Table of contents

- 1. Introduction
- 2. OWL-S tools
- 3. Web Service Execution Environment (WSMX)
- 4. Comparing OWL-S tools and WSMX
  - A. Service Discovery
  - B. Data Storage and Management
  - C. Mediation
  - D. Execution Management
  - E. Choreography and Orchestration
  - F. Programmatic access support
  - G. End user interaction support
  - H. Grounding
  - I. Reasoning support
  - J. Security issues
  - K. Fault tolerance support
- 5. Summary
- 6. Conclusions
- 8. References
- 9. Acknowledgements
1. Introduction

The application of semantics in Web Services as Semantic Web Services for dynamic discovery, composition, invocation and monitoring has been very helpful in enabling Enterprise Application Integration and E-Commerce. There are many initiatives that aim to realize the Semantic Web Services to enable effective exploitation of semantic annotations, and two major of them are Web Service Modeling Ontology (WSMO) and Ontology Web Language for Services (OWL-S). Several tools have been developed to realize both the conceptual models i.e. Web Services Execution Environment (WSMX) is the reference implementation for WSMO, on the other side OWL-S reference implementation exists in the form of loose collection of individual tools like OWL-S Editor, OWL-S Matchmaker, OWL-S Virtual Machine, OWL-S IDE, WSDL2OWL-S converter and OWL-S2UDDI converter etc. In this paper, we have conducted a comparison of both the reference implementations to identify similarities and differences between them and to evaluate their potential to become widely accepted implementation recommendations.

This paper presents a comparison between tools and technologies developed for two major initiatives to realize the vision of Semantic Web Services. These are, Web Services Modeling Ontology (WSMO) [Roman et al., 2004] based on Web Services Modeling Framework [Fensel et al., 2002] and Ontology Web Language for Services (OWL-S) [Martin et al., 2004] based on the DARPA Agent Markup Language program. For both, several tools and technologies have been developed i.e. Web Services Execution Environment (WSMX) [Cimpian et al., 2005] as reference implementation for WSMO whereas for OWL-S, several individual tools like OWL-S Editor, OWL-S Matchmaker, OWL-S Virtual Machine, OWL-S Integrated Development Environment, WSDL2OWL-S converter and OWL-S2UDDI converter for OWL-S.

The purpose of this comparison is to highlight the similarities and differences between the both references implementations for Semantic Web Services. The conducted comparison has been carried out with respect to implementations available for both, WSMX and OWL-S environment. Section 2 describes an overview of the Web Services Execution Environment (WSMX) as a reference implementation for Web Services Modeling Ontology (WSMO). In section 3, several individual OWL-S tools have been described. Section 4 carries out a detailed comparison of both the reference implementations from many different aspects i.e. Service Discovery, Reasoning, Fault tolerance support, Communication, Semantic Data Storage, Mediation, Negotiation and Orchestration, Service Composition, Execution Management, Security Issues, Programmers interaction support, End user interaction support and Groundings. Section 5 summarizes the whole comparison in the form of table and Section 6 draws the final conclusion.
2. OWL-S Tools

The OWL-S environment exists in the form of a loose collection of individual tools that focus on different specific aspects of its conceptual model. The set of tools which are recognized by the DARPA Agent Markup Language (DAML) consortium, are OWL-S Editor, OWL-S Matchmaker, OWL-S Virtual Machine, WSDL2OWL-S converter and OWL-S2UDDI converter.

The OWL-S Editor [Scicluna et al., 2004] enables development of semantic descriptions for Web-Service can easily and possibly hiding the complex constructs which this markup comprises.

The OWL-S matcher implements an algorithm that outputs different degrees of matching for individual elements of OWL-S descriptions. It considers elements of the service profile. With ranking a criterion is available to select a service among a large set of results. An ordered list of services provides a decision support to autonomously choose the best service possible. The implementation is provided as a Java tool with a Swing-based GUI, which allows to select a pair of OWL-S descriptions for requester and provider and compare the results.

The OWL-S Virtual Machine (OWL-S VM) [Paolucci et al., 2003] provides a general purpose Web service client for the invocation of a Web service based on the process model provided by OWL-S. The main functionality offered by the OWL-S VM is to control the interaction between Web services based on the process description described using OWL-S.

It is divided into two logical components. The first is the OWL-S Processor which uses an inference engine and set of rules to enforce the operational semantics of the OWL-S Process model and Grounding to enable the interaction with the Web service. This can be considered in terms of the OWL-S VM Processor determining and executing specific OWL-S processes based on the inputs supplied to the machine.

The second component is the invocation module which is responsible for making the actual invocations of operations on the Web service. Where the data schema for the specific ser-vice is specified in terms of XML Schema rather than OWL, a translation is required from the XML syntax of OWL to that required by the service, with the help of XSLT. Once the data to be sent to the message is syntactically correct, OWL-S VM uses the Apache Axis libraries to make the actual invocation.

WSDL2OWL-S [Paolucci et al., 2003] provides transition between WSDL and OWL-S. The results of this transformation are a complete specification of the Grounding and partial specification of the Process Model and Profile of OWL-S.
The incompleteness of the specification is due to the fact that OWL-S and WSDL differ in the information contained. WSDL does not provide any process composition information, therefore the result of the translation will also lack process composition information; furthermore, WSDL does not provide a service capability description, therefore the OWL-S Profile generated from WSDL is also necessarily sketchy and has to be completed manually. The output of WSDL2OWL-S converter provides the basic structure of OWL-S description of the Web Services and saves a great deal of man-power.

OWL-S2UDDI [Paolucci et al., 2003] converter converts the OWL-S profile descriptions into corresponding UDDI advertisements, which can then be published in a UDDI registry. This converter acts as a part of the OWL-S/UDDI registry [Paolucci et al., 2003]. The OWL-S/UDDI registry enhances UDDI registry with OWL-S matchmaking functionalities. Similarly several other related tools exist as implementation of specific aspects of OWL-S and it needs attention to integrate all these individual tools. OWL-S Integrated Development Environment (IDE) is an effort to make the individual OWL-S tools integrate and to work together as cohesive framework.

3. Web Service Execution Environment (WSMX)

The Web Service Execution Environment WSMX [Zaremba et al., 2005] is an execution environment for the dynamic discovery, selection, mediation, invocation and inter-operation of the Semantic Web Services providing a reference implementation for a service-oriented architecture that uses semantic annotation of all its major elements. Therefore a general architecture as well as necessary components has been defined and the interfaces and communication of components has been standardized. WSMX is a reference implementation for WSMO. The development process for WSMX includes defining its conceptual model (which is WSMO), standardizing the execution semantics for the environment, describing architecture and a software design and building a working implementation.

While WSMX is going to remain reference implementation of the Semantic Web Services systems, at this stage its major purpose is also to trigger standardization activities in OASIS and to connect many of the SWS European efforts to provide justification for and description of the Semantic Web Services infrastructure in general terms (on the conceptual level), rather than only focus on specific implementation. WSMX specification is currently further developed through the OASIS as Semantic Execution Environment (SEE). Figure 1 presents the WSMX architecture and its most important components.
WSMX is a useful framework for both Web Service providers and requesters. As a provider, one may register its service using WSMX in order to make it available to the consumers and, as a requester, one can find the Web Services that suits their needs and then invoke them in a transparent, secure and reliable way. WSMX itself is made available as a Web Service, so either for finding a Web Service or for actually invoking Web Services a requester has just to invoke WSMX itself. In the first case, a formal description of requester goal has to be provided, and in the second case, the actual data the requester wants to use for the invocation. In this way, WSMX can take care of all the other required computations such as heterogeneity reconciliation, composition, security or compensation.

Creating ontologies and semantic descriptions for Web Services is only useful if these descriptions can ultimately be applied. WSMX is an execution environment for finding and using Semantic Web Services that are described using WSMO. Considering current Web Service technologies there is a large amount of human effort required in the process of finding and using Web Services. Firstly the user must browse a repository of Web Services to find a service that meets their requirements. Once the Web Service has been found the user needs to understand the interface of the service, the inputs it requires and outputs it provides. Finally the user would write some code that can interact with the Web Service in order to use it. The aim of WSMX is to automate as much of this process as is possible. The user provides WSMX with a WSMO Goal that formally describes what they would like to achieve. WSMX then uses the Discovery component to find Web Services, which have semantic descriptions registered with WSMX that can fulfill this Goal. During the discovery process the users Goal and the Web Services description may use different ontologies. If this
occurs Data Mediation is needed to resolve heterogeneity issues. Data Mediation in WSMX is a semi-automatic process that requires a domain expert to create mappings between two ontologies that have an overlap in the domain that they describe. Once these mappings have been registered with WSMX the runtime data Mediation component can perform automatic mediation between the two ontologies. Once this mediation has occurred and a given service has been chosen that can fulfill the users Goal, WSMX can begin the process of invoking the service. Every Semantic Web Service has a specific choreography that describes they way in which the user should interact with it. This choreography de-scribes semantically the control and data flow of messages the Web Service can exchange. In cases where the choreography of the user and the choreography of the Web Service do not match process mediation is required. The Process Mediation component in WSMX is responsible for resolving mismatches between the Choreographies (often referred to as public processes) of the user and Web Service.

4. Comparing OWL-S tools and WSMX

In this section, both the reference implementations have been analyzed and compared from several different perspectives i.e. Service Discovery, Semantic Data Storage, Mediation, Execution Management, Choreography and Orchestration, Programmatic access support, End user interaction support, Groundings, Reasoning support, Security Issues, Fault tolerance support. Each of them is explained below in detail:

A. Service Discovery

WSMX supports discovery in terms of keyword based and semantics based [Kilgarriff 2005]. The keyword based discovery can be used to filter and quickly rank large amounts of goal and service descriptions to determine a subset of services, or goal on which to perform some semantic based discovery. The semantics based discovery has two major sections which are discovery based on concept i.e. and discovery based on rich semantic descriptions of services. Keyword based discovery and concept based semantic discovery is available currently available in WSMX however discovery based on rich semantic descriptions is still in process of implementation.

On the other hand, for discovery in OWL-S, there is OWL matchmaker which is itself a web service that helps make connections between service requesters and service providers. The Matchmaker allows users to find services by providing a mechanism for registering service capabilities. Registration information is stored as advertisements. When the Matchmaker service receives a query from a user, it searches its dynamic database of advertisements for agents that can fulfill the incoming request(s). Thus, the Matchmaker also serves as a liaison between a
service requester and a service provider. The matchmaking techniques are based on information retrieval, AI, and software engineering to compute both the syntactical and semantic similarity among service capability descriptions. The matching engine of the matchmaking system contains five different filters for namespace comparison, word frequency comparison, ontology similarity matching, ontology subsumption matching, and constraint matching. The user can configure these filters to achieve the desired tradeoff between performance and matching quality.

B. Data Storage and Management

The Resource Manager in WSMX provides a heterogeneous interface to its internal storage repositories. These repositories in WSMX are used to store the data, i.e. ontologies, goals, mediators and Web Services descriptions in the form of WSML. There are six kinds of repositories. Four of these repositories correspond to the top level concept of WSMO i.e. Web Services, ontologies, goals, and mediators. The fifth repository is for non-WSMO data items e.g. events and messages. Finally the sixth repository stores WSDL documents used to ground WSMO service descriptions to SOAP.

WSMX has also been adapted to use Triple Space Computing as its underline communication and coordination paradigm based on the principle of read and write of RDF triples. One of the benefits of integration of Triple Space Computing with WSMX will be to store the WSMX data in Triple Space [Martin-Recuerda et al., 2005]. The storage of WSMO top level entities on Triple Space helps in enhancing and fastening the process access of the data items afterwards. For instance, in the current discovery mechanism of WSMX, the WSML reasoners have to reason on each and every Web Service description available in the local repositories which takes significant amount of time. When the Web Services descriptions will be stored over Triple Space, the template matching based simpler reasoning will be used as a first step in order to filter-out the most relevant and possibly required Web Service descriptions. The filtered Web Services descriptions based on template based matching over Triple Space are retrieved and converted back to WSML to be used by WSML reasoners. It makes the process of resource access in WSMX simpler and faster by performing reasoning operations only on relevant Web Service descriptions rather than all. On the other hand, OWL-S based tools have not yet realized any framework for persistent storage.

C. Mediation

Due to the nature of Web the resources are developed in isolation. As such the heterogeneity and interoperability problems are imminent and a complete solution for Semantic Web and Semantic web Services demand a comprehensive support for mediation. In this respect, WSMO prescribes mediation as first class citizen and defines four types of mediators: ooMediators,
ggMediators, wgMediators and wwMediators [Lausen et al., 2005]. Though they are designed to serve different scopes they share the same common structure. It is important to note that WSMO mediators are actually description of a specific mediation capability that has to be applied to solve a particular mismatch. This capability is to be performed by the mediation service as depicted in Figure 1. Such a service is a concrete realization of a certain mediation technique, best suited to solve the given problem. We can say that in the WSMO and WSMX we have two dimensions for mediation: a description level (WSMO mediators) and the concrete implementation level (the mediation services), respectively.

From concrete mediation techniques and implementation we can identify three levels of mediation: data level mediation, process level mediation, and functional level mediation [Cimpian et al., 2005]. WSMX implements two of these levels (data and process level mediation) as distinct components, the third one (functional level mediation) being covered by discovery (and we refer to it as future work). The Data Mediation component in WSMX deals with heterogeneity problems that can appear at the data level. All messages in WSMX are semantically described in WSML, meaning that data to be mediated is described in terms of ontologies, i.e. data consists of ontology instances. The heterogeneity problems at the data level appear when the requester and the provider of a service use different ontologies to conceptualize their domain. As a consequence, data has to be transformed from terms of one ontology (e.g. requester’s ontology) into terms of the other ontology (e.g. provider’s ontology). Due to the fact that these transformations are taking place during run-time the whole process has to be completely automatic. The data mediator component in WSMX achieves this by relying on a set of mappings (semantic relationships) between the source and target ontology identified during design-time and stored in a persistent storage [Mocan & Cimpian 2005].

Process level mediation deals with solving the interaction mismatches. There could be cases where the requester and the provider of a Web Service understand the semantics of data but they have different requirements for the message exchange pattern to be followed. Essentially this means that one of them expects to receive/send messages in a particular order while the other one has a different messages or a different message order that doesn’t match. The role of the process mediator is to retain, postpone, rebuild or create messages that would allow the communication process to continue [Cimpian & Mocan 2005]. The third level of mediation, functional level mediation, is referring to the functional mismatches (capabilities mismatches) that may appear between the requestor and the provider of a service, two requestors or two providers. These mismatches can be represented as logical functions, expressing the functionalities differences. However, the work on the functional level mediation is still on the very beginning, not concrete proposal on what exactly these logical functions should contain or how they can be used being done yet.
Unlike WSMO and WSMX, OWL-S conceptual model and implementation do not consider mediators as first class citizens, OWL-S assuming the existence of external mechanisms that can perform similar tasks as the mediators, which could even be modeled using WSMO [Paolucci et al., 2004].

D. Execution Management

There are two aspects to execution management in WSMX as described in [Haselwanter 2005]. The first is conceptual and the second is specific to the implementation. From a conceptual perspective WSMX aims to be Service Oriented Architecture. To differentiate between Web services that can be invoked through WSMX and the services that from part of the architecture, we name the latter application services. Communication between the latter takes place through events. As application services can be distributed, the events mechanism must be able to handle this architectural style. Space-based messaging is a technique used by WSMX that abstracts messaging over a distributed system. Every software system has one or more execution (or operational) semantics which represent the control and data flow through the invocation order of its components. In many cases, the execution semantics is implicit and difficult to extract or configure for anyone not directly involved in code development. In the case of WSMX, multiple execution semantics can be defined for the system external to the actual implementation of the application services. The advantage to this is that the operation of WSMX can be configured in much the same way as a workflow engine can execute many independent workflow definitions.

The second aspect of this brief description of the execution management is in terms of the WSMX implementation. This is relevant only from the perspective of how existing standard technology can be applied to provide an execution environment for the emerging technology of Semantic Web services. The core of WSMX is realized as a Java Managed Extensions (JMX) microkernel. This provides the basis for managing the deployment and run-time of the application services on WSMX and the basis for the event management. The event-based messaging is implemented using Java-Space technology which allows the collection of WSMX application services to appear as if all on one virtual machine. The management information made possible through JMX is accessible through a Web application supported by a built-in HTTP adapter. The execution semantics are implemented as a special Java class that is passed between component wrappers like a token. This class maintains the state of that instance of the execution semantics. Work is underway to replace this implementation with one based on the abstract state formalism used by the WSMX choreography and composition application services.

The execution management for the OWL-S VM is more difficult to analyze as quite limited information is published on the software design of the system [Stollberg et al., 2006]. This is understandable as the implementation was aimed as a proof-of-concept client for OWL-S process descriptions. However, binary
files and some test cases have been published. On the basis of this code and the documentation in [Cimpian et al., 2005], the OWL-S VM is designed as a straight-forward Java application. The purpose of the implementation did not call for any complexity in terms of SOA or event-based messaging. Consequently the execution semantics are implicit in the implementation code itself. Although this limits the flexibility of extending the system in the future, this was likely not the intention of the authors.

E. Choreography and Orchestration

There exists a conceptual model for describing choreography and orchestration interfaces in WSMO. The state-based mechanism for describing WSMO choreography and orchestration interfaces is based on the Abstract State Machine methodology where an ASM is used to abstractly describe the behavior of the service with respect to an invocation instance of a service. Choreography engine in WSMX further distinct between provider and requester choreographies similar to the distinction of provided and required interface behavior. The distinction is based on the initial communication task the user is required to use. A provider choreography can only use communication task types where a receive event occurs first. A reply task is not necessarily mandatory, but desirable in most cases. A requester-choreography can only use communication task types where an event always occurs first, after which optional receive events may occur. The idea is that a requester-choreography works as the counterpart of the provider-choreography and that therefore the send event corresponds to the request event and vice versa. Only if the requester and provider choreography are perfectly symmetric a direct interaction is possible. Moreover, Choreography Engine deals with binary collaborations only. In order to define n-ary collaborations, orchestration interface is still needs to be defined.

In OWL-S community, no choreography exists both at conceptual and implementation level till yet. OWL-S conceptual model defines only orchestration. However, OWL-DL (description logic) and WS-CDL (Web Services Choreography Description Language) have been considered as OWL-S choreography and orchestration to bring automated service composition, coordination, and cooperation. No implementations have been found in this regard.

F. Programmatic access support

Programmatic access support enables read, write, update, validate and execute the specifications. WSMO4J [de Bruijn et al., 2005] is an application programming interface for Web Services Modeling Ontology (WSMO, v.1.0), which allow for basic manipulation of WSMO descriptions, e.g. creation, exploration, storage, retrieval, parsing, and serialization. wsmo4j is another reference implementation of the WSMO API, including a WSML parser and a file-system-based data store.
OWL-S API [OWL-S API] provides a java API for programmatic access to read, execute and write OWL-S service descriptions. It supports different versions of OWL-S (OWL-S 1.0, OWL-S 0.9, DAML-S 0.7) descriptions. The API provides an ExecutionEngine that can invoke AtomicProcesses that has WSDL or UPnP groundings, and CompositeProcesses that uses control constructs Sequence, Unordered, and Split. Executing processes that relies on conditionals such as If-Then-else and RepeatUntil is not supported in the default implementation. But this implementation can be extended to handle these constructs if the application that uses the OWL-S descriptions has a custom syntax and evaluation procedure for the conditions. The API also provides some limited support for the generic OWL manipulation. All the objects are derived from OWLResource class which has accessor functions to get the value of any OWL property. It is also possible to access the underlying data model for more advanced queries. But the underlying data model is not part of this API and may be changed to another library (OWL API being the most probable candidate) so it is recommended not to rely on this feature. API does not have support for the Conditional aspects defined in OWL-S conceptual model. Executing processes that relies on conditionals such as If-Then-else and RepeatUntil are also not supported by the API till yet.

G. End user interaction support

WSMT (Web Service Modeling Toolkit) is a framework for the rapid deployment of graphical administrative tools which can be used with WSMO, WSML and WSMX. WSMT provides the functionality to deploy other tools as plug-ins to a larger system. Currently WSMT contains WSMO Ontology Visualizer tool, Data Mediation Mapping tool and a Monitoring tool. WSMT is based on WSMO API and uses WSMO4j as its reference implementation. The WSML Editor is a tool for the creating and publishing WSML documents, that is implemented as a plug-in to the Web Services Modeling Toolkit (WSMT) providing a graphical mechanism for the creation of logical expressions and providing visualizations of ontologies using directed graphs. There are some other tools that exists independent to WSMT as well i.e. WSMO studio which is a Semantic Web Service editor compliant with the Web Service Modeling Ontology. It provides Editor for WSMO elements (ontologies, services, goals, mediators), Import/export from WSML, Choreography designer, for WSMO centric choreographies, Front-end for ontology / service / goal repositories, WSML text editor with syntax coloring, Integrated WSML Validator.

OWL-S editor tool provides a tool that will help the un-experienced user and/or programmer to create OWL-S descriptions for a Web Service in a short time. The tool is divided into three main parts i.e. Creator, Validator, Visualiser. The creator enables to create an empty OWL-S description either from a template or through a wizard called "OwlsWiz" which accepts an input WSDL file and extracts partial information from it to create a basic OWL-S description. The OwlsWiz provides the user with the tools needed to create an OWL-S description in the least
amount of time possible and without exposing the user to the (at times) complex
OWL-S structures. The validator part provides for validating of the URIs used in the
OWL-S descriptions and also validate the syntax of the ontologies. The
Visualiser part enables the user to visualise the descriptions and service
compositions in a graphical manner by exploiting UML activity diagrams. The
OWL-S Protégé based Editor tool provides a comprehensive set of capabilities
for creating and maintaining OWL-S service descriptions, with a user-friendly
style of interaction that is organized around the conceptual structure of OWL-S.
The Web Service composer tool helps in dynamic composition of web services
by a semi-automatic process including presenting matching services to the user
at each step of a composition, filtering the possibilities by using semantic
descriptions of the services. The generated composition is then directly
executable through the WSDL grounding of the services.

H. Grounding

Grounding is essential aspect inorder to represent semantic descriptions in
WSDL which is current industry standard for defining how messages can be
exchanged between services over the Internet. WSMO service descriptions can
be grounded to WSDL [Kopecký et al., 2005]. The data model of the input and
output messages for WSDL services is defined using XML Schema while the
data model for a WSMO service is defined using the conceptual model provided
by WSMO ontologies. This leads to the requirement to map between the
ontological data in the state machine and its representation as XML messages.
Moreover, it also needs to specify how and when messages to and from the
service are generated and sent. WSMO choreography [Scicluna et al., 2005]
only
says that the client can read the data but in fact it is the responsibility of the
service to send the data to the client in the form of a message. The grounding
must also provide the necessary serialization and networking details, i.e. what
underlying protocol (e.g. SOAP, HTTP) should be used for passing the
messages, how the XML data is encapsulated in the underlying protocol, and
where exactly the data should be sent. No implementation of grounding support
is available for WSMX so far [Moran et al., 2004].

Grounding in OWL-S is also specified in terms of OWL-S/WSDL mapping [Martin
et al., 2004]. It involves a complementary use of the OWL-S and WSDL, in a way
that is in accord with the intentions of the authors of WSDL. Both are required for
the full specification of grounding, because they do not cover the same
conceptual space. However, they do overlap in the area of providing for the
specification of what WSDL calls `abstract types", which in turn are used to
characterize the inputs and outputs of services. WSDL, by default, specifies
abstract types using XML Schema, whereas OWL-S allows for the definition of
abstract types as (description logic-based) OWL classes. However, WSDL/XSD
is unable to express the semantics of an OWL class. Similarly, OWL-S has no
means, as currently defined, to express the binding information that WSDL
captures. Thus, it is natural that a OWL-S/WSDL grounding uses OWL classes
as the abstract types of message parts declared in WSDL, and then relies on
WSDL binding constructs to specify the formatting of the messages. WSDL2OWL-S converter tool [Paolucci et al., 2003] provides transition between
WSDL and OWL-S. The result of this transformation is a complete specification
of the Grounding and partial specification of the Process Model and Profile.
Moreover, OWL-S2UDDI converter tool [Paolucci et al., 2003] converts the OWL-
S profile descriptions into corresponding UDDI advertisements, which can then
be published in a UDDI registry.

I. Reasoning support

The reasoner in WSMX provides reasoning services for the mapping process of
mediation, validation of a possible composition of services, determination if
composed services in process are executable in a given context. It allows
exploiting information represented in the formal model of the domain of discourse.
There are several tools exists that can support reasoning in WSMX like
wsdl2reasoner [WSML reasoners, 2005] which is a reasoning API and
implementation of a mapping from wsml to a vendor-neutral rule representation
that can be mapped to various different rule engines. It supports Query
Answering for WSML-Core, Query Answering for WSML-Flight, Built-in
Predicates. However complex Data Types are not supported till yet. MINS
[WSML reasoners, 2005] is another reasoner which is basically a rule engine
tailored for WSML which essentially is or rather will be a reimplementation of the
SILRI rule system. WSML-DL-Reasoner [WSML reasoners, 2005] consists of a
wrapper of WSML-DL expressions to a classical Description Logics syntax. As
the core reasoner FaCT++ is used. The Description Logics expressions are sent
to the reasoner via its DIG interface and supports different T-Box reasoning tasks
which are provided by FaCT++, Querying for all concepts, Querying for the
equivalents, for the children, for the descendants, for the parents and for all
ancestors of a given concept, subsumption test of two concepts with respect to
the knowledge base and wrapper of WSML-DL to the XML syntax of DL used in
the DIG interface.

Pellet [Sirin et al., 2004] is an OWL DL reasoner based on the tableaux
algorithms developed for expressive Description Logics. It supports the full
expressivity OWL DL including reasoning about nominals (enumerated classes).
Therefore, OWL constructs owl:oneOf and owl:hasValue can be used freely.
Currently, Pellet ensures soundness and completeness by incorporating the
recently developed decision procedure for SHOIQ (the expressivity of OWL-DL
plus qualified cardinality restrictions in DL terminology). The Protege-OWL
Reasoning API [Martin et al., 2004] helps explore and edit OWL ontologies,
Protege-OWL features a reasoning API, which can be used to access an external
DIG compliant reasoner, thereby enabling inferences to be made about classes
and individuals in ontology.

J. Security issues
Security issues have not been handled in both the reference implementations. However, at conceptual model level, WSMO plans to represent it as ability of a Web service to provide authentication, authorization, confidentiality, traceability/auditability, data encryption and non-repudiation [de Bruijn et al., 2005] whereas nothing about security has been planned for OWL-S till yet.

K. Fault tolerance support

Fault tolerance refers to ability of the system to respond gracefully to an unexpected failure. There are many levels of fault tolerance, the lowest being the ability to continue operation in the event of a power failure.

WSMX supports fault tolerance to some extent by having well-defined interfaces of its components to interact with each other which would lead it to allow duplication/mirroring of critical services deployed as backup on different systems so that the backup could take services could takeover incase the original one fails. The OWL-S environment is in very early stages [OWL-S] and no fault tolerance support has been planned for it so far.

5. Summary

The results of our comparison show that OWL-S conceptual model is a subset of WSMO. WSMO is based on defining Ontologies, semantic descriptions of Web Service, Mediators and user requirements in terms Goals, whereas OWL-S conceptual model just focuses on providing a semantic description of Web Services using OWL ontologies. Same is reflected in the comparison of reference implementations of both that WSMX provides a more comprehensive implementation support for many different aspects of Semantic Web Services as it follows a consolidated approach and categorizes different components in different layers. OWL-S environment is based on a loose collection of several individual developed tools which focus on specific aspects of Semantic Web Services. This loose collection of OWL-S tools also requires a separate effort to integrate and make them work together. Table 1 below provides a summary of the whole comparison.

<table>
<thead>
<tr>
<th>Table 1: Comparison Chart</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Comparison</strong></td>
</tr>
<tr>
<td>Purpose</td>
</tr>
<tr>
<td>Overall Architecture</td>
</tr>
<tr>
<td>Comparison</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td>Service Discovery</td>
</tr>
<tr>
<td>Data Storage</td>
</tr>
<tr>
<td>Mediation</td>
</tr>
<tr>
<td>Execution Management</td>
</tr>
<tr>
<td>Comparison</td>
</tr>
<tr>
<td>----------------------------------</td>
</tr>
<tr>
<td>Choreography and Orchestration</td>
</tr>
<tr>
<td>Programmatic Access Support</td>
</tr>
<tr>
<td>End User Interaction Support</td>
</tr>
<tr>
<td>Grounding</td>
</tr>
<tr>
<td>Comparison</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td>Reasoning</td>
</tr>
<tr>
<td>Security</td>
</tr>
<tr>
<td>Fault Tolerance</td>
</tr>
</tbody>
</table>

### 6. Conclusions and Future Work

In this document we carried out a comparison of reference implementations of two most commonly known Semantic Web Services conceptual models i.e. WSMO and OWL-S. WSMX is the reference implementation of WSMO and OWL-S environment based on different individual tools acts as reference implementation for OWL-S conceptual model. WSMO is more comprehensive than that of OWL-S and covers most of the aspects of Semantic Web Services in terms of defining Web Services descriptions, Ontologies, User goals and mediation whereas OWL-S focuses only on semantic description of services. Same is inherited by reference implementations for both. WSMX exists as close to complete reference implementation Semantic Service Oriented Architecture, specifically WSMO. It follows a comprehensive and consolidated approach having well-defined components categorized in different layers and support for coordination of components in whole framework. The OWL-S not only lacks features in its conceptual model but also in its reference implementation due to loose collection of several independent tools which still require another effort to make them work together.
The future work includes adding METEOR-S [Patil et al., 2004] in this comparison and to keep the overall comparison updated with respect to the developments going on in all execution environments.

References


[Scicluna et al., 2005] J. Scicluna, A. Polleres, D. Roman, D. Fensel: *Ontology-based Choreography and Orchestration of WSMO Services*, WSMO working draft D14v0.3. Available at [http://www.wsmo.org/TR/d14/v0.3](http://www.wsmo.org/TR/d14/v0.3).


**Acknowledgement**

The work is funded by the European Commission under the projects DIP, Knowledge Web, InfraWebs, SEKT, SWWS, ASG and Esperonto; by Science Foundation Ireland under the DERI-Lion project; by the FIT-IT (Forschung, Innovation, Technologie - Informationstechnologie) under the projects RW² and TSC.

The editors would like to thank to all the members of the WSMO working group for their advice and input to this document.