**
- **Goal.**
- **Web services.**
- **Mediators.**

The OWL-S initiative has the upper ontology for services with the:

- **Service Profile.**
- **ServiceGrounding.**
- **ServiceModel.**

To evaluate the two to each other the elements of OWL-S is compared to WSMO elements:

The **service** concept in OWL-S links the service profile, the service model and the service grounding. This means to connect what the service is and how it is used. WSMO also includes a link between the capabilities of the service to its interface.

The **Service Profile** in OWL-S includes the purpose of both requester and provider in the service profile, WSMO separate the requester and provider so they can use different terminology and a mediator is used to mediate the differences. OWL-S service profile does the same as WSMO goals and capabilities and the non-functional parameters. The WSMO solution is more natural in the form that a requestor is interested in what the service can do for them rather than what the characteristics are for the service.

- **Separation of provider and requester.** OWL-S uses a single modelling element for requestor and provider while WSMO separates them in goal and capability.
• **Description of requests.** WSMO describes the request in goals and OWL-S uses profiles.

• **Non-functional properties.** WSMO uses a set of core non-functional properties accessible for all WSMO modelling elements and uses widely accepted vocabularies as the Dublin Core Metadata Element Set. OWL-S restricts the non-functional properties to the profile description and does not specify such standard vocabulary.

The Service Model indicates how to interact with the service. A service is considered a process and the process model is the base for this as they define the service as a process. The process model describes the functional properties and how to interact with the service. This corresponds to the capability of a WSMO Web Service and the interaction details of WSMO choreographies.

• **Model cardinality:** OWL-S links the service to the model through the property `describedBy/describes`. A service is described by at most one model. In WSMO it allows different ways of operation for one service whilst OWL-S only allows one way of interoperation.
  o WSMO allows multiple interfaces.
  o OWL-S allows only one model for interaction.

• **Functional description:** In OWL-S the functionality description is as the profile split into input, output, precondition and effect through the properties `hasInput`, `hasOutput`, `hasPrecondition`, and `hasResult`. The capability provides the functional description for similar properties in WSMO. The OWL-S profile and model are in WSMO merged into the web service capability.

• **Profile/model consistency:** Since OWL-S specifies a web service description in both the service model and the service profile. And consistency is not imposed there might be problems. If the descriptions do not match the service it might not function as intended but they will both be valid OWL expressions. In WSMO this are located in the same place and one will not run into such problems.

• **Atomic processes:** OWL-S separates atomic, simple and composite processes. An atomic process is invoked and executed in a single step from the requesters point of view. In WSMO an atomic process can be defined by the service capability. OWL-S simple processes are executed as one step. In WSMO it is described in the capability of the web service.

• **Composite processes:** OWL-S composite services may be broken down in to other processes. OWL-S includes control structure for this as sequence and split. It also includes a binding class for using data from other processes. WSMO models composite processes through choreographies.

The OWL-S Service Grounding. Service grounding is details of how to access the service from an abstract to a concrete specification of the service. Both initiatives grounds to WSDL.

**Other elements of interest** in the comparison of OWL-S and WSMO include:

• **Mediation:** WSMO is based on loose coupling and mediators for dealing with heterogeneity problems from the distributed environment. It facilitates the translation between different languages and is considered very important. This is something OWL-S does not facilitate, and it is considered as an underlying architectural problem. Therefore this is not included in the OWL-S language. They state it would not help in the basic functionality of discovery, composition and invocation processes and evaluate it as merely a selection problem. But the OWL-S initiative acknowledges the need for translation. Despite this they state that rather than stipulate the existence of a
new type of component in the web service infrastructure that OWL-S provides the Web services and their client the information that is needed to find existing mediators that can fix translation problems as described in [54].

- **Description of orchestration:** WSMO defines the orchestration of the service, describing what other goals have to be fulfilled to provide the functionality. OWL-S does not model this.

- **Languages:** OWL-S combines notations and semantics with OWL. WSMO provides a family of languages.

My conclusion is that both initiatives aim at the same goals with differences in how to approach a solution through different views of what logic reasoning is. OWL-S has the advantage of being more mature. WSMO has the advantage of seeing the problems of the mature OWL-S and “solving” them. Between the initiatives there is some what of a competition which in it self is a positive thing for more thought out solutions based on criticism from the other initiative. The initiatives have to coexist and the efforts from the WSMO mediators are very important to get interoperability. OWL-S has now stated that efforts are put towards interoperability with WSMO. From the developer of Semantic Web Services point of view, there is a need for good tutorials and user guides including good tools for easy development. OWL-S as the most mature language has a head start at this and the tools are somewhat better than the ones for WSMO. The available user guides/tutorials are also better for the OWL-S initiative, but this will be evened out as the WSMO initiative “catches” up to the OWL-S initiative.

### 10.2. OWL-S vs. WSDL-S

Both WSDL-S and OWL-S aim to support the use of semantic concepts in Web service discovery, interoperability and composition. Descriptions and issues in this area are found in [28], [45] and [55].

OWL-S includes its own ontology semantics through OWL whilst WSDL-S relay on external ontology semantics. One design goal of OWL-S is to take in to consideration the existing industry standards including WSDL. The WSDL operations represent atomic services of OWL-S and the grounding model describes the elements of the WSDL file. The process model indicates the orchestration of the atomic services to composite services. WSDL-S uses the OWL ontology (or other external ontologies) for ontological semantics and reasoning. The connection set by WSDL-S to a semantic concept is very similar to the grounding of OWL-S to WSDL. But WSDL-S sees the service ontology OWL-S as unnecessary complex and aims at providing a simpler approach to Web service descriptions without losing significant expressiveness. WSDL-S allows both semantic and non semantic descriptions of Web services. They admit that the description logic in OWL-S makes it a richer language for semantic representation but believe that extending the industry standards such as WSDL is a more practical approach. And with the referral to the external domain models it will bring the better of the two worlds together.

OWL-S is a mature initiative with rich semantic expressivity. Still it suffers from limitations. This including the profile model duplicates with possible multiple definitions for describing the same service. The second limitation is that OWL-S assumes that everyone uses OWL for representing ontologies. WSDL-S has one clear advantage over OWL-S. This is the user communities’ familiarity with WSDL. But WSDL-S initiatives extract the semantic domain models and take a more sceptic approach to ontology representation languages. By this,
developers can choose their ontology language themselves unlike OWL-S where OWL is the language you have to use. Then it would be easier to reuse existing ontologies in different ontology languages.

By my evaluation the most important advantage of OWL-S is the expressiveness of the language constructs. The greatest advantage of WSDL-S is that it is seen as a new layer of WSDL and WSDL is a industry standard much used. The WSDL is also a widely used technology and by adding extensions to the user community it would easier deploy in their systems. An other advantage of WSDL-S compared to OWL-S is the simplicity of the initiative as it is easy to use. Further it aims at solving the same as OWL-S but does it by combining already existing technology and benefits of the research of the chosen ontologies. Another major advantage of WSDL-S is the implicit mediation between different initiatives. The advantage of OWL-S is the maturity and the expressiveness in the representation language. The WSDL-S initiative will always depend on other initiatives design decisions and has to accept them as it is.

10.3. WSDL-S vs. WSMO

Some of the comparison of WSDL-S to OWL-S also applies to the WSDL-S to WSMO comparison. Both WSMO and WSDL-S aims at providing the same functionality, but have different approaches. WSDL-S covers the functionality of WSMO (as it does for OWL-S) and takes advantage of the different existing technology and uses this to fulfil the goals. In the following we can see how WSDL-S covers the WSMO functionality in [56]:

- The semantic model of a Web service includes the semantics of input and output parameters, preconditions and effect. The WSMO model where a Web service has a capability including preconditions, assumptions, post conditions and effects. It also includes an interface with choreography and orchestration. All data is described by ontologies.
- Both WSDL-S and WSMO link XML Schema elements to ontology concepts.
- A WSDL interface in WSDL-S compares to the non functional properties of subject and type in WSMO.
- The WSDL-S interface splits the operations with preconditions and effects. In WSMO this is not split and the Web service is described as a whole in the capability. To comply with this the WSDL-S precondition and effect should be made available in the WSDL interface to cover both post condition and effect in WSMO.
- WSMO models choreography and orchestration of a Web service. This is not covered in WSDL-S. This needs a solution.
  - WSDL-S should extend with a choreography element with structure similar to post condition and effect elements. For use with WSMO this elements should refer to a WSMO choreography. It is worth mentioning that choreography for description of the constraints of the use of the operation is an alternative to preconditions and effects which may be described in WSML and referenced from WSDL-S using existing mechanisms.
  - Orchestration is an aspect of the implementation of the Web service and not of the public interface for the clients to use. WSDL-S should link from WSDL service to WSMO Web service since WSMO Web service is a semantic model of the WSDL service.
- Grounding in WSMO is intended to be able to ground WSMO to any type of grounding. WSDL-S is independent of any semantic language and includes extensions to ground WSMO descriptions so it can be used for those grounding elements in
WSMO. For every WSDL-S link we find a WSDL interface and the link goes from the concept to the WSDL interface.

The conclusion for this comparison is that WSDL-S covers to a high degree the functionality of WSMO and extensions to cover the deficiencies are proposed. The clear advantage for using WSDL-S is the already established use of WSDL in the user community and by adding extensions to it makes the transition form WSDL to WSDL-S easier. But on the other hand we need external ontologies to be defined. If it does not already exist the WSDL-S developer has to provide it and need to learn another language as OWL or WSML to write it. From my point of view the WSDL-S initiatives simplicity is the best feature of the initiative. The mapping to already existing ontology technology makes it a good alternative by not adding an entirely new ontological language to the mix. It facilitates the goal of interoperability between different initiatives and allows by this several solutions for Semantic Web Services.

To fully evaluate the initiatives I will in the next chapter consider the initiatives to the requirements I defined in chapter 6.5

**11. Evaluation of the initiatives to the high-level requirements**

All three initiatives (OWL-S, WSMO and WSDL-S) aim to provide a Semantic Web Services standards. In chapter 6.5 high level requirements are defined for evaluating the initiatives. Through the study and comparison of the initiatives these requirements are here evaluated.

The first three requirements are general requirements all initiatives aim at solving. They provide some what different solutions but all fulfil the following three:

I. **Automatic Web Service discovery**
   All the initiatives include semantic descriptions for automatic localisation of services.

II. **Automatic service invocation and access**
   All initiatives include semantic descriptions for machine understandable process descriptions through interfaces for enabling services.

III. **Service composition**
   The semantic description of a service might be used for service composition.

Interoperability is an important property of Web services.

IV. **Semantic Web Services Interoperability**
   The services are loosely coupled components for all initiatives. Through the interfaces they can be used together to compose complex tasks. As for interoperability between the initiatives it differs. WSDL- S uses external ontology elements from different languages to achieve the semantic basis for the Web services. It can use both WSMO and OWL-S semantics. WSMO use the mediators to translate between initiatives and give the ability to include several semantic descriptions for the same Web service by including several mediators. OWL-S does not support this as well as it focuses on the OWL-S basis for the semantics. Still it recognises the problem and efforts on providing such mediation are now under development.

By international standardisation the online community interest and the possible adoption and use increases. And by this it is very important as the aim of all initiatives is to be accepted as standard.

V. **Standardisation**
OWL-S has gained the lead as it is built on the W3C OWL standard. And the maturity of the initiative gives it leeway in contrast to WSMO which employs the WSML for semantic description. Also the WSDL-S initiative gains an automatic advantage over WSMO as it builds on the WSDL standard and the wide uses of WSDL in Web services. All initiatives are submitted to W3C for evaluation and are still under development.

Tools that are easy to use are a very important aspect of getting developers to use the initiative.

XIV. Tools
All the initiatives include tools for developing Semantic Web Services. Some of the tools for the initiatives are listed in the chapters 7.4, 8.3, and 9.4. One for each initiative is then used in the development in chapter 14, and an assessment of them is included in the chapter 14.3.1. For OWL-S and WSMO there are several tools available, but the WSDL-S initiative does not include many alternatives but promises an easy convertment of WSDL tools to WSDL-S tools.

It is important to keep the technology simple to use and understand.

XV. KISS – Keep it simple stupid.
If you have an already existing ontology the WSDL-S is easy to implement as you avoid the ontology implementation. But if the ontology does not already exist you have to use one of the other alternatives. Tools simplify the implementation process for Semantic Web Services. But by implementing directly through the language constructs for the initiatives, my experience is that WSMO is easier to implement than OWL-S.

Cost reduction is an effect of public available ontologies.

XVI. Public available and reusable ontologies
WSMO and OWL both incorporate ontologies as components used in Semantic Web Services. As ontology definition is costly the use of public available ontologies are of great value (see chapter 15).

Ontology evolution is crucial for Semantic Web Services. Ontology as it initially was defined is an ontology of all existing things. The ideal solution is to have public ontologies interconnected to together define all existing things and have a “universal” ontology.

XVII. Ontology evolution.
The fact that the different initiatives solve the ontology descriptions in different ways indicates the need for mediation between them to interconnect to this “universal” ontology. WSDL-S incorporates both WSMO and OWL-S. WSMO uses mediators for such interconnection and OWL-S is now using resources to incorporate this in the development.

Description of services and how they are used has to be unambiguous for the service to function as intended.

XVIII. Unambiguous descriptions.
OWL-S separates the description of the service and the actual use of the service. If they don’t match the service will not function as expected. WSMO solves this by including both elements in the capability element. WSDL-S also includes this as one element.

Separation of provider and consumer adds a logical separation to the description of a service.
XIX. **Separation of consumer and provider.**
OWL-S includes the view of both consumer and provider in the same service profile. This is also the case of WSDL-S. This might contribute to confusion of the separation. WSMO separates the two in the goal of the user and the capability of the service.

For deployment of Semantic Web Services in industry there has to be a clear benefit of using Semantic Web Services in business.

XX. **Clear business vision.**
The business vision of using Semantic Web Services from the initiatives is taken to consideration by using business scenarios as a basis for the description of the initiatives. As an example uses OWL-S flight booking scenario for describing a business vision.

How mature the initiatives are has some importance of deployment in industry. As standardised solutions is more likely to be used.

XXI. **Maturity**
OWL-S is the oldest initiative and probably the most used. WSMO is not as well evolved but takes advantage of solving problems specific to OWL-S (as the mediation). WSDL-S is the newest initiative but has the advantage of the vide use of WSDL and the ease of including the WSDL-S extensions.

The initiatives are now evaluated against the requirements. But for a full evaluation there needs to be a real world implementation of a Semantic Web Services.
PART 3 – Semantic Web Services for location services

In the previous parts of this paper I have investigated motivation factors of Semantic Web service and the details of some initiatives for implementation of Semantic Web Services. The different initiatives were evaluated to each other and then evaluated in accordance to Semantic Web Services requirements.

To complete the study of Semantic Web Services a real world implementation is needed. The different initiatives are used for implementing the same real world service. The implemented service is a location service from a traveller scenario. I start by explaining the context of the prototype implementation, the scenario background and the requirements for the scenario. This is followed by the actual implementation and a cost analysis of Semantic Web Service implementation concludes this paper.

In regards to the traveller scenario the references [5], [14], [57] and [58] is used for details and background information.

12. The context of prototype implementation

There are several standards under development to support Semantic Web Services as shown in earlier chapters. A platform is under development to support Semantic Web Services through semantic specification, registration, discovery, composition and enactment of composed and atomic services in the ASG project.

ASG is an Integrated Project supported by the Sixth Framework Program of the European Commission under the Information Society Technology Objective Open Development Platforms for Software and Services. It includes 22 partners from seven countries including software, telecommunications and the telematics industry. It aims at building a platform for adaptive discovery, creation, composition and enactment of services. The ASG is a type of SOA system that integrates Web Services and Semantic web/grid. ASG uses the WSMO as the ontology language for defining Semantic Web Services semantics.

ASG’s goal is to offer an independent platform for Semantic Web Services. ASG positions itself as a service composition project. It is based on the SOA paradigm by using and expanding web service standards to achieve interoperability with different platforms. Services are available independent of the internal implementation and are used in the grid infrastructure or linked through a proxy to the ASG. All the services are called a service grid. ASG builds on top of this structure and offers semantic interoperability between different services in the grid. ASG offers both the infrastructure for composition of services as well as an extra level of resource handling.

The ASG project identifies three domains, these are Telecommunications, Enterprise IT and Telematics. The scenarios for the first two domains are described shortly in the next chapter. The telematics scenario is investigated in more detail and limited to a traveller scenario including the location service which is used for real world implementation. The location service is used where an application might need information about the users’ location. The implementation of the location service is implemented in ASG as a WSMO implementation. To evaluate the different initiatives in a real world implementation the location service is also implemented in OWL-S and WSDL-S.
13. The traveller scenario

The ASG Integrated project proposes different scenarios. This includes a service monitoring scenario for complex monitoring of telecommunication environment and a dynamic supply chain management to ease the internal supply chain in a ISPs IT infrastructure. There is an E-Government scenario for handling the complexity in taking decisions in government structures, and a scenario for a dynamic marketplace where users dynamically can get services and this way easily enable the user to have his/her own business. There is also a scenario for IT-infrastructure and a Scenario for Quality of Service in Enterprise IT. In my master thesis the scenario of telematics was most appealing and the traveller scenario is a part of this scenario. I evaluated the scenario as service ordinary users would get interested in and use. The service could be a positive influence on the users normal everyday and would have a wide consumer group and therefore large commercial potential. As a part of the telematics scenario my contribution is the location service semantics for the traveller.

Figure 14 The service in the traveller scenario

Figure 14 is based on Intelligent Transportation System/Services (ITS) supported travelling. ITS is a system intelligently dealing with transportation related things as vehicle, roads,
public transportation, travel information, traffic management, traffic emergency, etc. The figure shows the participating services. It includes the transition from house to car in the Pre-trip session handover. The traffic alert is triggered by traffic events. This can be reported by other travellers as well as Road authorities, police etc. The alternative route service starts if needed and the transportation service guides the driver to the destination.

The scenario is based on the need for a service delivered to commuters travelling in cities. My example assists people going to work by car. It is a location based transportation service where the user accesses the service through a mobile unit which guides the user to the destination in the least amount of time. The user can start at home checking the traffic. In the scenario [18] the service tells the commuter that the traffic situation is fine. When the user is half way to his destination point the service discovers that there has been a traffic accident 2 miles ahead and that there is traffic congestion. This is triggered by a traffic alert, and video of the situation can be streamed to the user. The user asks what to do next and the service responds with two alternatives. His first choice is to choose an alternate route travelling through downtown which always has traffic jams, the other is a long drive around. The user selects one of the two possible alternatives and is guided to the destination.

The scenario has the potential of reducing traffic chaos, improved navigation, safety, energy saving, pollution reduction and increased travel efficiency.

For describing the scenario in more detail the requirements of such a service is gathered in the following chapter.

13.1. **Requirements for the scenario**

Requirements for the services include that they are reliable, precise and timely. The system needs historic data about the users travelling patterns to be able to alert the commuter of traffic abnormalities. There is also a need for an advanced user interface with possible voice recognition and video streaming. The unit needs a Global Positioning System (GPS) to know where the commuter is so it can sort out relevant information and propose the best action. This is also necessary for calculating directions to alternative routes in case of accidents. Unusual traffic events can be caused by other things than just accidents. Also things like sporting events and road maintenance can cause problems. Unexpected situations can be detected by camera surveillance. Events should possibly be placed in ITS manually. Regular commuters should be able to report events as well as road authorities and the police.

From this the participants in the scenario is listed as actors in the scenario.

13.2. **Actors in the scenario**

The scenario has three possible actors using the system:

1. End service consumer
   The end service consumer is a commuter travelling to and from work. The user can retrieve services on a mobile phone.

2. End service provider
   The end service provider is a company offering access to services for the end service consumer. In this scenario it is a traffic information service provider which is aggregating information from other service providers. This actor can be car association like NAF and Via Michelin that currently provides rout planners, maps etc.
It could also be telecom operators like Telenor that would like to provide this service to their mobile phone customers.

3. Service Providers
The service providers are various providers of basic services as location service, traffic alerts etc.

The actors are defined and now the services needed for realising the scenario is listed in the following chapter.

13.3. Participating services in the scenario
In this case the participating services are:

- **Composite services:**
  - Information system keeping track of the traffic situation in the area
  - Automatic warning about unusual traffic events
  - Alternative routes

- **Basic services:**
  - Positioning information
  - Payment service
  - Location specific information (this is shown as a real world implementation in the following chapter.)
  - Weather report services
  - Map service
  - Route planning services

The services are defined and for the real world implementation I have chosen to implement the location service need for location specific information.

14. Implementation of real world service for the scenario
The ASG platform uses WSMO as the framework for adding semantics to services. Through my participation in the project I implemented the location ontology and the service description for location services in WSMO. For this paper I have also implemented the semantics for the services in OWL-S and WSDL-S to see how this compares. This chapter starts by looking at the different descriptions of the service. This is followed by a comparison of the implementation of the different initiatives.

The basis for the location service is the service offered by Telenor’s PATS lab. The following subchapter introduces PATS lab and the details of how I have connected the lab.

14.1. Services from Telenor PATS lab
Advanced Telecom service delivery has moved from vertical to horizontal service architectures. An example of such a lab is Telenor’s Innovation lab PATS (Figure 15).
PATS lab is a research initiative. Its vision is to provide a setting where students and researchers can experiment with advanced telecommunication services. In the PATS lab, location can be provided currently through four different units:

- **Mobile positioning** (POS) using the GSM/UMTS positioning in Telenor Mobil’s network.
- **Advanced GSM positioning** combining GSM location techniques with assisted GPS (A-GPS) provided by Telenor R&D.
- **WLAN positioning** from Telenor R&D, Radionor or Ekahau.
- **Touch-base location systems** using RFID or Near Field Communications (NFC).

This set-up presents the complexity of providing location, and even more the complexity of getting location exchange across networks, countries, and operators.

Figure 16 illustrates the connection to the location service from Telenor through a gateway to Pats lab.
The gateway is installed by a Trustixs firewall configuration. From Pats the service is accessed through ParlayX to Telenor Mobile and its GSM location service. ParlayX uses XML based messages for communication with telecom applications. To set up the Trustixs gateway we needed a PC with at least 128 MB RAM, two or more NICs and some GB of free space, and the Trustixs Enterprise Firewall installation setup. For the network connections (Ethernet to Internet and LAN) you need fixed IP addresses for both sides and the appropriate connection properties (IP for each interface, subnet mask, and default gateway for the external link). We also used connection information of the Pats lab side of the tunnel (gateway, subnet, authentication and password). To set up the gateway, the instructions from the setup files was followed and the basic firewall installation was used. To access the Telenor location service we then used the Trustixs Gateway to Pats lab through the IP address 193.156.96.70 (sumotrust.unik.no). The server which functions as the access to the Pats lab is a Windows 2003 Server with the IP address 193.156.96.146 (pats1.unik.no). Through a VPN to Unik (vpn.unik.no) and a Remote Desktop client we can access the Windows server and the service from Pats/Telenor from everywhere.

For the further development of semantics to the location service provided by PATS lab I used the WSDL file (Appendix I) for the development of ontology and service description.

14.2. WSDL file for location service
The WSDL file from Pats follows the definition of WSDL files listed in the WSDL subchapter (9.5.1). The full file is located in appendix I. We start by defining the namespaces, and XML Schema types used by the service:

```xml
<wsdl:definitions
    targetNamespace=http://www.csapi.org/wsdl/parlayx/terminal_location/v1_0/service
    xmlns="http://schemas.xmlsoap.org/wsdl/" ........ etc. ">
<wsdl:types>
    <schema
```
Then the messages for the service were defined. You can call them parameters for the operation, these are the variables needed for performing the service. In the WSDL files several exceptions are added in the definition and are not taken in to much consideration in this paper due to the focus of the location server. The messages of interest include the “getLocationRequest” and “getLocationResponse”.

The “getLocation Request “ is defined as:

```xml
<wSDL:message name="getLocationRequest">
  <wSDL:part name="endUser" type="tns2:EndUserIdentifier"/>
  <wSDL:part name="requester" type="tns2:EndUserIdentifier"/>
  <wSDL:part name="accuracy" type="tns1:LocationAccuracy"/>
</wSDL:message>
```

This uses the tns1 namespace for definition of the endUserIdentifier. The tns1 namespace is defined by the:

“...../parlayx/terminal_location/v1_0”

The other part is the LocationAccuracy which is identified by the:

“...../parlayx/common/v1_0”

The “getLocationRequest” message is written:

```xml
<wSDL:message name="getLocationResponse">
  <wSDL:part name="result" type="tns1:LocationInfo"/>
</wSDL:message>
```
The following part of the WSDL is the port type definition which is the interface to the service functions. The portType for the location service is the “MobileTerminalLocationPort” which includes the operation “getLocation” which has the operation named getLocation. The order of the parameters is set and the input and output message is set:

```xml
<wSDL:portType name="MobileTerminalLocationPort">
  <wSDL:operation name="getLocation" parameterOrder="endUser requester accuracy">
    <wSDL:input message="impl:getLocationRequest" name="getLocationRequest"/>
    <wSDL:output message="impl:getLocationResponse" name="getLocationResponse"/>
    <wSDL:fault message="……etc."/>
  </wSDL:operation>
</wSDL:portType>
```

Then the WSDL binding defines the communication protocol for the service which uses remote procedure call and SOAP for transport:

```xml
<wSDL:binding name="MobileTerminalLocationPortSoapBinding" type="impl:MobileTerminalLocationPort">
  <wSDLsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http />
  <wSDL:operation name="getLocation">
    <wSDLsoap:operation soapAction=""/>
    <wSDL:input name="getLocationRequest">
      <wSDLsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://www.csapi.org/wsdl/parlayx/terminal_location" use="encoded"/>
    </wSDL:input>
    <wSDL:output name="getLocationResponse">
      <wSDLsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://www.csapi.org/wsdl/parlayx/terminal_location/v1_0/service" use="encoded"/>
    </wSDL:output>
  </wSDL:operation>
</wSDL:binding>
```

Then to indicate the location of the service and the port used by the service we use the “wSDL:service”:

```xml
<wSDL:service name="MobileTerminalLocationService">
  <wSDL:port binding="impl:MobileTerminalLocationPortSoapBinding" name="MobileTerminalLocationPort">
    <wSDLsoap:address location="http://129.241.219.154/parlayx/services/MobileTerminalLocationPort"/>
  </wSDL:port>
</wSDL:service>
</wSDL:definitions>
```

This WSDL file is the basis for the ontology and service definition used to add semantics to the web service available through PATS. This is also the basis used for the WSDL-S file. But first I start by showing the ontology implementation both for WSMO and OWL-S. The WSDL-S use existing ontologies or defines the ontology in existing ontology languages and in this example I have used OWL ontology.
14.3. Ontology description for location services

The starting point of the OWL-S and the WSMO ontology development are both based on the WSDL file above. But the WSMO ontology is merged directly into the ontology written for the advanced attraction booking to add the location part of the PATS service. The OWL ontology is based entirely on the WSDL file.

All definitions and axioms in the ontologies are located in the ontology definition.

The ontologies start with a header. For OWL it looks like:

```xml
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
<!ENTITY owl "http://www.w3.org/2002/07/owl#" >
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
<!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>)
```

For the WSMO ontology it looks like:

```xml
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"
namespace { _"http://www.marekatt.net/master/WSMOLocationOntology#",
  wsml _"http://www.wsmo.org/wsml/wsml-syntax#",
  dc _"http://purl.org/dc/elements/1.1#" }
ontology Location
  rfp
dc#title hasValue "Location ontology"
dc#subject hasValue "location, coordinates, enduser"  
dc#description hasValue "Domain Ontology for location"
dc#contributor hasValue { "Marianne Rustad" }  
dc#format hasValue "text/plain"
dc#language hasValue "en-US"
endnfp
```

Both headers include namespaces for the ontology development. This is how the elements are uniquely identified in distributed environments. All elements described in the ontology have a set namespace, and it uses a default namespace for all elements not described as being a part of a different document.

To start the development of the ontologies the classes or concepts are defined. In this development I used the WSDL for the location service as the starting point and found which classes/concepts that needs to be added.

By example the “locationInfo” is in WSMO defined by:

```xml
class locationInfo
class nonFunctionalProperties
```

65
In OWL the same “locationInfo” is defined by:

```xml
<owl:Class rdf:ID="locationInfo">
  <rdfs:subClassOf rdf:resource="&owl;Thing"/>
  <rdfs:comment datatype="&xsd;string">The class of location information</rdfs:comment>
</owl:Class>
```

In WSMO we can define non functional parameters for all elements in OWL-S this is done by annotation elements.

To add sub concepts and sub classes to the ontology the WSMO sub concept is declared as:
```
concept coordinates subConceptOf locationInfo
  nonFunctionalProperties
    dc#description hasValue "The concept of coordinates"
  endNonFunctionalProperties
```

Whilst the OWL ontology defines its sub classes as:

```xml
<owl:Class rdf:ID="coordinates">
  <rdfs:subClassOf rdf:resource="#locationInfo"/>
  <rdfs:comment datatype="&xsd;string">The class of coordinates</rdfs:comment>
</owl:Class>
```

The classes/concepts need to be related to each other. This is done in WSML by relations and in OWL by properties. The use of sub classes/concepts includes that a property/relation for a class/concept also is true for its sub classes/concepts. In OWL all classes are sub classes of owl:Thing. It is also possible to define properties/relations in hierarchies as it is done for classes and concepts.

In WSMO this is denoted by:
```
relation hasAccuracy( ofType locationInfo, ofType accuracy)
  nonFunctionalProperties
    dc#description hasValue "Relation between the coordinates given and the accuracy of the coordinates"
  endNonFunctionalProperties
```

And in OWL it is written:
```
<owl:ObjectProperty rdf:ID="hasLocation">
  <rdfs:domain rdf:resource="#endUserIdentifier"/>
  <rdfs:range rdf:resource="#locationInfo"/>
  <owl:inverseOf rdf:resource="#locatesUser"/>
  <rdfs:comment datatype="&xsd;string">Property that indicates the user to have a location</rdfs:comment>
</owl:ObjectProperty>
```
In both languages we can include describe the properties/relations as transitive, symmetric, invers-of, and in both languages they include domain and range. A triplet is the basis for RDF and is described in figure 17:

![Figure 17 RDF triplet](image)

In the property /range description above, you can see the different way WSMO and OWL indicates domain and range. Domain corresponds to the subject of a statement, and range is the object of a statement. Two instances are related through the property relationship. Both initiatives has both object and data properties. By this you could either set range to another element or to a data value as string, integer etc.

Also instances or individuals are included in the definition. As an example I have added an instance of the end user.

For the WSML ontology the instance is written as:

```wsml
instance MariannesMobilePhone memberOf endUserIdentifer
   nfp
dc#description hasValue "Marianne's phone is an end user identifier"
   endnfp
   hasNumber hasValue "004740845356"
```

For OWL it is denoted:

```xml
<user rdf:ID="MariannesMobilePhone">
   <rdfs:comment rdf:datatype="&xsd;string">Mariannes phone is an end user identifier</rdfs:comment>
   <hasIdentefier rdf:datatype="&xsd;string">004740845356</hasIdentefier>
</user>
```

Both initiatives represent ontologies with the same basis. How this is done differs mainly in the logic used for syntax. The most important differences between the initiatives ontology modelling are:

- WSML uses the WSMO logic (epistemology) while OWL uses Description Logics (epistemology). This difference is shown in that WSML have local attribute definitions where as the properties in OWL are global even though there is functionality for setting local restrictions. WSML clearly separates the syntax and logical expression syntax, and OWL has one type of syntax for all elements.
- OWL uses Description Logic-style primitives through primitives like UnionOf, IntersectionOf and subClassOf for describing. WSML uses first-order style constants, predicates and logic connectives as and, implies and impliedBy.

### 14.3.1. Using editors for domain ontology development

To develop the domain ontology in both initiatives I used ontology editors. For OWL I used the Protégé editor for OWL, and for WSMO I used the Eclipse based WSMO studio. The user interface of the editors has its positives and negatives. The following shows an example of
screenshots of both editors. And the positive and negative form my user experience is listed beneath the attached screenshots. It should be noted that I have used the protégé editor in previous developments and have avoided some of the typical beginner problems of using new software.

For the WSMO Studio [61] the following screenshot shows the WSML ontology and how this is shown in WSMO studio, figure 18:

![Figure 18 WSMO studio screenshot](image)

WSMO studio has initially a good user interface. It is easy to understand the different elements and also importing ontologies are easy. Mostly it is easy to understand but there where some kinks using it. I had some problems making the ontology graphical visible, but was an easy task of click and drag. After some investigation I found that the ontology also could be edited and shown as text in the editor. This was a very useful for the development.
The biggest plus for this editor was the simplicity of right clicking to get the wanted functionality.

On the more negative side of the user experience of the WSMO studio editor, the biggest problem was actually saving the ontology. The resulting code often differed from the edited code in the text editor. This resulted in the practicality of graphical presentation of the concepts, relations and axioms. But the actual code was edited in notepad and checked for consistency by importing into the editor and checking it there.

To write the service description with goals, mediators and Web service the editor was quite useful. After much searching I found a tutorial in [60] which describes an example implementation of a Web service. This was useful for the development of the service file.

The example screenshot from Protégé shows the class with the properties view in figure 19:

![Protégé screenshot](image)

**Figure 19 Protégé screenshot**

Oppose to the development in WSMO Studio I developed the ontology entirely in the Protégé OWL editor. As mentioned I have used the tool before so it was familiar to me. In addition I
used the user guide, which was easily available from the help menu, and was a good reference for the development of the location ontology. I found the user interface easy to understand. It also included in oppose to the WSMO studio a graphical representation of adding elements to the ontology.

The biggest negative was the lack of ability to edit directly in the code. It is displayed by request but you can not edit it directly.

14.4. Services description for location services
As described in previous chapters OWL-S includes service, profile, process and grounding. The service is described in OWL-S as:

```xml
<service:Service rdf:ID="LocationService">
  <service:presents rdf:resource="#LocationProfile"/>
  <service:describedBy rdf:resource="#AtomicLocationProcess"/>
  <service:supports rdf:resource="#WsdlLocationGrounding"/>
</service:Service>
```

WSMO provides this through capability and interfaces. But WSMO distinguishes between the requester and provides. It is shown in the location service description as:

- capability LocationServiceCapability
- sharedVariables {?P}
- precondition
- ......
- effect
- ......
- interface LocationServiceInterface
- choreography LocationServiceChoreography

The service profile describes the use of a service both for the provider and the requestor in OWL-S as:

```xml
<profile:Profile rdf:ID="LocationProfile">
  <service:presentedBy rdf:resource="#LocationService"/>
</profile:Profile>
```

In WSMO we split this into goal and web service capability. The details of this are described in previous chapters.

The OWL-S process model represents how to interoperate with the service:

```xml
<process:AtomicProcess rdf:ID="AtomicLocationProcess">
  <service:describes rdf:resource="#LocationService"/>
  <process:hasInput rdf:resource="#endUserID"/>
  <process:hasOutput rdf:resource="#location"/>
  <process:hasOutput rdf:resource="#locationAccuracy"/>
</process:AtomicProcess>
```

The functionality of the OWL-S service model cooresponds with capabilities in the WSMO web service ontology. How to interact with the service cooresponds to WSMO.
choreographies. The functional description includes input, output, precondition and effect. It is the capabilities in WSMO that provide the functional description with the input, output, precondition and effect for the process.
As an example we have in OWL-S the input defined as:

```xml
<process:Input rdf:ID="endUserID">
    <process:parameterType rdf:datatype="... #endUserIdentefier"/>
</process:Input>
```

Chorography represents a state by a signature. It models changes in the state through transition rules. The choreography for the location service is:

```xml
choreography LocationServiceChoreography
    stateSignature
        in
            ssWSDL#wsdl.interfaceMessageReference(MobileTerminalLocationPortType/location/In)
        out
            ssWSDL#wsdl.interfaceMessageReference(MobileTerminalLocationPortType/location/Out)
    transitionRules
        forAll{?P} with (?P memberOf ssWSDL#endUserIdentefier) do
            add(?L memberOf ssWSDL#location and ssWSDL#hasLocation(?P,?L) and d=ssWSDL#hasAccuracy(?A,))
        endForall
```

OWL-S offers details of how to access the service through a grounding. A mapping between an abstract to a specific service is not currently defined in WSMO. In OWL this is written:

```xml
<grounding:WsdlGrounding rdf:ID="WsdlLocationGrounding">
    <service:supportedBy rdf:resource="#LocationService"/>
</grounding:WsdlGrounding>
```

For the WSDL-S extensions the following indicates the coupling to the OWL service file:

```xml

```

This defines the namespaces used in the WSDL-S file. It includes the OWL-S location service file. The namespaces are used to connect the WSDL elements to the classes of the ontology:

```xml
<complexType name="LocationInfo" wssem:modelReference="Ontology0#locationInfo">
    <sequence>
        <element name="longitude" type="xsd:float"
            wssem:modelReference="Ontology0#coordinates.hasLongitude"/>
        <element name="latitude" type="xsd:float"
            wssem:modelReference="Ontology0#coordinates.hasLatitude"/>
    </sequence>
```

```xml

```
The operations defined in the WSDL file are connected to the ontology through precondition and effect to indicate the class of the input and output:

```xml
<wsdl:operation name="getLocation">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="getLocationRequest">
        <wssem:precondition name="endUserIdentifier"
            wssem:modelReference="Ontology1#endUserIdentifier"/>
    </wsdl:input>
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://www.csapi.org/wsdl/parlayx/terminal_location" use="encoded"/>
</wsdl:input>
<wsdl:output name="getLocationResponse">
    <wssem:effect name="locationInfo"
        wssem:modelReference="Ontology1#locationInfo"/>
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://www.csapi.org/wsdl/parlayx/terminal_location/v1_0/service"
        use="encoded"/>
</wsdl:output>
```

The task of adding ontology definitions to WSDL files and extending them to WSDL-S files is quite simple. I have used the Eclipse Radiant plugin to define the WSDL-S. It has an easy to use interface and some user guidance. But the “tutorial” did not include much information and appeared to be constructed for an earlier version as it did not completely match the capabilities located in the editor. The solution was to use a syntax definition document from W3C to get the details of the language constructs. A screenshot of the user interface is shown in figure 20:

![Figure 20 Screenshot from the WSDL-S Radiant tool](image)
14.5. **Comparison of development through editors**

I have used both direct editing and graphical editors. As a part of this paper the syntax behind the document has been important. But for the inexperienced user a good graphical editing tool is imperative. The simplest is of course the WSDL-S if the ontology is already defined in another ontology language. And if ontology is to be built there is some functionality in the Radiant editor to build simple ontologies and this is where the protégé editor has its strength. It is easy to use and by following one of the user guides the process of modelling an ontology is not very difficult. The worst I used is the WSMO studio which might be ok if you have experience developing ontologies and services, but for the first time user it is not very good. The user interface is good but the functionality and intuitive use of the editor is not as good. But in WSMO Studios defence I have to say that I have used the Protégé editor before and had never used the WSMO Studio before and this might be reflected in my evaluation.

To fully evaluate the viability of Semantic Web Services it has to be cost efficient for industry to use it in there developments.

15. **Costs of semantics for Web Services**

For Semantic Web Services to be a profitable solution to Web service solutions there has to be a clear cost effective estimate of implementation efforts. To analyse and produce such an estimate I implemented the location ontology and service description for the location service to provide a time estimate. The result from this is used in this analysis [59].

The cost of implementation of semantics for a service is based on implementation work. It is based on the following steps:

- **Overview.** Creation of an overview of the services available for the user
- **Real world service identification** from already existing services and making them available as Web services.
- **Methodology establishment** for converting existing services to semantic services. Which includes:
  - a) Semantic service creation
    - Semantic description of service. It is assumed that the service is available as Web Services and the main effort here is to establish a semantic description.
    - Testing. The testing is assumed to be easier in a Semantic Web service, since one test can be used for all testing.
    - Registration at the service platform includes a similar effort to the conventional services.
  - b) Development of ontology
  - c) Development of end-user application interface (this is not included in the estimates as it is an element of further work)

The efforts are measured in a days work for one person (pd). The following compares semantic service provision to conventional service provisioning:

<table>
<thead>
<tr>
<th>Description</th>
<th>Provider A</th>
<th>Provider B</th>
<th>Conventional Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.) Semantic description</td>
<td>Serv. x 20</td>
<td>Serv. x 5</td>
<td>Next serv. 2</td>
</tr>
<tr>
<td>Registration</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Testing</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>
b.) Domain creation/update

<table>
<thead>
<tr>
<th>Service Type</th>
<th>30</th>
<th>3</th>
<th>4</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 1. Service provisioning for Semantic Web Services and conventional services

It is clear from the table that the ontology development is a huge cost which is not needed in conventional service development. We can also see that when the first semantics are developed for a service this may be reused for following services and the cost is decreasing. Through these observations one can create an estimation of person per day cost of Semantic Web Services development compared to conventional development. For five following service developments the observations is shown in figure 21:

Figure 21 Cost of service creation, registration, testing and ontology creation [59]

From figure 21 the conclusion is that the semantic service creation increases the costs for development, and the domain ontology development is a huge cost driver. For the second provider the ontology development is small since the ontology is already set and they do not have to develop the domain ontology from scratch, but will only have to add additional semantics needed for the service. With use of Semantic Web Services the cost for testing and upgrading of services has an advantage over the conventional service development.

Figure 22 shows on the left the break even point between conventional and Semantic Web Services development. In the left figure we see that the costs of one conventional and the cost for only one provider that the cost for adding semantics will always be higher than for the conventional. When the number of providers increases the picture is different. The cost for conventional services increase more rapidly than the cost for Semantic Web Services and the break even point is for 3 providers at 5 services and 5 providers at 3 services. If we include
the domain ontology development the costs increases and the break even point for 5 providers at 15 services as shown in the figure at the right.

![Graph showing break-even analysis for Semantic Web Services versus conventional service delivery](image)

**Figure 22 Break-even analysis for Semantic Web Services versus conventional service delivery [59]**

The conclusion made from this observation and analysis is that for Semantic Web Services when more services or providers come in to play, then the cost is smaller than for conventional services. And if the update of service is added the break even point would even be lower which emphasises that service upgrade and maintenance will give Semantic Web Services delivery an advantage over conventional service delivery.

### 16. Conclusion and further work

The rational for developing Semantic Web Services is based on the deficiencies in existing web technology. With the enormous growth of information and applications available on the Web there needs to be technology for adding machine understandable semantics. I have found that this is achievable through the Semantic Web and the Semantic Web Services. Through Semantic Web Services developments there is a basis for automasiation of finding, providing, composition and accessing services in an efficient way.

There are different initiatives for achieving this atomisation including WSMO, OWL-S and WSDL-S. To evaluate the initiatives I have found and defined different requirements of importance. Some of the results includes that the maturity of the initiatives is important for the deployment of Semantic Web service language. It will depend on maturity and standardisation and not just functionality. When a standard is set, there is a bigger chance for inclusion in industry developments. So being standardised is a positive for the possible deployment in industry, but this adds some rushing towards this goal and important issues might be omitted. By using the deficiencies of other initiatives as a base for new initiatives is a good idea, the problem might be the “competitive” aspect between the initiatives and the
unwillingness of the initiatives to take into consideration the problems pointed out by the other initiatives. But as a positive I have found it also to be something the initiatives after some time might evaluate as good input. As I see it the initiatives would gain at linking their efforts in a less competitive manner, and instead of criticising the other alternatives rather have a common ground for ideas etc.

To complete the study of Semantic Web Services a real world implementation of the initiatives is handled in part 3 as part of the location service from the traveller scenario. The experience of implementing this in the three initiatives was to so degree confusing as they all want to achieve the same thing but through different means. The development also brought some problems to the surface as the initiatives are not fully developed and some functionality was missing. By using the graphical interfaced editors the implementation work should not create too many problems. But my experience of using them raises the clear need for further development in this area.

Further work with Semantic Web Services includes getting the initiatives fully defined syntactically and functionality wise, and the different initiatives has to take in to consideration that they will have to cooperate in Web service environments. Also the need for better editors to hide complexity and also better user guides is eminent for deployment of Semantic Web Services in industry.
References


[31] DERI(?).WSML 2 Reasoner Framework [online]. Available at <URL: http://dev1.deri.at/wsml2reasoner/>


[45] Ankolekar A. (?).OWL-S' Relationship to Selected Other Technologies [online]. Available at <URL: http://www.ai.sri.com/daml/services/owl-s/1.2/related.html>

[46] Semwebcentral.(?). The OWL-S Editor [online]. Available at <URL:http://owlsem.editor.semwebcentral.org/>

[47] Semwebcentral. (?).The TUB OWL-S Matcher [online].Available at<URL:http://owlsem.projects.semwebcentral.org/>


APPENDIX A – WSML, OWL-S WSDL and WSDL-S files

1. WSDL for location

<?xml version="1.0" encoding="UTF-8"?>
xmlns:impl="http://www.csapi.org/wsdl/parlayx/terminal_location/v1_0/service"
xmlns:intf="http://schemas.xmlsoap.org/wsdl/soap/encoding"
xmlns:tns1="http://www.csapi.org/schema/parlayx/terminal_location/v1_0"
xmlns:tns2="http://www.csapi.org/schema/parlayx/common/v1_0"
xmlns:wsl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wslsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"><wsdl:types><schema

targetNamespace="http://www.csapi.org/schema/parlayx/terminal_location/v1_0"
xmlns="http://www.w3.org/2001/XMLSchema"><import

namespace="http://schemas.xmlsoap.org/soap/encoding/"/><simpleType

targetNamespace="http://www.csapi.org/schema/parlayx/common/v1_0"
xmlns="http://www.w3.org/2001/XMLSchema"><import

xmlns:xsd="http://www.w3.org/2001/XMLSchema"><complexType

name="LocationAccuracy"><restriction base="xsd:string"><enumeration value="Low"/><enumeration

targetNamespace="http://www.csapi.org/schema/parlayx/common/v1_0"
xmlns="http://www.w3.org/2001/XMLSchema"><import

xmlns:xsd="http://www.w3.org/2001/XMLSchema"><complexType

name="LocationInfo"><sequence><element name="longitude" type="xsd:float"/></sequence></complexType><complexType

name="EndUserIdentifier"><sequence><element name="value" type="xsd:anyURI"/></sequence></complexType></schema><schema

targetNamespace="http://www.csapi.org/schema/parlayx/terminal_location/v1_0"
xmlns="http://www.w3.org/2001/XMLSchema"><import

namespace="http://schemas.xmlsoap.org/soap/encoding/"/><complexType

name="LocationAccuracy"><restriction base="xsd:string"><enumeration value="Low"/><enumeration

targetNamespace="http://www.csapi.org/schema/parlayx/common/v1_0"
xmlns="http://www.w3.org/2001/XMLSchema"><import

namespace="http://schemas.xmlsoap.org/soap/encoding/"/><complexType

name="LocationInfo"><sequence><element name="longitude" type="xsd:float"/></sequence></complexType><complexType

name="EndUserIdentifier"><sequence><element name="value" type="xsd:anyURI"/></sequence></complexType></schema></wsdl:types>

<wsdl:message name="UnknownEndUserException">
<wsdl:part name="UnknownEndUserException" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="ServiceException">
<wsdl:part name="ServiceException" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="PolicyException">
<wsdl:part name="PolicyException" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="InvalidArgumentException">
<wsdl:part name="InvalidArgumentException" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="getLocationRequest">
<wsdl:part name="endUser" type="tns2:EndUserIdentifier"/>
<wsdl:part name="requester" type="tns2:EndUserIdentifier"/>
<wsdl:part name="accuracy" type="tns1:LocationAccuracy"/>
</wsdl:message>
<wsdl:message name="getLocationResponse">
<wsdl:part name="result" type="tns1:LocationInfo"/>
</wsdl:message>
<wsdl:message name="getInvalidArgumentRequest">
<wsdl:part name="result" type="tns1:LocationInfo"/>
</wsdl:message>
<wsdl:message name="getInvalidArgumentResponse">
<wsdl:part name="InvalidArgumentException" type="xsd:string"/>
</wsdl:message>
<wsdl:portType name="MobileTerminalLocationPort">
<wsdl:operation name="getLocation" parameterOrder="endUser requester accuracy">
<wsdl:input message="impl:getLocationRequest" name="getLocationRequest"/>
<wsdl:output message="impl:getLocationResponse" name="getLocationResponse"/>
<wsdl:fault message="impl:InvalidArgumentException" name="InvalidArgumentException"/>
<wsdl:fault message="impl:UnknownEndUserException" name="UnknownEndUserException"/>
<wsdl:fault message="impl:PolicyException" name="PolicyException"/>
<wsdl:fault message="impl:ServiceException" name="ServiceException"/>
</wsdl:operation>
</wsdl:portType>
</wsdl:definitions>
<wsdl:operation/>
<wsdl:portType>
<wsdl:binding name="MobileTerminalLocationPortSoapBinding" type="impl:MobileTerminalLocationPort">
<wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
<wsdlsoap:operation name="getLocation">
<wsdlsoap:input name="getLocationRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://www.csapi.org/wsdl/parlayx/terminal_location" use="encoded"/>
</wsdlsoap:input>
<wsdlsoap:output name="getLocationResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://www.csapi.org/wsdl/parlayx/terminal_location/v1_0/service" use="encoded"/>
</wsdlsoap:output>
</wsdlsoap:operation>
</wsdl:binding>
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
II. WSMO Domain Ontology

/**
 * Traveler Scenario - Location Domain Ontology - M12
 */

wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"
namespace (_"http://www.marekatt.net/master/WSMOLocationOntology#",
   wsml _"http://www.wsmo.org/wsml/wsml-syntax#",
   dc _"http://purl.org/dc/elements/1.1#" )

ontology Location
   nfp
dc#title hasValue "Location ontology"
dc#subject hasValue "location, coordinates, enduser"
dc#description hasValue "Domain Ontology for location"
dc#contributor hasValue { "Marianne Rustad" }
dc#format hasValue "text/plain"
dc#language hasValue "en-US"
endnfp

concept locationInfo
   nonFunctionalProperties
      dc#description hasValue "The concept of location information"
endNonFunctionalProperties

concept coordinates subConceptOf locationInfo
   nonFunctionalProperties
      dc#description hasValue "The concept of coordinates"
endNonFunctionalProperties
   haslongitude ofType _int
   haslatitude ofType _int

axiom ValidCoordinates
   definedBy
      forall (?co, ?x, ?y)
      ?co memberOf coordinates
      implies
      ?x[value hasValue _int] and
      ?y[value hasValue _int].

concept endUserIdentifier
   nonFunctionalProperties
      dc#description hasValue "The concept of id of end user"
endNonFunctionalProperties

concept user subConceptOf endUserIdentifier
   nonFunctionalProperties
      dc#description hasValue "The concept of the end user value"
endNonFunctionalProperties
   hasUserID OfType _string

concept accuracy
   nonFunctionalProperties
      dc#description hasValue "The concept of location accuracy"
endNonFunctionalProperties

axiom ValidLocationAccuracy
forall \( ?x \) ( ?x memberOf locationAccuracy implies
\( ?x[value hasValue "Low"] \) or
\( ?x[value hasValue "Medium"] \) or
\( ?x[value hasValue "High"] \) ).

class high subConceptOf locationAccuracy
  nonFunctionalProperties
  dc#description hasValue "The concept of high accuracy"
endNonFunctionalProperties
isValueFor inverseOf(hasAccuracy) impliesType locationAccuracy

class medium subConceptOf locationAccuracy
  nonFunctionalProperties
  dc#description hasValue "The concept of medium accuracy"
endNonFunctionalProperties
isValueFor inverseOf(hasAccuracy) impliesType locationAccuracy

class low subConceptOf locationAccuracy
  nonFunctionalProperties
  dc#description hasValue "The concept of low accuracy"
endNonFunctionalProperties
isValueFor inverseOf(hasAccuracy) impliesType locationAccuracy

relation hasAccuracy( ofType locationInfo, ofType accuracy)
  nonFunctionalProperties
  dc#description hasValue "Relation between the coordinates given and the accuracy of the coordinates "
endNonFunctionalProperties

relation indicatesAccuracy( impliesType accuracy, ofType locationInfo)
  nonFunctionalProperties
  dc#description hasValue "Relation that indicates the accuracy of location information "
endNonFunctionalProperties

relation hasLocation( ofType endUserIdentifier, ofType locationInfo)
  nonFunctionalProperties
  dc#description hasValue "Relation that indicates the user to have a location "
endNonFunctionalProperties

relation locatesUser( impliesType locationInfo, ofType endUserIdentifier)
  nonFunctionalProperties
  dc#description hasValue "Relation that indicates a location information to apply to an enduser "
endNonFunctionalProperties

instance MariannesMobilePhone memberOf endUserIdentifier
  nfp
  dc#description hasValue "Mariannes phone is an end user idnetifier "
endnfp
  hasNumber hasValue "004740845356"
III. WSMO Location Service file

/**
 * External location service - Service Specifications
 */

wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"

namespace {
    dO _"https://www.marekatt.net/master/WSMOLocationOntology#",
    dc _"http://purl.org/dc/elements/1.1#",
    ssWSDL _"http://129.241.219.154/parlayx/services#"}

/**
 * Please notice that I use the non-functionals:
 * dc#title to store the name of the service
 * dc#publisher to store the provider of the service
 **/

/*******************************************************************************/
Pats Lab
MobileTerminalLocation
Service
******************************************************************************/

webService _http://129.241.219.154/parlayx/services/MobileTerminalLocationPort

nfp
    dc#title hasValue "Location Service"
    dc#publisher hasValue "PatsLab"
endnfp

importsOntology _"https://www.marekatt.net/master/WSMOLocationOntology.wsml"
capability LocationServiceCapability
    sharedVariables {?P}
    precondition
        definedBy
            ?P memberOf dO#endUserIdentefier.
    effect
        definedBy
            ?L memberOf dO#locationInfo and
            dO#hasLocation(?P,?L) and
            ?A memberOf dO#hasAccuracy and
            dO#hasAccuracy (?A)

interface LocationServiceInterface
    choreography LocationServiceChoreography

stateSignature
    in
        dO#endUserIdentefier withGrounding
    out
        dO#location withGrounding

ssWSDL#wsdl.interfaceMessageReference(MobileTerminalLocationPortType/location/In)
transitionRules
    forAll {?P} with (?P memberOf dO#endUserIdentefier) do
        add(?L memberOf dO#location and dO#hasLocation(?P,?L) and dO#hasAccuracy(?A))
endForall
IV. OWL location ontology file

<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [ 
  <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>

<rdf:RDF xmlns="http://www.marekatt.net/master/OWLlocationOntology.owl#"
  xml:base="http://www.marekatt.net/master/OWLlocationOntology.owl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#">
  <owl:Ontology rdf:about="/"
  <owl:Class rdf:ID="accuracy">
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    <rdfs:comment rdf:datatype="&xsd;string">
The class of location accuracy
</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="coordinates">
    <rdfs:subClassOf rdf:resource="#locationInfo"/>
    <rdfs:comment rdf:datatype="&xsd;string">
The class of coordinates
</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="endUserIdentifier">
    <rdfs:comment rdf:datatype="&xsd;string">
The class of id of end user</rdfs:comment>
</owl:Class>

<owl:ObjectProperty rdf:ID="hasAccuracy">
    <rdfs:domain rdf:resource="#locationInfo"/>
    <rdfs:range rdf:resource="#locationInfo"/>
    <owl:inverseOf rdf:resource="#indicatesAccuracy"/>
    <rdfs:comment rdf:datatype="&xsd;string">
Property combining the coordinates given and the accuracy of the coordinates
</rdfs:comment>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="hasIdentifier">
    <rdfs:domain rdf:resource="#user"/>
</owl:DatatypeProperty>
</rdf:RDF>
<owl:DatatypeProperty rdf:ID="hasLatitude">
    <rdfs:domain rdf:resource="#coordinates"/>
    <rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="hasLocation">
    <rdfs:domain rdf:resource="#endUserIdentifier"/>
    <rdfs:range rdf:resource="#locationInfo"/>
    <owl:inverseOf rdf:resource="#locatesUser"/>
    <rdfs:comment rdf:datatype="&xsd;string" >Property that indicates the user to have a location</rdfs:comment>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="hasLongitude">
    <rdfs:domain rdf:resource="#coordinates"/>
    <rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>

<owl:Class rdf:ID="high">
    <rdfs:subClassOf rdf:resource="#accuracy"/>
    <rdfs:comment rdf:datatype="&xsd;string" >The class of high accuracy</rdfs:comment>
</owl:Class>

<owl:ObjectProperty rdf:ID="indicatesAccuracy">
    <rdfs:domain rdf:resource="#locationInfo"/>
    <rdfs:range rdf:resource="#locationInfo"/>
    <owl:inverseOf rdf:resource="#hasAccuracy"/>
    <rdfs:comment rdf:datatype="&xsd;string" >Property that indicates the accuracy of location information</rdfs:comment>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="locatesUser">
    <rdfs:domain rdf:resource="#locationInfo"/>
    <rdfs:range rdf:resource="#endUserIdentifier"/>
    <owl:inverseOf rdf:resource="#hasLocation"/>
    <rdfs:comment rdf:datatype="&xsd;string" >Property that indicates a location information to apply to an enduser</rdfs:comment>
</owl:ObjectProperty>

<owl:Class rdf:ID="locationInfo">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#locatesUser"/>
            <owl:allValuesFrom rdf:resource="#endUserIdentifier"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasAccuracy"/>
            <owl:allValuesFrom rdf:resource="#accuracy"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasLatitude"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasLongitude"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:comment rdf:datatype="&xsd;string" >The class of location information</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="low">
    <rdfs:subClassOf rdf:resource="#accuracy"/>
    <rdfs:comment rdf:datatype="&xsd;string" >The class of low accuracy</rdfs:comment>
</owl:Class>

<user rdf:ID="MariannesMobilePhone">
    <rdfs:comment rdf:datatype="&xsd;string" >Mariannes phone is an end user identity</rdfs:comment>
</user>
<hasIdentefier rdf:datatype="&xsd;string">004740845356</hasIdentefier>
</user>
<owl:Class rdf:ID="medium">
  <rdfs:subClassOf rdf:resource="#accuracy"/>
  <rdfs:comment rdf:datatype="&xsd;string">
    The class of medium accuracy</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="user">
  <rdfs:subClassOf rdf:resource="#endUserIdentifier"/>
  <rdfs:comment rdf:datatype="&xsd;string">
    The class of the end user value</rdfs:comment>
</owl:Class>
</rdf:RDF>
v. OWL-S location service file including the ontology

<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY swrl "http://www.w3.org/2003/11/swrl#" >
    <!ENTITY swrlb "http://www.w3.org/2003/11/swrlb#" >
    <!ENTITY dc "http://purl.org/dc/elements/1.1" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY daml "http://www.daml.org/2001/03/daml+owl#" >
    <!ENTITY time "http://www.isi.edu/~pan/damltime/time-entry.owl#" >
    <!ENTITY service "http://www.daml.org/services/owl-s/1.2/Service.owl#" >
    <!ENTITY profile "http://www.daml.org/services/owl-s/1.2/Profile.owl#" >
    <!ENTITY grounding "http://www.daml.org/services/owl-s/1.2/Grounding.owl#" >
    <!ENTITY p1 "http://www.example.org/owls/getLocation.owl#" >
    <!ENTITY time "http://www.isi.edu/~pan/damltime/time-entry.owl#" >
    <!ENTITY service "http://www.daml.org/services/owl-s/1.2/Service.owl#" >
    <!ENTITY profile "http://www.daml.org/services/owl-s/1.2/Profile.owl#" >
    <!ENTITY grounding "http://www.daml.org/services/owl-s/1.2/Grounding.owl#" >
    <!ENTITY p2 "http://www.marekatt.net/master/OWLlocationOntology.owl#" >
    <!ENTITY service "http://www.marekatt.net/master/OWLlocationOntology.owl#" >
    <!ENTITY list "http://www.daml.org/services/owl-s/1.2/generic/ObjectList.owl#" >
    <!ENTITY expr "http://www.daml.org/services/owl-s/1.2/generic/Expression.owl#" >
    ]>

<rdf:RDF xmlns="http://marekatt.net/LocationOntology.owl#"
    xmlns:list="http://www.daml.org/services/owl-s/1.2/generic/ObjectList.owl#"
    xmlns:time="http://www.isi.edu/~pan/damltime/time-entry.owl#"
    xmlns:p1="http://www.example.org/owls/getLocation.owl#"
    xmlns:time="http://www.isi.edu/~pan/damltime/time-entry.owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:p2="http://www.marekatt.net/master/OWLlocationOntology.owl#"
    xmlns:swrl="http://www.w3.org/2003/11/swrl#"
    xmlns:grounding="http://www.daml.org/services/owl-s/1.2/Grounding.owl#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:daml="http://www.daml.org/2001/03/daml+owl#"
    xmlns:swrl="http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <owl:Ontology rdf:about="" >
        <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.2/Profile.owl"/>
        <owl:imports rdf:resource="http://www.w3.org/2002/07/owl#"/>
        <owl:imports rdf:resource="http://www.w3.org/2003/11/swrl#"/>
        <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.2/Service.owl"/>
        <owl:imports rdf:resource="http://www.marekatt.net/master/OWLlocationOntology.owl#"/>

    </owl:Ontology>
    <owl:Class rdf:ID="accuracy">
        <rdfs:subClassOf>
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasAccuracy"/>
  <owl:allValuesFrom>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#high"/>
        <owl:Class rdf:about="#low"/>
        <owl:Class rdf:about="#medium"/>
      </owl:unionOf>
    </owl:Class>
  </owl:allValuesFrom>
</owl:Restriction>

<owl:ObjectProperty rdf:ID="applysTo">
  <rdfs:domain rdf:resource="#accuracy"/>
  <rdfs:range rdf:resource="#coordinate"/>
  <owl:inverseOf rdf:resource="#hasAccuracy"/>
</owl:ObjectProperty>

<process:AtomicProcess rdf:ID="AtomicLocationProcess">
  <service:describes rdf:resource="#LocationService"/>
  <process:hasInput rdf:resource="#endUserID"/>
  <process:hasOutput rdf:resource="#location"/>
  <process:hasOutput rdf:resource="#locationAccuracy"/>
</process:AtomicProcess>

<owl:Class rdf:ID="coordinate">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasPosition"/>
      <owl:allValuesFrom rdf:resource="#latitude"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="coordinate">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasPosition"/>
      <owl:allValuesFrom rdf:resource="#longitude"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="defines">
  <rdfs:domain rdf:resource="#accuracy"/>
  <rdfs:range rdf:resource="#coordinate"/>
  <owl:inverseOf rdf:resource="#hasValue"/>
</owl:ObjectProperty>

<process:Input rdf:ID="#endUserID">
  <process:parameterType rdf:datatype="&xsd;anyURI"
    >http://marekatt.net/LocationOntology.owl#endUserIdentifier</process:parameterType>
</process:Input>
<owl:Class rdf:ID="endUserIdentifier">
  <owl:disjointWith rdf:resource="#locationInfo"/>
  <owl:disjointWith rdf:resource="#accuracy"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="hasAccuracy">
  <rdfs:domain rdf:resource="#coordinate"/>
  <rdfs:range rdf:resource="#accuracy"/>
  <owl:inverseOf rdf:resource="#applysTo"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasLocation">
  <rdfs:domain rdf:resource="#locationInfo"/>
  <rdfs:range rdf:resource="#coordinate"/>
  <owl:inverseOf rdf:resource="#isInLocation"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasPosition">
  <rdfs:domain rdf:resource="#coordinate"/>
  <rdfs:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#latitude"/>
        <owl:Class rdf:about="#longitude"/>
      </owl:unionOf>
    </owl:Class>
    <owl:inverseOf rdf:resource="#isPartOf"/>
  </rdfs:range>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasValue">
  <rdfs:domain rdf:resource="#accuracy"/>
  <rdfs:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#high"/>
        <owl:Class rdf:about="#low"/>
        <owl:Class rdf:about="#medium"/>
      </owl:unionOf>
    </owl:Class>
    <owl:inverseOf rdf:resource="#defines"/>
  </rdfs:range>
</owl:ObjectProperty>

<owl:Class rdf:ID="high">
  <rdfs:subClassOf rdf:resource="#accuracy"/>
  <owl:disjointWith rdf:resource="#medium"/>
  <owl:disjointWith rdf:resource="#low"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="isInLocation">
  <rdfs:domain rdf:resource="#coordinate"/>
  <rdfs:range rdf:resource="#locationInfo"/>
  <owl:inverseOf rdf:resource="#hasLocation"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="isPartOf">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#latitude"/>
        <owl:Class rdf:about="#longitude"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="#coordinate"/>
  <owl:inverseOf rdf:resource="#hasPosition"/>
<owl:ObjectProperty>
  <owl:Class rdf:ID="latitude">
    <rdfs:subClassOf rdf:resource="#coordinate"/>
    <owl:disjointWith rdf:resource="#longitude"/>
  </owl:Class>
</owl:ObjectProperty>

<process:Output rdf:ID="location">
  <process:parameterType rdf:datatype="&xsd;anyURI">http://marekatt.net/LocationOntology.owl#locationInfo</process:parameterType>
</process:Output>

<process:Output rdf:ID="locationAccuracy">
  <process:parameterType rdf:datatype="&xsd;anyURI">http://marekatt.net/LocationOntology.owl#accuracy</process:parameterType>
</process:Output>

<owl:Class rdf:ID="locationInfo">
  <rdfs:subClassOf rdf:resource="&owl;Thing"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#applysTo"/>
      <owl:cardinality rdf:datatype="&xsd;int">2</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#endUserIdentifier"/>
  <owl:disjointWith rdf:resource="#accuracy"/>
</owl:Class>

<profile:Profile rdf:ID="LocationProfile">
  <service:presentedBy rdf:resource="#LocationService"/>
</profile:Profile>

<service:Service rdf:ID="LocationService">
  <service:presents rdf:resource="#LocationProfile"/>
  <service:describedBy rdf:resource="#AtomicLocationProcess"/>
  <service:supports rdf:resource="#WsdlLocationGrounding"/>
</service:Service>

<owl:Class rdf:ID="longitude">
  <rdfs:subClassOf rdf:resource="#coordinate"/>
  <owl:disjointWith rdf:resource="#latitude"/>
</owl:Class>

<owl:Class rdf:ID="low">
  <rdfs:subClassOf rdf:resource="#accuracy"/>
  <owl:disjointWith rdf:resource="#medium"/>
  <owl:disjointWith rdf:resource="#high"/>
</owl:Class>

<owl:Class rdf:ID="medium">
  <rdfs:subClassOf rdf:resource="#accuracy"/>
  <owl:disjointWith rdf:resource="#high"/>
  <owl:disjointWith rdf:resource="#low"/>
</owl:Class>

<grounding:WsdlGrounding rdf:ID="WsdlLocationGrounding">
  <service:supportedBy rdf:resource="#LocationService"/>
</grounding:WsdlGrounding>
</rdf:RDF>
VI. WSDL-S location service file

```xml
<wsdl:definitions targetNamespace="http://www.csapi.org/wsdl/parlayx/terminal_location/v1_0/service"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://www.csapi.org/wsdl/parlayx/terminal_location/v1_0/service"
    xmlns:intf="http://www.csapi.org/wsdl/parlayx/terminal_location/v1_0"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tns1="http://www.csapi.org/schema/parlayx/terminal_location/v1_0"
    xmlns:tns2="http://www.csapi.org/schema/parlayx/common/v1_0"
    xmlns:wdl="http://www.csapi.org/wsdl/parlayx/terminal_location/v1_0"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:ontology="http://www.marekatt.net/master/LocationOntology.owl"
    xmlns:wssem="http://www.ibm.com/xmlns/WebServices/WSSemantics"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:Ontology0="http://www.marekatt.net/master/OWLsLocationServiceOntologyWSDL_S.owl"
    xmlns:Ontology1="http://www.daml.org/services/owl-s/1.2/generic/ObjectList.owl#">
  <wsdl:types>
    <schema targetNamespace="http://www.csapi.org/schema/parlayx/terminal_location/v1_0"
        xmlns="http://www.w3.org/2001/XMLSchema">
      <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
      <simpleType name="LocationAccuracy"
          wssem:modelReference="Ontology0#accuracy">
        <restriction base="xsd:string">
          <enumeration value="Low" wssem:modelReference="Ontology0#low"/>
          <enumeration value="Medium" wssem:modelReference="Ontology0#medium"/>
          <enumeration value="High" wssem:modelReference="Ontology0#high"/>
        </restriction>
      </simpleType>
      <complexType name="LocationInfo" wssem:modelReference="Ontology0#locationInfo">
        <sequence>
          <element name="longitude" type="xsd:float"
              wssem:modelReference="Ontology0#coordinates.hasLongitude"/>
          <element name="latitude" type="xsd:float"
              wssem:modelReference="Ontology0#coordinates.hasLatitude"/>
          <element name="accuracy" nillable="true" type="tns1:LocationAccuracy"
              wssem:modelReference="Ontology0#accuracy"/>
          <element name="dateTime" nillable="true" type="xsd:dateTime"/>
        </sequence>
      </complexType>
    </schema>
    <schema targetNamespace="http://www.csapi.org/schema/parlayx/common/v1_0"
        xmlns="http://www.w3.org/2001/XMLSchema">
      <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
      <complexType name="EndUserIdentifier" wssem:modelReference="Ontology0#endUserIdentifier">
        <sequence>
          <element name="value" nillable="true" type="xsd:anyURI"
              wssem:modelReference="Ontology0#user"/>
        </sequence>
      </complexType>
    </schema>
  </wsdl:types>
  <wsdl:message name="UnknownEndUserException">
    <wsdl:part name="UnknownEndUserException" type="xsd:string"/>
  </wsdl:message>
</wsdl:definitions>
```
<wsdl:message name="ServiceException">
  <wsdl:part name="ServiceException" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="PolicyException">
  <wsdl:part name="PolicyException" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="getLocationRequest">
  <wsdl:part name="endUser" type="tns2:EndUserIdentifier"
wssem:modelReference="Ontology0#user"/>
  <wsdl:part name="requester" type="tns2:EndUserIdentifier"
wssem:modelReference="Ontology0#endUserIdentifier"/>
  <wsdl:part name="accuracy" type="tns1:LocationAccuracy"
wssem:modelReference="Ontology0#accuracy"/>
</wsdl:message>

<wsdl:message name="getLocationResponse">
  <wsdl:part name="result" type="tns1:LocationInfo"
wssem:modelReference="Ontology0#locationInfo"/>
</wsdl:message>

<wsdl:message name="InvalidArgumentException">
  <wsdl:part name="InvalidArgumentException" type="xsd:string"/>
</wsdl:message>

<wsdl:portType name="MobileTerminalLocationPort">
  <wsdl:operation name="getLocation" parameterOrder="endUser requester accuracy">
    <wsdl:input message="impl:getLocationRequest" name="getLocationRequest"/>
    <wsdl:output message="impl:getLocationResponse" name="getLocationResponse"/>
    <wsdl:fault message="impl:InvalidArgumentException" name="InvalidArgumentException"/>
    <wsdl:fault message="impl:UnknownEndUserException" name="UnknownEndUserException"/>
    <wsdl:fault message="impl:PolicyException" name="PolicyException"/>
    <wsdl:fault message="impl:ServiceException" name="ServiceException"/>
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="MobileTerminalLocationPortSoapBinding" type="impl:MobileTerminalLocationPort">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="getLocation">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="getLocationRequest"
wssem:precondition name="endUserIdentefier" wssem:modelReference="Ontology1#endUserIdentefier"/>
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://www.csapi.org/wsdl/parlayx/terminal_location" use="encoded"/>
  </wsdl:input>
  <wsdl:output name="getLocationResponse"
wssem:effect name="locationInfo" wssem:modelReference="Ontology1#locationInfo"/>
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://www.csapi.org/wsdl/parlayx/terminal_location/v1_0/service" use="encoded"/>
  </wsdl:output>
  <wsdl:output name="InvalidArgumentException">
    <wsdlsoap:output encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://www.csapi.org/wsdl/parlayx/terminal_location" use="encoded"/>
  </wsdl:output>
  <wsdl:output name="UnknownEndUserException">
    <wsdlsoap:output encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://www.csapi.org/wsdl/parlayx/terminal_location" use="encoded"/>
  </wsdl:output>
  <wsdl:output name="PolicyException">
    <wsdlsoap:output encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://www.csapi.org/wsdl/parlayx/terminal_location" use="encoded"/>
  </wsdl:output>
</wsdl:binding>
<wsdl:fault name="ServiceException">
    <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://www.csapi.org/wsdll/parlayx/terminal_location" use="encoded"/>
</wsdl:fault>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="MobileTerminalLocationService">
    <wsdl:port binding="impl:MobileTerminalLocationPortSoapBinding"
        name="MobileTerminalLocationPort">
        <wsdlsoap:address
        location="http://129.241.219.154/parlayx/services/MobileTerminalLocationPort"/>
    </wsdl:port>
</wsdl:service>
</wsdl:definitions>