Semantic Web Services Languages and Technologies: Comparison and Discussion

Rohit Aggarwal
LSDIS Lab, University of Georgia, Athens, 30602
aggarwal@cs.uga.edu

1 Introduction

Web services were primarily designed to be loosely coupled and to provide inter-operability between business applications. The large amount of human interaction, that is required for integration of multiple applications by the current Web service technologies, needs to be reduced. To make integration automated we need machine processable description of the data and functions of the involved entities. Semantic Web technologies use description logic based languages to annotate the content with ontology concepts to make it machine understandable. Due to the increasing interest in knowledge than data, and the rising popularity of the Semantic Web and Web Services, there have been significant interests in developing technologies that support Semantic Web Services. The Semantic Web industry is experiencing a need for identifying and developing technology that will provide a firm and long-term foundation for the future of Web services on the internet. This foundation is required to support the universal approaches to service deployment and use. It should support the currently technically feasible approaches; and should have the features of flexibility, extensibility, and consistency with the vision of the Semantic Web.

A variety of tools and modeling frameworks that support publication, discovery and composition of Semantic Web Services have been developed in the near past. These initiatives include OWL-S [Martin et al., 2003] (formerly DAML-S [DAML-S]), METEOR-S [METEOR-S, 2004], and WSMO [Roman et al., 2004] etc. In general however we feel that no tool/framework provides all that is required for a generic Web services modeling platform for the Semantic Web. These standards are still incomplete and may not fulfill the future demands of the industry like complexity, scalability, dependability, to name a few. The SWSI Language Committee [SWSL] is working on identifying and developing a computer language technology which will standardize the semantic information about web services and develop a language for its declarative specification.

Also, the semantic information about web services needs to be general enough to be able to support nontrivial mechanized interactions between Web services and intelligent software agents. Ideally, the Semantic Web services language should enable:

- dynamism in all aspects of Web service use, such as selection, discovery, composition, invocation, negotiation, and recovery from failure;
- be extensible and should be closely integrated with knowledge resources on the Semantic Web.

The aim of this paper is to survey the current semantic Web services languages and modeling frameworks by outlining their features and capabilities. We will then compare the approaches and identify the deficient features which need to be overcome to meet the requirements of the industry and the SWSL in developing a formal language/technology for supporting semantic Web services.

The rest of the paper of organized as follows: Section 2 details the current technologies in this area. Section 3 presents the overall discussions and comparisons. Section 4 discusses the current best approach with METEOR-S, and finally Section 5 concludes the paper.

2 Current Approaches

Although there have been a number of initiatives and straw proposals [SWSL Straw Proposals] in this area like “SCLP Rules + Ontologies to Describe E-Services” [Grosof, 2004], “FLOWS (First-order Logic Ontology for Web Services)” [Berarde et al., 2004], OWL-S, The Enchilada Proposal [Kifer, 2004], WSMO, METEOR-S etc., we will, in this section, detail the three most significant of these initiatives.

2.1 Enhanced OWL-S (using SWRL)

As part of the DARPA Agent Markup Language program, OWL-S is being developed as an ontology of services, that will help users and software agents to discover, compose, invoke, and monitor Web services. The structure of OWL-S can be divided into three main parts:

- service *profile* for publishing and discovering services;
- *process model*, which describes a service's operation;
- *grounding*, which defines interoperability with a service, via messages.

The Service Profile describes the services offered by the service providers and the services that are required by the service requesters. An OWL-S Profile represents three basic types of information: the organization that provides the service, what does the service compute or provide, and other characteristics of the service.

The process model represents a process. A process can be abstractly viewed as something that transforms data from a set of inputs to outputs. A process (and the profile) is represented with IOPE’s (Inputs, Outputs, Preconditions and Effects). There are three types of processes: atomic, simple and composite. Atomic processes are single step and can be directly invoked. Simple processes cannot be directly invoked but represent elements of abstraction. Composite processes use control constructs and can be a composition of multiple composite or non-composite processes.

The grounding of a service specifies the details of the protocol and message formats supported by the service, if it can be serialized, which transport it uses, and about it’s addressing. Grounding provides mapping from the abstract specifications to a concrete specification of primarily inputs and outputs of the atomic processes. It specifies the description elements which are necessary for interacting with the service. The main aim of grounding is to concretely realize the inputs and outputs of an atomic process as messages. These messages further carry the inputs and outputs in a specific defined communicable form. Web Services Description Language (WSDL) is the current industry standard for describing web service. It is used to describe a web service in a way that hides the implementation details but defines the inputs and outputs of a web service, the data-types of the inputs/outputs, the binding information (the communication protocol the web service supports and what messages can be sent in and out of the web service) and other details about the web service. The WSDL’s concept of binding is very similar to the grounding in OWL-S.

OWL Web Ontology Language is a recommendation of the W3C. Instead of presenting information to humans, OWL is designed to provide machine processable information content. Although XML, RDF, and RDF Schema are also machine processable, OWL provides additional vocabulary along with formal semantics to assist in better machine interpretability. OWL is the Web Ontology Language used to write ontologies. It has three different flavors. OWL- Lite, OWL DL and OWL Full. OWL Lite ontology is a subset of OWL DL ontology while OWL DL ontology is a subset of OWL Full ontology. OWL DL supports those users who want the maximum expressiveness while retaining computational completeness. The different flavors were provided with different restrictions to suit different needs. The combination of OWL DL and OWL Lite with Unary/Binary Datalog RuleML sublanguages of Rule Markup Language [Boley et al., 2004] make up SWRL [Horrocks et al., 2004]. OWL-DL and OWL Lite are sublanguages of the OWL Web Ontology Language [McGuinness et al., 2004]. OWL-Lite supports cardinality constraints but it only allows values of 0 or 1 which makes it not very useful. If the cardinality constraints problem in OWL-Lite is fixed [OWL-Lite-] and the not very useful equality statement in OWL-Lite is eliminated, this subset of OWL-Lite can then be extended in a natural manner into the course of a rule language. SWRL provides Horn-like rules for both OWL DL and OWL Lite. It includes a high-level syntax for representing these rules. To provide a formal meaning for OWL ontologies which do not include rules written in the syntax provided by SWRL, a model-theoretic semantics is given. Model-theoretic semantics provide generalization of data models (relational or semi-structured) so that can deal with uncertain/vague information and identifiable objects. Being a combination of OWL and Horn Logic, SWRL can be used to define rules e.g. business logic, where a supplier can set different discounts for different clients etc. The rules can be written in a file which can be associated with the web service or a process. To check if a rule is satisfied (or if it fires) we need SWRL reasoners. A future enhanced version of OWL-S will make use of SWRL rules for inference.

SWRL extends OWL DL abstract syntax by a further axiom:

```
“axiom ::= rule  where:
rule       ::=      'Implies(' [ URIreference ] { annotation } antecedent consequent ')'

antecedent  ::=     'Antecedent(' { atom } ')'
consequent ::=      'Consequent(' { atom } ')'

atom       ::=      description '(' i-object ')'
                    | dataRange '(' d-object i-object ')'
                    | individualvaluedPropertyID '(' i-object i-object ')'
                    | datavaluedPropertyID '(' i-object d-object ')'
                    | sameAs '(' i-object i-object ')'
                    | differentFrom '(' i-object i-object ')'
                    | builtIn '(' builtIn builtinID { d-object } ')'

builtinID ::= URIreference
```

A rule is satisfied by an interpretation iff every binding that satisfies the antecedent also satisfies the consequent. The semantic conditions relating to axioms and ontologies are unchanged, e.g., an interpretation satisfies an ontology iff it satisfies every axiom (including rules) and fact in the ontology.” [Horrocks et al., 2004]

A rule is a combination of an antecedent and a consequent. A rule typically claims that if the antecedent is true then the consequent has to be true. An antecedent/consequent is an assertion e.g. parent(x,y) which means x is a parent of y.
Now, Implies(Antecedent(parent(x,y)), Consequent(older(x,y))) means, if x is a parent of y then x is older than y.

Let us try to understand this by asserting that the combination of the hasSon and hasSister properties implies the hasDaugher property.

In SWRL the rule would be written like:

```
Implies(Antecedent(hasSon(I-variable(x1) I-variable(x2))
                hasSister(I-variable(x2) I-variable(x3)))
    Consequent(hasDaugher(I-variable(x1) I-variable(x3))))
```

From this rule, if x1 has x2 as a son and x2 has x3 as a sister then x1 has x3 as a daughter.

In OWL, a new enterprise cannot be defined as a type of company (existing concept in the ontology) with more than 5000 employees (another concept in the ontology). This is because OWL does not support comparison operators and other complex conversions. Such rules which cannot be expressed in normal OWL-S can be expressed using SWRL. A Rule Language can become an important component of Web processes. Take the example of a supply chain, wherein different vendors may be given different discounts or depending on the price the client is willing to pay; different vendors can be chosen to deliver the consignment. Such processes may require complex rules which OWL cannot provide.

My view:
Description Logics (DLs) are widely used to provide reasoning for OWL. OWL-S which is based on OWL uses DL reasoners to check the logical consistency of classes in the ontology. However sometimes it leads to some unexpected inferencing which is difficult to explain. [Schlobach et al., 2003][Borgida et al., 2000]. Although OWL is gaining an industry wide acceptance, it is not expressive enough for all applications. SWRL (OWL in conjunction with RuleML) is being used to overcome this difficulty. SWRL is based on the ORL proposal [Horrocks et al., 2003] and is the subject of ongoing development by the DAML joint committee. The enhanced OWL-S will use SWRL to achieve more reasoning than can be achieved with DL reasoners. Reasoning in SWRL is undecidable in the general case, because of recursive rules combined with the problem of existential quantification. There are currently no implementations of SWRL, also because the language is in an early stage of development (the version of the language is 0.7). However, because SWRL has a first-order style semantics, a first-order theorem prover could potentially be used for reasoning in SWRL. Although, there have been initiatives and prototype implementations [Tsarkov and Horrocks, 2003] and Hoolet SWRL Reasoner [Hoolet] wherein reasoning is being provided via a translation to First Order Logic and the use of a theorem prover like Vampire[Vampire] but they have a naïve approach and are highly unlikely to scale. Another ongoing project is to extend the OilEd [OilEd] to deal with SWRL Ontologies. [MSc Projects, 2004]. SWRL has significant expressive power. It allows much more expressiveness with properties (binary predicates), negation, chaining etc, none of which can be captured in OWL. SWRL is more than syntactic sugar and adds Horn logic to DL at the cost of decidability. Decidability although is an issue but since in real systems where typically some form of bounded reasoning is done, the lack of decidability is not much important. The current issues and future improvements required in SWRL are discussed in [Tabet et al., 2004]. OWL-S has the ability to have multiple values for the properties. For example a business expressed in OWL-S can have different contact numbers for different clients. The OWL-S modeling includes information about the geographical radius of the service which might be an important parameter in a supply chain application.

2.2 WSMO

The mission of Web Service Modeling Ontology Project is to create a Web Service Modeling Ontology (WSMO) for describing various aspects of Semantic Web Services. WSMO is a refinement of WSMF [Fensel & Bussler, 2002]. By exhaustively deploying WSMO, they aim to solve the Web services integration problem. The service integration problem refers to the problem of seamlessly integrating multiple web services into a process. Since, different services may refer to different ontologies, may have incompatible inputs and outputs, it is hard to integrate Web services in an automated way. WSMO provides various features which will help in easy Web service integration.

WSMO build upon its main features of simplicity, completeness and easy executability. WSMO is aimed to be simple so that it can be easily understood by everyone, it will be complete by incorporating all the aspects of Web services and their composition and it will be easily executable.

WSMO consists of ontologies which are used to provide common vocabularies used by others. Goal repositories are used to define problems that need to be solved using Web services. Goals can be seen as the output of the execution of the process built using WSMO. Web service descriptions are provided to express the functionality and data interaction of a Web service. Finally, WSMO provides mediators which help in easying interoperability. A mediator is an element which can either refine an existing component to produce a new component or it can make two otherwise incompatible components interact with each other. The mediators which help refine components are called Refiners and the ones which can make two components interact are called Bridges. Mediators help make WSMO simple, efficient and support reuse of existing components.

Refiners are used to refine ontologies and goals. Given a pre-existing goal in the repository and there is need for a goal which is a little different but very similar to the pre-existing goal, a special refiner called ggMediator can be used to create the target goal from the pre-existing source goal. Similarly, another kind of refiner, ooMediator can be used to import pre-existing ontologies and refine them for use.
Bridges are used with Web services and goals. To link a Web service to a goal, wgMediators can be used and to make two Web services interact with each other, wwMediators can be used.

A Web service is usually defined by the operations it exposes, the inputs and outputs of the operations, the Quality of Service (QoS) parameters, etc. Similarly, in WSMO a Web service is defined by the Non-functional properties which include the QoS parameters of performance, reliability, security, trust etc. Mediators can be used to describe Web services by making use of ooMediators. The ooMediators can be used to import a different ontology in the present ontology and the ontology concepts can be used to define the inputs and outputs of the service. The functionality (operations exposed) of the Web service is defined by the Capability of the Web service. Multiple Web services can be composed together to get a composite Web service (commonly known as a Web process) to achieve complex functionality. In WSMO, interfaces are used to describe ways to achieve this functionality by using orchestration and composition of multiple Web services.

The WSMO ontology further consists of various elements. The non-functional properties like title, publisher, date and other Quality of Service (QoS) parameters like reliability, security etc. Ontologies make use of ooMEdiators which can help import a pre-existing ontology into the present ontology to help reuse. An ontology can also have axioms, which are logical expressions enriched by some extra-logical information. Axioms can be considered as rules which can be used to define constraints and other complex information. Concepts are used to provide an abstract view of the problem domain. The ontology can have relations which define relationships between several concepts and it can also have instances which would be the actual instances with which the ontology is populated.

WSMO uses F-Logic [F-Logic] syntax to provide axioms and logical inferencing support. F-logic (short for Frame Logic) was originally developed for defining, querying and manipulating database schema. F-logic has a sound and complete proof theory which makes it stand out from other logic languages. F-Logic also finds applications in Artificial Intelligence frame-based languages. F-Logic is extensible and can be combined with other specialized logics. F-Logic combines the advantages of typical frame based languages with the expressiveness, the compact syntax, and the well defined semantics from logics. Object identity, methods, classes, signatures, inheritance and rules are some of the features of F-Logic.

F-Logic is a full first order logic language. F-Logic provides a standard model theory and a second order syntax while staying in the first order logic semantics. It has minimal model semantics and implemented inference engines are already available. Let us take a look at the following WSMO example written in F-Logic:

```
"/* ontology */
        woman::person.
        man::person.
        person[father=>man].
        person[mother=>woman].
        person[daughter=>>woman].
        person[son=>>man].

/* rules consisting of a rule head and a rule body */
        FORALL X,Y X[son->>Y] <- Y:man[father->X].
        FORALL X,Y X[son->>Y] <- Y:man[mother->X].
        FORALL X,Y X[daughter->>Y] <- Y:woman[father->X].
        FORALL X,Y X[daughter->>Y] <- Y:woman[mother->X].
/* facts */
        abraham:man.
        sarah:woman.
        isaac:man[father->abraham; mother->sarah].
        ishmael:man[father->abraham; mother->hagar:woman].
        jacob:man[father->isaac; mother->rebekah:woman].
        esau:man[father->isaac; mother->rebekah].
/* query */
        FORALL X,Y <- X:woman[son->>Y[father->abraham]].
```

The first part of this example states that every woman is a person, every man is a person. For a person the attributes father, mother, daughter together with their respective ranges are given. The first rule describes that if X is the father of Y and Y is a man, then Y is the son of X. A similar relationship holds for sons and mothers, for daughters and fathers and daughters and mothers. The given set of facts indicate that some people belong to the classes man and woman, respectively, and give information about the father and mother relationships among them. The query shows the ability of F-logic to nest method applications. It asks for all women and their sons, whose father is Abraham." [Angele and Lausen]

By using F-Logic syntax, WSMO has achieved more expressivity. As can be seen in the example above, WSMO is written in F-Logic. Although it makes WSMO difficult to read, it adds the facility of expressing rules and facts based on ontology and also provides an easy querying mechanism.

My View:

The current standard of WSMO does not define how the orchestration is described. This defeats the purpose of having wwMediators. The main purpose of wwMediators is to reuse components to make one web service interact with another. Unless orchestration/web service composition is defined, wwMediators will have no use. In addition to that, the WSMO standard does not consider the case when the mediator fails or throws an exception. Will the previous Web Service be compensated, or will the mediator be restarted correctly and ultimately follow through with the invocation of subsequent Web Services? By incorporating a new type of component with different execution semantics, WSMO has made exception handling and transaction recovery even more difficult.

WSMO is based on F-logic which is a full First Order Logic and DL is a subset of it. Everything that can be expresses in OWL can be expressed in F-Logic. [Balaban, 1993] explains how F-Logic covers DL and how F-Logic can be used as a framework to define extensions of DL. In contrast to F-Logic, Description Logics are not applicable to reasoning in large sets of instances and thus cannot be used as a run-time system for applications based on ontologies having large number of instances. However there are misgivings that First Order F-Logic implementation may not be efficiently implementable, yet, Dieter Fensel is working on an implementation which will be decidable. A discussion thread about SWRL and F-Logic can be found at [Deri-WSMO].

The cardinality constraints in WSMO standard do not allow the definition of more than one value for the non-functional properties of the service which might become a big restriction for some applications. For Example, in a supply chain scenario, a manufacturer may have different contact numbers for different suppliers or vendors. The inability to have more than one contact may become a big deterrent. Geographic radius has been identified as an important property of a Web service in a supply chain scenario, but WSMO does not offer any such property for Web service description.

2.3 FLOWS

FLOWS (First Order Ontology of Web Services) is another proposal to the SWSL Committee. It provides the representational necessity for Web Service Choreography ontology. It can be seamlessly integrated with the existing and emerging standards like BPEL, W3C choreography, etc. It can capture activities. It can also process preconditions and effects. It has a taxonomic representation.

Using First Order Logic gives FLOWS a rich expressive power (e.g., variables, quantifiers, terms, etc.) and helps overcome the expressiveness issues that have haunted OWL-S. It also enables exploitation of the existing FOL reasoning engines and database query engines; however, it also makes it semi-decidable and intractable for many tasks in the worst case.

It is a first-order logic ontology based on PSL [PSL, 2004], a dialect of situation calculus. The aim of PSL is to create a language which will be common to all manufacturing applications. The language will allow for interoperability among different applications based on the common understanding of the shared concepts. PSL has been under development for years now, in the business process modeling area. It has been proved useful as exchange language and is extensible. It has specific expressiveness properties wherein it treats both simple and complex actions as first-class objects and has explicit representation of state.

PSL is a dialect of situation calculus [McCarthy and Hayes, 1969] which is a reasoning language. It can reason about actions based on Predicate Calculus (i.e. first-order logic). It defines fluents as the predicates and functions describing the current state of the world. Predicates are represented by Relational Fluents and function symbols are represented by Functional fluents which can be further used in equations.

"*Situations are terms of the form s or do(act; s), where s denotes a situation and act denotes an action and do is a distinct function symbol expressing the execution of an action. The initial situation is denoted by the constant symbol s0. For instance, in order to express that a block a is on top of block b after being moved there in situation s can be expressed as follows:*

∀a, b, son(a, b, do(move(a, b), s))" [Keller et al., 2004]

My View:

FLOWS ontology is based on PSL, a dialect of situation calculus. Although PSL is extensible and has been proved to be a useful exchange language, it is difficult to read or write. Also no implementation of an associated reasoner is currently available. There is no presentation of the entire ontology and no definition of concepts, composition, negotiation or dataflow is provided in the current FLOWS standard. It is still in a nascent stage and hopefully, its future standards will have improvements and additions to its present many deficiencies. No surface syntax for FLOWS has been defined as yet. Once this is done, it will make FLOWS easy to use and understandable by the users.

3. Comparison and Discussion

The three technologies described above use different formats for Web service description, different ontology standards to define the inputs/outputs and functionality of Web services, have different ways of defining composition and orchestration and different logic languages to provide rules and inference support. Also no implementation of the above technologies is available for testing and comparison.

To compare the three approaches, the following metrics will be used:

1) Surface Syntax: to check if a syntactic language is available to describe Web services
2) Computational Infrastructure: to check and compare the computational infrastructures (if available)
3) Concept Coverage: to check if all elements to define a Web service or process are defined
4) Ontology: to see if an ontology is available in the technology

5) Logic Language: comparison of the logic languages used in each initiative
6) Composition: to check if the technology defines composition of Web services
7) Properties: to compare the different types of properties offered by each technology

Table 1 shows the comparison of Enhanced OWL-S, WSMO, FLOWS and METEOR-S based on these metrics.

	Enhanced OWL-S	**WSMO**	**FLOWS**	**METEOR-S**
Surface Syntax	Available	Available	Not-Available	Available
Computational Infrastructure	Under Development although workarounds like [Hoolet] are available	Under Development	Under Development	Mostly Developed
Concept Coverage	Good	OK	Poor	Good
Ontology	Yes	Yes	None	Yes
Logic Language	SWRL	F-Logic	First Order	OWL
Composition Defined?	Yes	No	No	Yes
Properties	Profile can have multiple non functional properties	Cardinality restrictions on non-functional properties	No definition available	Properties are defined using OWL

Table 1: Comparison of Enhanced OWL-S, WSMO, FLOWS and METEOR-S

Let us now discuss and compare the OWL-S, WSMO and FLOWS based on the above metrics. We will discuss METEOR-S in the next section.

Surface Syntax: Enhanced OWL-S provides a surface syntax using OWL. OWL-S has defined the syntax to be used for creating service profile, process model and grounding. Once SWRL is integrated with it, it will help express more complex rules and constructs in OWL-S. WSMO also provides a surface syntax using F-Logic. WSMO is written in F-Logic. Although it is not as readable as OWL-S, F-Logic adds the power of expressing everything in a simple way. The ontology, facts, rules and queries can be expressed using F-Logic syntax. FLOWS although has no description of what surface syntax it will be using. Since PSL is difficult to read or write, FLOWS may have to come up with a different surface syntax which will integrate with PSL.

Computational Infrastructure: The computational infrastructure provides reasoning support for interpreting queries, rules and semantic descriptions. OWL-S is working on creating a SWRL reasoner to do inferencing with rules expressed in SWRL. Although workarounds like [Hoolet] are available for SWRL inferencing but they mostly have a naïve approach and are not likely to scale. WSMO also does not have any reasoner or computational infrastructure associated with it. They are trying to come up with a WSMO F-Logic reasoner which will convert F-Logic into First Order Logic. No further details were provided. The computational infrastructure for FLOWS is also unavailable. Before building a reasoner over FLOWS, other important descriptions like surface syntax, properties, composition are required.

Concept Coverage: OWL-S has the most concept coverage, since Web service definition elements in WSMO are a subset of those in OWL-S. For example WSMO does not include geographic radius and a few other Web service description elements. By rating WSMO as OK we are saying that the concept coverage in

WSMO is not as good as in OWL-S (rated good) but is sufficient enough to define Web services. FLOWS on the other hand has minimal concept coverage mostly, because they are still in the process of creating an surface syntax upon which the concept coverage will be based. Its concept coverage is therefore rated poor.

Ontology: Enhanced OWL-S and WSMO have available ontologies. OWL-S uses OWL ontologies and will use SWRL in future. WSMO can create its own ontologies using F-Logic or import existing ontologies by using mediators. FLOWS has no ontology available as yet.

Logic Language: Enhanced OWL-S will use SWRL as its logic language. SWRL is based on horn logic. WSMO makes use of F-Logic to add rules and axioms to it. FLOWS is based on PSL which is a dialect of Situation Calculus. Unless implementations of the associated reasoners are not made available, it will be difficult to compare the efficiency of the logic languages and the reasoner.

Composition: OWL-S defines composition in its process model. It has various constructs to define sequence, flow, parallel execution etc, to define the way multiple Web services can be composed into a process. WSMO, on the other hand, does not define composition. Although the ability of WSMO to compose Web services is mentioned, but the way it is achieved is still not clear. No syntax or procedure for composition was provided by WSMO. FLOWS also does not define composition.

Properties: OWL-S offeres the most properties as compared to the other two technologies. OWL-S can have multiple non-functional properties. For example a business may like to have different contact addresses for different partners or clients. But cardinality constraints in WSMO, restricts it to having only one value for each non-functional property. Therefore, a WSMO implementation cannot express multiple encryption standards that are supported or multiple contact addresses. On the other hand, the OWL-S profile hierarchy does not define the IOPEs of the service, but defines taxonomy of profiles instead. WSMO standard defines this taxonomy by making use of real goals i.e. it gives the categorization by using mediators to refine existing goals. FLOWS does not define any properties. The cardinality constraint in WSMO may also be considered on the positive side as providing the ability to restrict the values of a property to one which may be useful in some scenarios.

So, we can see that it is hard to say which technology is better than the other since none of them is complete as yet and neither do they provide an implementation which can be tested and compared. All three technologies use First order logic to add more expressivity by rules and inferencing. The problem with more expressivity using First Order Languages is that it makes it semi-decidable and intractable in the worst case. To address the decidability issue, one practical approach is to set and use a time-limit, for finding the answer. If within a certain period of time, proof of obligation could not been established, stop the prover and assume goal cannot be matched. The other approach is to use a subset of OWL-Lite [OWL-Lite-] which addresses the cardinality constraint problem [Raphael et al., 2004] and has other language restrictions to make SWRL decidable. For WSMO, restricting the language for specifying goals and capabilities can make it decidable.

4. Comparison of OWL-S with METEOR-S

The METEOR-S project in the LSDIS Lab at the University of Georgia aims to create a comprehensive framework for composing Web processes. METEOR-S is based on industry standards like WSDL [Christensen et al., 2001], OWL and BPEL [Andrews et al., 2003]. METEOR-S uses the extensibility elements in these languages to add semantics to them. Through semantic annotations METEOR-S achieves automated discovery and composition. The annotations are reduced to the source code level [Rajasekaran et al., 2004] since it is believed that the person who is writing the code is the one who best understand the functionality and definition of the service and hence its semantics. A WSDL document can also be annotated to create a semantic Web service. METEOR-S constructs annotated WSDL1.1 from the source code and can publish semantic information in the registry. METEOR-S builds a layer above the UDDI v.2 [Belwood et al., 2002] standard to incorporate semantics in the available data structures.

There can be multiple approaches for Web process composition. METEOR-S uses an abstract process containing abstract services as a starting point. An abstract service is a placeholder for a set of services matching the abstract service's template. In some cases the set may have cardinality greater than one, for example, a set of competing services. In this way, the topology of the service process is largely fixed; however, actual service selection may be highly dynamic. An alternative approach to composition used by OWL-S is planning. The OWL-S approach is to not start with a basic abstract process, but rather form a set of goals and build the whole process. Several AI researches are investigating the use of planning agents for this purpose.

While METEOR-S builds on the existing standards of WSDL and BPEL, OWL-S defines its own service, profile, grounding and process. There is not much difference in the METEOR-S and OWL-S approach to discovery. OWL-S has Inputs, Outputs, Preconditions and Effects while METEOR-S defines Inputs, Outputs, Operations, Preconditions, Postconditions and Exceptions. Discovery in OWL-S is logical subsumption based discovery whereas METEOR-S has a heuristic approach for discovery which has a polynomial time complexity. In the heuristic approach, while trying to give a match value to two concepts, METEOR-S does not use a subsumption algorithm to check whether a concept is subsumed by the other. Instead it checks to see how many properties of one concept matches the properties of the other, checks if one concept is the parent of another, checks if the two concepts are siblings, check the distance of the common parent, find distance between the two concepts etc. Thus it is able to achieve concept matching in polynomial time. As an example consider a concept "Car" with two sub-concepts "sports car" and "luxury car". If we are trying to find a match between sports car and luxury car, the subsumption algorithm will give it a "no match". The METEOR-S heuristic discovery algorithm will match the properties of sports car and luxury car (most of them will match), check for siblings (true in this case) and check the distance of the common parent (1 in this case) and hence, give a high match value. This helps METEOR-S include other important relationships which are not just subsumption based. For example logical subsumption may ignore concepts that are placed at a relatively greater distance in the ontology and that do not satisfy parent-child relationship. These concepts will, however, be considered while matching in the discovery algorithm of METEOR-S.

METEOR-S is based on OWL. It provides the ability to define constraints on services for discovery and composition. METEOR-S uses Snobase [Snobase, 2003] as an inference engine which queries ontologies to estimate cost for constraints. The constraints are then fed into an Integer Linear Programming (ILP) solver which matches the user constraints with the service constraints and returns optimized results [Aggarwal et al., 2004]. METEOR-S uses OWL as its logic language. Being the industry standard, OWL has wide applications and acceptance but it also restricts the expressivity to some extent. METEOR-S was also successfully experimented [SQUID, 2003] with Prolog [SWI-Prolog] and by adding constraints and rules as RDF triples. METEOR-S will soon support SWRL. Currently, the constraints in METEOR-S are expressed in OWL which is a very restrictive language as in it does not allow complex computations and comparison operators. Using SWRL's built-in functions for computations otherwise impossible in OWL, the expressivity of METEOR-S will increase many folds.

Enhanced OWL-S will have SWRL built into it and they are working on a SWRL reasoner. SWRL makes OWL-S very expressive and since SWRL is based on OWL, the OWL concepts can be used and enhanced using SWRL. Although OWL-S has well defined semantics, it does not give us a lot more than link semantics in BPEL. Improvements are required in the OWL-S process model to make the process and service descriptions machine readable and unambiguous. Unlike BPEL, OWL-S classes may draw property inheritance and other relationships to other OWL-S classes whereas in BPEL, port type information is used for service descriptions, WSDL does not define side effects and the expressiveness of inputs/outputs is constrained by XML Schema Definition (XSD) and WSDL's complex types. On the other hand, BPEL4WS defines a mechanism for catching and handling faults similar to common programming languages like Java, while compensation and recovery protocols are not defined in OWL-S. The forthcoming release of BPWS4J will include Java constructs in BPEL which will allow service composers to write service composition and Java code in the same file to increase the expressive and computational power.

METEOR-S defines four different kinds of semantics: data, functional, QoS and Execution which are more comprehensive than the OWL-S semantics. Moreover, OWL-S has no available implementation of their virtual machine for executing OWL-S processes while METEOR-S also is releasing its first implementation in August 2004, complete with the annotation, publication, discovery, composition and the BPWS4J execution engine. Also the digression of OWL-S from the industry standards may make it difficult for the industry to adopt.

Based on the metrics defined in Section 3 for comparing the current semantic Web services technologies, METEOR-S has the surface syntax available. METEOR-S builds upon the existing standards of WSDL, SOAP, UDDI and BPEL. METEOR-S uses the syntax defined for these standards and makes use of the extensible elements for adding semantics. The Computational Infrastructure for METEOR-S is also mostly developed. The logic languages OWL used by METEOR-S to specify rules and constraints already have associated reasoners available. The future versions of METEOR-S will have SWRL integrated with the system, for which computational infrastructure will be required. The Concept Coverage in METEOR-S is good since they have no restriction on the properties that can be used or the values which the properties can have. WSDL is used to describe the Web services and its extensible elements are used to further add semantics to it. Using the extensibility elements, any new element/property to describe a Web service can be seamlessly added to METEOR-S. METEOR-S uses OWL ontologies to annotate the inputs/outputs and functionality of a Web service. It also makes use of a QoS ontology to describe the QoS parameters of the service. It uses BPEL as the standard for composing Web processes. BPEL has support for compensation and recovery which is lacking in the other technologies.

5. Conclusion

As we can see from the above discussion that no work on defining a technology for semantic Web service language is complete as yet. There is still room for improvement. Although there has been progress in defining semantics for composition, execution semantics have not been defined by any of the initiatives discussed. Due to the idiosyncrasies and incomplete specifications of the current initiatives, the industry and research community is finding it difficult to adopt one specific technology/initiative for universal application. Through the survey presented above and also through our experience in the METEOR-S project, we have discussed the many areas which need to be addressed and the problems which need to be solved to make the semantic Web language specification complete. We feel that if these areas can be improved upon, a much more comprehensive, robust and complete semantic Web language technology can actually be realized which would provide a strong foundation for the future of semantic Web services.

6. References

[Aggarwal et al., 2004] R. Aggarwal et al., Constraint Driven Web Service Composition in METEOR-S, proceedings of IEEE SCC, 2004

[Andrews et al., 2003] Andrews et al., Business Process Execution Language for Web Services Version 1.1, available at http://www-106.ibm.com/developerworks/webservices /library/ws-bpel/, 2003

[Angele and Lausen] J. Angele and G. Lausen, Ontologies in F-Logic, Handbook on Ontologies in Information Systems. International Handbooks on Information Systems, Springer Verlag.

[Belwood et al., 2002] Universal Description and Discovery Interface Version 2.04, available at http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm, 2002

[Boley et al., 2004] Boley et al., The Rule Markup Initiative, http://www.ruleml.org, 2004

[Borgida et al., 2000] Borgida et al., Explaining ALC Subsumption, http://www.cs.man.ac.uk/~horrocks/Publications/download/2000/ECAI-2000.ps.gz, 2000

[Chinnici et al., 2004] Web Services Description Language (WSDL) 2.0, available at http://www.w3.org/TR/2004/WD-wsdl20-20040326/, 2004

[Christensen et al., 2001] Web Services Description Language (WSDL) 1.1,http://www.w3.org/TR/wsdl, 2001

[Curbera et al., 2004] Curbera et al., IBM Business Process Execution Language for Web Services Java^{TM} Run Time, http://www.alphaworks.ibm.com/tech/bpws4j, 2004
[DAML-S] DAML-S Coalition, Web Service Description for the Semantic Web, 2002
[Deri-WSMO] Deri-wsmo-discussion, http://informatik.uibk.ac.at:2081/pipermail/deri-wsmo-discussion/, 2004
http://www.cs.man.ac.uk/~horrocks/DAML/Rules/, 2003
[F-Logic] Frame Logic, http://flora.sourceforge.net/aboutFlogic.php
[Fensel & Bussler, 2002] D. Fensel and C. Bussler: *The Web Service Modeling Framework WSMF*, Electronic Commerce Research and Applications, 1(2), 2002.
[FLOWS][Berardi et al., 2004] Daniela Berardi, Michael Gruninger, Rick Hull, Sheila McIlraith, *FLOWS (First-order Logic Ontology for Web Services)*,
http://www.daml.org/services/swsl/straw-proposals/FLOWS-Proposal-05-13-2004.pdf, 2004
[FOL] First Order Logic , http://mathworld.wolfram.com/First-OrderLogic.html
[Grosof, 2004] Benjamin Grosof, SCLP Rules + Ontologies to Describe E-Services, 2004
[Hoolet] Hoolet SWRL Reasoner, http://owl.man.ac.uk/hoolet/, 2004
[Horrocks et al., 2003] A Proposal for an OWL Rules Language, Semantics and Abstract Syntax,
[Horrocks et al., 2004] Horrocks et al., SWRL: A Semantic Web Rule Language Combining OWL and RuleML, http://www.daml.org/rules/proposal/, 2004
[IRH1] MSc Projects at University of Manchester,
http://www.cs.man.ac.uk/mscprojects/projects.04/irh1.html,
www.cs.man.ac.uk/ugrad/projects/year04/know.html, 2004
[Keller et al., 2004] Language Evaluation and Comparison, http://www.wsmo.org/2004/d8/v01/20040308/
[Kifer, 2004] Michael Kifer, The Enchilada Proposal: F-logic, HiLog and Concurrent Transaction Logic, 2004
[McCarthy and Hayes, 1969] McCarthy, J. and Hayes, P. J. (1969). Some Philosophical Problems from the Standpoint of Artificial Intelligence. In Meltzer, B. and Michie, D., editors, *Machine Intelligence 4*, pages 463-502. Edinburgh University Press.
[METEOR-S, 2002] METEOR-S:Semantic Web Services and Processes, http://swp.semanticweb.org, (2002).
[OilEd] Ontology Inference Layer Editor, http://oiled.man.ac.uk/, 2001
[OWL] [McGuinness et al., 2004] McGuinness et al., OWL Web Ontology Language Overview http://www.w3.org/TR/owl-features/ , 2004
[OWL Rules draft] [Grogof, 2003] slightly revised long version of warning label section for OWL Rules draft, available at, http://www.daml.org/listarchive, /joint-committee/1491/html, 2003
[OWL-S][Martin et al., 2003] Martin et al., OWL-S 1.0 Release, http://www.daml.org/services/owl-s/1.0/owl-s.html , 2003
[SWI-Prolog] http://www.swi-prolog.org/
[PSL, 2004] Process Specification Language, http://www.mel.nist.gov/psl/, 2004
[Rajasekaran et al., 2004] P. Rajasekaran et al., Enhancing Web Services Description and Discovery to Facilitate Orchestration, proceedings of SWSWPC, 2004
[Raphael, 2004] Volz, Raphael, Web ontology reasoning with logic databases, http://www.ubka.uni-karlsruhe.de/indexer-vvv/2004/wiwi/2, Universität Karlsruhe, Fak. f. Wirtschaftswissenschaften. Diss. v. 17.02.2004
[Schlobach et al., 2003] Explanation of Terminological Reasoning, A Preliminary Report,
http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-81/schlobach.pdf, 2003
[Snobase, 2003] Semantic Network Ontology Base, IBM Ontology Management System, 2003,
http://www.alphaworks.ibm.com/tech/snobase
[SQUID, 2003] Rajasekaran et al., Semantically Querying the UddI for Discovering Web Services, 2003
[SWSL] Semantic Web Services Language (SWSL) Committee, available at http://www.daml.org/services/swsl/
[SWSL Requirements] [Grogof et al., 2004] Semantic Web Services Language Requirements Version 1, available at, http://www.daml.org/services/swsl/requirements/swsl-requirements.shtml

[SWSL Straw Proposals] SWSL Straw Proposals, available at, http://www.daml.org/services/swsl/straw-proposals/, 2004
[Tabet et al., 2004] Tabet et al., SWRL Issues List and Language Extensions, http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/issuesList.html, 2004
[Tsarkov and Horrocks, 2003] Dmitry Tsarkov and Ian Horrocks, DL Reasoner vs. First-Order Prover, http://www.cs.man.ac.uk/~horrocks/Publications/download/2003/TsHo03a.pdf, 2003
[Vampire] Vampire, http://www.cs.man.ac.uk/~riazanoa/Vampire/
[WSMO][Roman et al., 2004] Roman et al., Web Service Modeling Ontology - Standard (WSMO – Standard) available at, http://www.wsmo.org/, 2004
[WSMO vs. OWL-S] http://www.wsmo.org/2004/d4/d4.2/v0.1/