Language Evaluation and Comparison

Semantic Web Fred

Net Dynamics & University of Innsbruck

Ioan Toma
Michael Stollberg
Jos de Bruijn
Axel Polleres
Rubén Lara

SWF-Deliverable: D2
Version: 1.0
Date: August 2004

Executive Summary

This document analysis existing representation technologies relevant for the Semantic Web Fred project with regard to their usability in the project. This document provides the scientific rationale for the choice of representation formalisms applied in the SWF Description Language for Cooperative Goals and SWF Services, specified in SWF Deliverable D3 "SWF Goal and Service Description Language". The Semantic Web Fred project is funded by the "Wiener Wirtschaftsföderungsfonds", short: WWFF. The project partners are:

1. Net Dynamics Internet Technologies GmbH, project leader
2. Next Web Generation Group at Institute for Computer Science at the University of Innsbruck (DERI Austria), scientific cooperation partner.

Contents

1 Introduction

This document analyzes different languages for describing, representing, and handling different kinds of technological constructs that are relevant in the Semantic Web Fred project. In the context of SWF, the representation techniques for the following aspects are relevant:

1. **Static World Representation** for expressing ontological knowledge.
2. **Dynamics & Business Process representation** for specifying business processes and multi-step services.
3. **Overall Description Frameworks for Semantic Web Services** as similar approaches to SWF.
4. **Interaction & Protocol techniques** for describing the external visible behavior of SWF Services and determine their compatibility of automated cooperation.

In this document we inspect existing techniques for each of these aspects and point out the differences with regard to the usability of specific technologies in SWF. The document is structured as follows: Chapter 2 briefly describes the two most relevant initiatives in the Semantic Web Services area: WSMO and OWL-S, being centered on comparison of these two approaches. Chapter 3 survey the existing languages used for representing static aspects of the world. Dynamic aspects of the world, business processes and also interaction and protocol techniques (Chapter 4) used within SWF are not presented in this version of the document. They will be part of the next version. Finally, Chapter 5 summarizes our conclusions.

2 Service Description Frameworks

This section provides an overview of the service description frameworks used to semantically describe Web Services in order to enable the automation of Web Service discovery, composition, interoperation and invocation. There are two major initiatives, namely: OWL-S and WSMO which aim to realize Semantic Web Services by enhancing current Web Services with semantics. In this section we provide an overview of OWL-S and WSMO, and in parallel we conduct a comparison that identifies the overlaps and differences between the two initiatives. Based on this comparison and on the requirements of SWF we select the most appropriate service description framework for use within SWF project.

2.1 OWL-S

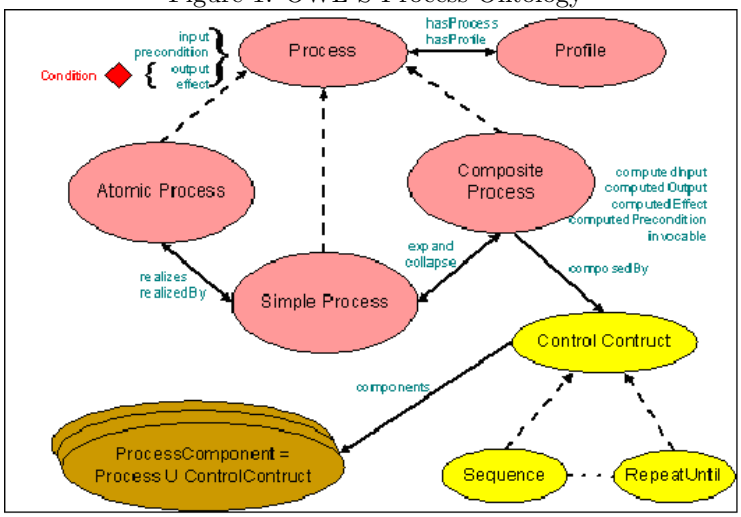
OWL-S is the Semantic Web Services effort of the DAML-programme, which is the major US-American Semantic Web research effort[1]. OWL-S has been the first approach for an overall framework for describing Semantic Web Services, starting in 2001 and have as a predecessor DAML-S. OWL-S defines an ontology system for describing Web Services, using OWL as the description language. The OWL-S upper level ontology comprises four major elements: Service, Service Profile, Service Model and Service Grounding.

1. the **Service** concept serves as an organizational point of reference for declaring Web Services; every service is declared by creating an instance of the Service concept
2. the **Service Profile** holds information for 'service advertisement' which is used for Web Service Discovery. This is the name of the service, its provider and a natural language description of the service, as well as a black-box description of the Service (specifying the input, output, preconditions and effects (short: IOPE).
3. the **Service Model** contains descriptive information on the functionality of a service and its composition out of other services, described as a process. The model defines three types of processes (atomic, simple, and composite processes), whereof each construct is described by IOPEs, as in the Service Profile, with optional conditions over these.
4. the **Service Grounding** gives details of how to access the service, mapping from an abstract to a concrete specification for service usage. Although not restricted to one grounding technology, WSDL (see below) is favored for this.

In OWL-S Web Services are understood as processes whereby the term process is used in the sense of an activity, as in its antecedent DAML-S[2]. The objective of the OWL-S process model is to define an ontology that covers all information needed for semantically enhanced Web Service composition [The OWL Services Coalition, 2004]. That is descriptive information of Web Services to enable dynamic discovery and composition following the declarative idea and procedural process modelling concepts to specify the control level. These are modelled in the OWL-S Process Ontology. Figure 1 show the structure of the OWL-S Process Model with further explanations below.

[1] `www.daml.org`.
[2] DAML Services, `http://www.daml.org/services`

Figure 1: OWL-S Process Ontology



OWL-S defines three types of process (i.e. Web Services): Atomic, Simple and Composite Processes. A process is described via inputs, outputs (concerned with data), pre-conditions and effects (concerning state-of-the-world conditions, described by according condition concepts). Further, the OWL-S Process Ontology defines basic control constructs (Sequence, Split, Fork + Join, Unordered, Condition, If-Then-Else, Iterate, Repeat-While, Repeat-Until). The Process Ontology only provides very basic modelling concepts for processes, especially regarding the control constructs. Further it is unclear how the process descriptions (input, output, pre-conditions and effects) are meant to be used for dynamic discovery and composition of Web Services or if it shall be used for describing the interaction behavior of a Web Services. Because of this, current research proposes results propose to replace or enhance the OWL-S Process Model with ontological descriptions based on the process models underlying BPEL4WS [Curbera et al., 2002] or BMPL [3]/ WSCI [4].

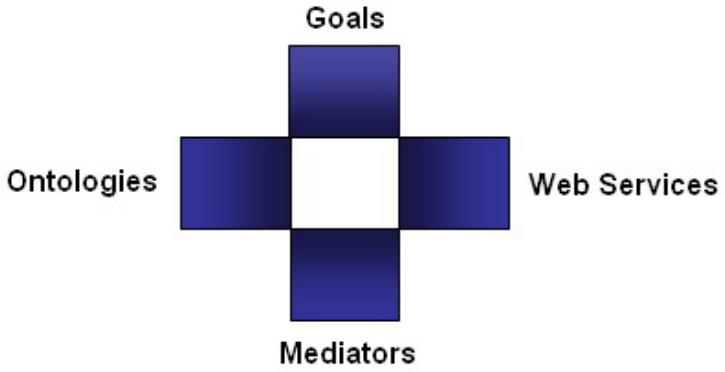
2.2 WSMO

The Web Service Modelling Ontology[5] is the Semantic Web Services research effort of a joint initiative of European research projects around the Semantic Web. Similar to OWL-S, WSMO aims at the development of an overall framework for Semantic Web Services in order to support automated Web Service discovery, composition, and execution. Although WSMO started in the beginning of 2004 and work is still ongoing, remarkable results have been achieved since then. WSMO defines four top-level notions related to Semantic Web Services, shown in Figure 2.

For management purposes, every WSMO component is described by non-functional properties, based on the Dublin Core Metadata Set [Weibel et al., 1998] that is defined as a generic description model for information items. In accordance to the Web Service Modelling Framework WSMF [Fensel and Bussler, 2002] as its theoretical foundation, WSMO applies two major design principles:

[3] http://www.bpmi.org/.
[4] http://www.w3.org/TR/wsci/.
[5] www.wsmo.org.

Figure 2: WSMO Components



1. *principle of maximal de-coupling*: all WSMO components are specified autonomously, independent of connection or interoperability with other components

2. *principle of strong mediation*: the connection and interplay between different components is realized by Mediators that resolve possible occurring heterogeneities between the connected components

In its current version, WSMO Standard [Roman et al., 2004] specifies the following description elements and components:

1. **Ontologies:** In WSMO Ontologies are the key to link conceptual real world semantics defined and agreed upon by communities of users. Ontologies define a common agreed upon terminology by providing concepts and relationships among the set of concepts. In order to capture semantic properties of relations and concepts, an ontology generally also provides a set of axioms, which means expressions in some logical framework.

2. **Goals:** describe user requests that should be resolved by invoking a service. The requests are expressed as logical expressions in the WSMO language on basis of domain ontologies which define by means of desired *post-conditions* and *effects* when the Goal is solved.

3. **Web Services** are described by a functional description, called *Capability*, and by an Interface that describes the usage interface for interacting with the Web Service in its *Choreography*, as well as how the Web Services is realized by using and combining other Web Services in the so-called *Orchestration*. The Capability has a similar intention as the Service Profile in OWL-S, i.e. advertising the Web Service as the support for Service Discovery. It defines the input requested by the service in order to fulfill its functionality and conditions over this (called *pre-condition*), arbitrary constraints on the world that have to hold before the Web Service can be executed (called *assumptions*), the output of the service in relation to the input and conditions over this (called *post-condition*), and conditions on the state of the world that result from the execution of the Web Service (called *effects*). The Web Service Interface notions have the same purpose as the Service Model within OWL-S (descriptions of service usage and service composition), but are more elaborated with regard to the conceptual model and the specification languages used. The *Choreography* of Web Service describes the consumption interface, meaning it specifies the external visible behavior of the service when consuming its functionality

along with the communication technology and an error framework for errors that can occur within service usage. It also comprises the grounding to executable technologies (as the Service Grounding in OWL-S) not limited to WSDL, SOAP. The *Orchestration* decomposes the functionality of a Web Service into sub-tasks, and defines how these functionalities are achieved by using and combining possibly dynamically bound other Web Services via mediators.

4. **Mediators** in WSMO are the component for realizing the underlying principles of strong de-coupling and mediation. Whenever WSMO components are to be connected (meaning: assembled in an application scenario), a Mediator connects these components and provides mediation facilities in order to resolve possibly occurring heterogeneities that humble interoperability. Consequently, the description elements for WSMO Mediators are source and target components, and the needed mediation facilities. WSMO defines four types of Mediators: *OO Mediators* connect ontologies and import them as terminology definitions into other components, *GG Mediators* for connecting Goals, *WG Mediators* connect Goals and Web Services, and *WW Mediators* connect Web Services.

2.3 Comparison & Conclusions

Although OWL-S serves as a basis for various research and development activities, it is heavily criticized for conceptual weaknesses and incompleteness. Especially, the meaning of the OWL-S description elements is not clearly defined, leading to misinterpretations and incompatible models; furthermore, OWL is used as the only specification language in OWL-S, which lacks in terms of expressiveness, especially for the process constructs required within the Service Model [Lara et al., 2004]. The Web Service Modelling Ontology WSMO[6], described in detail in Section 2.2, aims at overcoming these deficiencies by providing an unambiguous description framework for Semantic Web Services. WSMO is not restricted to Web Service description only, but includes additional top-level notions: Goals for representing user desires in a service-oriented architecture, and the concept of Mediators for handling heterogeneity. Thus, WSMO provides a more complete framework for aspects and challenges arising within Semantic Web Services.

As stated in Section 2.2, WSMO encompasses the notions defined within OWL-S for semantically describing Web Services. In addition, further top level constructs are defined which are considered to be relevant for realizing and facilitating Semantic Web Service technologies. Also, the shortcomings of OWL-S with regard to expressiveness of the used specification language and the unclear meaning of the description elements are tackled by more precise definitions. Summarizing, WSMO seems to be a promising approach towards Semantic Web Services which is also flexible and open enough to be adapted as *the* Semantic Web Services Framework in SWF.

[6] www.wsmo.org.

3 Representing static aspects of the World

This section provides an overview of the existing languages used for representing static aspects of the world and rationalize the use of WSMO/WSML languages for representing static knowledge in SWF. In our opinion the important approaches that deal with static world representation are: Description Logics, Logic Programming, Description Logic Programming (an intersection of the previous two), and F-Logic. In the context of Semantic Web another approach called OWL has emerged. OWL is based on Description Logics and is a Web oriented language for representing knowledge. We start with an overview of Description Logic, Logic Programming, Description Logic Programs and Frame Logic, the most prominent approaches for representing static knowledge, and then we continue with an overview of OWL, the Web language for knowledge representation. Finally we present a new initiative, a mediated approach based on OWL which exploits the intersection of Description Logics and Logic Programming, and commonly used features of F-Logic. In the last part of this section a comparison of the above mentioned approaches for representing knowledge about static aspects of the world, in the context of SWF project, is provided.

3.1 Description Logics

Description Logics [Baader et al., 2003] are logical formalisms for representing information about individuals, classes of individuals and their description. They historically evolved from semantic networks and frame based systems. Its main purpose was to overcome some of the problems of frame based systems and to provide a clean and efficient formalism to represent knowledge.

Description Logics use three basic elements to model knowledge about the world. These are: *concepts*, *roles* and *individuals*. Concepts are used to represent classes, entities or sets of individuals of some universe. They can be seen as unary predicates. Roles on the other hand denote properties and relations. They can be seen as binary predicates. There are also Description Logic languages, e.g. $\mathcal{DLR}$ [Calvanese et al., 1998] which allow n-ary roles. The third basic element in Description Logics are individuals which represents instances of the concepts, of the classes.

Description Logics distinguish between two levels of representing knowledge. The first one, known as TBox contains declarations that describe general properties of concepts. It captures the intensional knowledge about the domain in the form of a terminology. The second one, known as ABox, contains extensional knowledge, knowledge that is concerned with the individuals of the domain. The basic form of declaration in a TBox is a concept definition that is the definition of a new concept in terms of other previously defined concepts. The basic reasoning tasks in the TBox are: satisfiability, subsumption, equivalence, disjointness [Baader et al., 2003]. On the other hand in ABox, the basic reasoning task is instance checking which verifies whether a given individual is an instance of a specific concept. The main feature of Description Logics is their reasoning support. Reasoning in TBox can include one of the following four tasks: *satisfiability*, *subsumption*, *equivalence* and *disjointness*. TBox reasoning is supported by all important reasoners implementation. Some of them, like FaCT and Racer have optimized algorithms for TBox reasoning. In ABox the typically reasoning tasks are: *consistency*, *instance checking*, *retrieval* and *realization problem*. We should mention that not all the Description Logic reasoners support ABox reasoning. Reasoning in the ABox is not a trivial task because of the large amount of individuals. A solution to optimize ABox reasoning is to

used a relational database to speed up retrieval process. Such an optimization is implemented in a system called *instance store* [Bechhofer et al., 2002].

Formally, Description Logic languages typically have set-based model-theoretic semantics, in which a description D is mapped to a subset of a domain $\Delta^{\mathcal{I}}$ under an interpretation $\mathcal{I}$ using a mapping function $\cdot^{\mathcal{I}}$. Similarly, a role R is mapped to a binary relation over the domain $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Equivalence of descriptions is interpreted as equal subsets ($C \equiv D$ is interpreted as $C^{\mathcal{I}} = D^{\mathcal{I}}$), subsumption is interpreted as a subset relation ($C \sqsubseteq D$ is interpreted as $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$), and so on. The semantics of Description Logic adopts the open-world assumption, that means that the absence of information is not interpreted as negative information. This is different from Logic Programming and Databases approaches which adopt a close-world assumption.

3.2 Logic Programming and Datalog

Logic Programs are finite sets of Horn formulae. A typical Horn formula is a disjunction of literals with at most one positive literal, in which all variables are universally quantified. A Horn can be written in the following form:

$$p \leftarrow n_1 \wedge \ldots \wedge n_k \tag{1}$$

There are three types of Horn formulae: *rules*, *facts* or *queries*. A *rule* is a Horn clause with one positive literal, and a least one negative literal. The positive literal is called the *head* of the rule. The conjunction of the negative literals $n_1 \wedge \ldots \wedge n_n$ is called the *body* of the rule. A *fact* is nothing more then a rule without a body and a *query* is a rule without a head. A *logic program* consists of a set of horn clauses.

The semantics of Logic Programming can be defined in several ways, but the two most important approaches are: model theoretic semantics and fixpoint semantics. The model-theoretic semantics use the Herbrand theory, more precisely the semantic of an logical program P is given by the minimal Herbrand model M_P. which imposes a syntactic restriction on the admissible structures. The second approach is the fixpoint semantics. Fixpoint semantics utilizes a function T_P which takes the set ground facts in P and computes a new set of facts by applying the definite program clauses of P.

Logic Programming languages have many variants. One of the most interesting variant of Logic Programming with support for databases is Datalog.

Datalog is the most prominent language for *deductive database*, which are a combination of conventional databases, knowledge bases and inference engines. The underlying mathematical model of Datalog is basically a relational model. In Datalog the relations are represented by predicates symbols. Predicates can be of two types: *extensional* predicates and *intensional* predicates. Predicates which are defined by *facts* are stored in an extensional database(*EDB*). On the other hand predicates which are defined by *rules* are stored in an intensional database(*IDB*). The two databases, the extensional database and the intensional database are the two parts of each Datalog database.

Logic Programming and Deductive Programming(Datalog) support one basic type of reasoning, namely *query answering* or answering *atom queries*. Atom queries can be: *Open Atom Queries* where the task is to determine all substitution for all substitutions(variable binding), and *Ground Atom Queries* where the task is to determine whether a ground atomic formula is entailed. For the complexity problem of Logical Programming we can distinguished three types:

data complexity, *program complexity* and *combined complexity*, the combination of the previous two.

3.3 Description Logic Programs

Description Logic Programs [Volz, 2004] is the intersection of previous described formalisms: Description Logic(DL) and Logic Programming(LP), more precisely Datalog. The new define language $\mathcal{L}_0$ is more restrictive but provides a way to "build rules on top of ontologies" [Grosof et al., 2003]. Logic rules can be translated to/from Ontologies, and conclusions derived in one formalism can be translated into the other. Another benefit of using the intersection of DL and LP to define Description Logic Programs, is that the efficient existing implementation of LP rule and databases engines can be used for Description Logics fragment.

$\mathcal{L}_0$ has the following properties, which make it a good candidate to be the basis for an ontology language on the Semantic Web:

- $\mathcal{L}_0$ contains the most frequently occurring Description Logics modeling primitives. It turns out that most OWL (and DAML+OIL) ontologies in popular ontology libraries fall in this fragment [Volz, 2004].
- The most important reasoning task in deductive databases is query answering. There has been significant research on the properties of Datalog with respect to the task of query answering. In $\mathcal{L}_0$, all individual-related reasoning tasks can be reduced to query answering in Datalog. This allows us to benefit from the research and implementations in the deductive database area.
- Because $\mathcal{L}_0$ is a proper subset of both Datalog and a Description Logics, it is straightforward to extend the language in either direction. In fact, $\mathcal{L}_0$ can be used to allow inter-operation between the two paradigms.

3.4 Frame Logic

Frame Logic, shortly F-Logic, [Kifer et al., 1995] is a higher-order language for reasoning about objects, inheritance and schema. It provides second-order syntax for a first-order logical language. The basic idea of F-logic is to take complex data types as in object-oriented databases and to combine them with logic. The result is powerful programming and query language. F-Logic was inspired by work on deductive databases, Object-Oriented databases and Object-Oriented programming. It stands in the same relationship to the object-oriented paradigm as classical predicate calculus stands to relational programming [Kifer et al., 1995].

Important features of F-Logic include: *object identity*, *methods*, *classes*, *signatures*, *inheritance* and *rules*. *Object identity* [Abiteboul and Kanellakis, 1989] is used to unique identify an object. There are two disjoint universes of object identifiers. The one universe encompasses the classes, instances, values, etc. The other universe encompasses the attribute and method identifiers. A consequence of this is that in F-Logic there is no distinction between classes and instances. An object can be both a class and an instance.

One can write F-Logic programs using a set of symbols and variables. A term is a normal first-order term composed of functions symbols and variables. A language in F-Logic consists of a set of formulas constructed using symbols and variables. The simplest kind of formulas are called *molecules*.

Definition 1 (Molecule)
An molecule in F-Logic is one of the following statements:

1. *An* is-a *assertion of the form* $C : D$ *where* $C, D \in \sigma \cup \mathcal{V}$*;* $\mathcal{V}$ *is an infinite set of variables.*
2. *A* subclass-of *assertion of the form* $C :: D$ *where* $C, D \in \sigma \cup \mathcal{V}$
3. *Data molecules of the form* $C[D \ \rightarrow E]$ *where* $C, D, E \in \sigma \cup \mathcal{V}$

An F-Logic molecule is called ground, if it contains no variables.

An F-Logic formula is defined by combining molecules with the usual logic connectives $\wedge, \vee, \leftarrow, \leftrightarrow, \neg$ and quantifiers $\forall, \exists$.

F-logic has a model-theoretic semantics and a sound and complete resolution-based proof theory [Kifer et al., 1995]. This semantics is more complex than the predicate logic semantics.

There are several implementations of F-Logic, the most known are FLORA-2 [Guizhen Yang and Zhao, 2003] and Ontobroker [Decker et al., 1999]. All implementations have in common the way to represent the higher-order constructs in Horn-Logic rules.

3.5 OWL

OWL, which stands for Ontology Web Language, is actually the standard language for representing knowledge on the Web. This language was designed to be used by applications that need to process the content of information on the Web instead of just presenting information to humans users [McGuinness and van Harmelen, 2004]. Using OWL one can explicitly represent the meaning of terms in vocabularies and the relationships between those terms. In a Semantic Web stack [Berners-Lee, 2003] OWL is part of the ontology language layer and is place above RDF and XML layers. So far OWL is the most expressive knowledge representation for the Semantic Web. Using OWL, one can express much richer relationships than RDF(S) thus yielding the benefit of much enhanced inference capability.

OWL is based on Description Logics and has its origin in DAML+OIL ontology language [Horrocks and van Harmelen, 2001], incorporating lessons learned from the design and application of DAML+OIL. DAML+OIL is a combination of OIL [Fensel et al., 2001] and DAML-ONTO [McGuinness, 2003] ontology languages.

When designing a language there is always a trade-off between expressivity of the language and tractability of inference that can be done using that language. With OWL, this trade-off has been partly addressed through a layered approach. There are three OWL sub-languages: OWL Lite, OWL DL and OWL Full. Starting from the lower layer, OWL Lite, up to the upper layer, OWL Full, each sub-language is enriched in syntax and semantic.

OWL Full is layered syntactically and semantically on top of RDF(S) providing maximum expressivity, but with no guarantee of computational behavior. OWL DL restricts OWL Full to that part which offers completeness (a guarantee that an answer will can be found) and decidability (a guarantee that the computation will end in a finite time). OWL Lite restrict OWL DL further to a language that can be used to express classification hierarchy and simple constrains. OWL Lite and can be seen as a variant of the $\mathcal{SHIF}(\mathbf{D})$ description logic language, whereas OWL DL is a variant of the $\mathcal{SHOIN}(\mathbf{D})$ language [Horrocks and Patel-Schneider, 2003]. The $\mathcal{SHIF}(\mathbf{D})$ language allows complex class

descriptions, including conjunction, disjunction, negation, existential and universal value restrictions for roles, role hierarchies, transitive roles, inverse roles, a restricted form of cardinality restrictions (cardinality 0 or 1) and (limited) support for concrete domains. $\mathcal{SHOIN}(\mathbf{D})$ adds support for individuals in class descriptions and arbitrary cardinality constraints.

There are two major syntaxes for OWL. The most prominent one is based on RDF/XML syntax [Beckett, 2003]. The second syntax, called abstract syntax [Patel-Schneider et al., 2004] was developed for OWL DL and is similar to the syntax used in many Description Logics reasoners. The abstract syntax can only be used for the DL dialect of OWL. The RDF/XML syntax is more complex that the abstract syntax and not so intuitive to use.

The semantic of OWL assigns meaning to all variants of OWL. OWL has a direct model-theoretic semantics [Patel-Schneider et al., 2003] which is based on DAML+OIL model-theoretic semantics. There are two ways of describing the semantic: as an extension of RDF(S) model theory and as a direct model-theoretic semantics of OWL.

3.6 SWF Languages for Representing Knowledge about Static World

This section shortly presents the main features of previous described languages, the differences between them and their shortcomings for representing knowledge about a static world in Semantic Web and Semantic Web Services. In the last part of this section WSMO languages for representing knowledge about a static world are introduced and the rationale for choosing WSMO Languages for this task within SWF is presented.

3.6.1 Languages for Representing Knowledge about a static World: Differences and Drawbacks

Description Logics is one of the most prominent logical formalisms for representing knowledge. Its power consists in its *reasoning support*. Description Logic clearly distinguish between classes and individual. F-Logic on the other hand make no clear distinction between classes and individuals. One important feature that distinguish F-Logic for the other languages for representing knowledge about static world is the *object oriented support and frame based approach* for describing application domains and reasoning about instances, concepts and their relations. Another language that offers very good computation advantage is Logic Programming. With Logic Programming knowledge about static aspects of the world can be represented in a intuitive and powerful *rule-base style*. The benefits of Description Logic and Logic Programming are combined in Description Logic Programs. Being the intersection of this two languages Description Logic Programs offers a way *to build rules on top of ontologies*.

All previous mentioned languages were not specifically designed to represent knowledge on the Web. On the other hand OWL was designed especially with this purpose in mind. A closer look at OWL reveals some drawbacks of this language. We enlist them without detailed explanations. For detailed explanations we refer the reader to [de Bruijn et al., 2004b].

1. **ABox Reasoning in OWL is hard**.
   The main reason is that reasoning with large collections of instances and reasoning with equality is hard.

2. **Deriving Equality in OWL is non-intuitive**.
   The main reason is that when two instances of the functional property

have the same domain, but a different range value, the two instances in the range are consider equals according to the semantics of OWL.

3. **Lack of a notion of constrains in OWL**.
   OWL offers value and cardinality restrictions, that are different from constrains. The former are used to infer new information about individuals, while the latter are used to detect inconsistencies.

4. **Rule extension of OWL is not straightforward**. Adding rules support to OWL make this language undecidable.

5. **Inappropriate layering**.
   As was mentioned in the section 3.5, OWL Full lays on top of RDFS and OWL-DL. Because RDFS and OWL-DL are different syntactically and semantically, the OWL-Full syntax and semantic is not a clean one.

6. **Limited support for Datatypes**.
   OWL dose not support negated datatypes, datatypes predicates and user-defined datatypes.

3.6.2 Extensions of Languages for Representing Knowledge about a static World: OWL$^-$

The previous mention shortcomings are tackled in a new language proposal called OWL$^-$ [de Bruijn et al., 2004b]. OWL$^-$ is a restriction of OWL that can be translated to Datalog, which itself is a subset of Horn logic. OWL$^-$ address the problem of inappropriate layering of OWL. OWL$^-$ is based on Description Logic Programs (Section 3.3) and like OWL follows a layering approach. There three dialects of OWL$^-$ are: OWL Lite$^-$, OWL DL$^-$ and OWL Full$^-$.

1. **OWL Lite$^-$**
   OWL Lite$^-$ is a subset of OWL Lite, which can be translated to Datalog. OWL Lite$^-$ does not have some of the features available in OWL like: `owl:Thing` and `owl:Nothing`, (inverse) functional properties, (in)equality assertions of individuals, existential value restrictions, universal value restrictions in complete class definitions, minimal cardinality restrictions and maximal cardinality restrictions, but as is pointed in [de Bruijn et al., 2004b] these features are not intuitive in their usage and actually add computational complexity to OWL.

   OWL Lite$^-$ is based on $\mathcal{L}_0$ which is the intersection of $\mathcal{SHOIN}$ description language and Datalog. This offers OWL Lite$^-$ a very good support for *query answering* inherited from Datalog and the power of a Description Logic langauge like $\mathcal{SHOIN}$.

2. **OWL DL$^-$**
   OWL DL$^-$ extends OWL Lite$^-$ by introducing the `value` value restriction and by making several hidden feature explicit. The features OWL DL$^-$ that are not present in OWL Lite$^-$ are enlisted below:

   - OWL DL$^-$ allows the `intersectionOf` description in the place of name classes.
   - OWL DL$^-$ allows the `value` restriction in both partial and complete class definitions.
   - OWL DL$^-$ allows the `SubClassOf` construct which explicitly specifies a subsumption relation between two descriptions.

- OWL DL$^-$ adds the `unionOf` construct which may occur as the first argument of `SubClassOf`.
- OWL DL$^-$ allows the `someValuesFrom` modifier which occurs as the first argument of `SubClassOf`.

The OWL DL$^-$ is actually a subset of OWL DL and in consequence has a model-theoretic semantics. This semantics correspond to OWL DL semantics with the restriction of the abstract syntax to the OWL DL$^-$ abstract syntax.

3. **OWL Full$^-$**
OWL Full$^-$ extends OWL DL$^-$ with a meta-class facility, based on F-Logic [Kifer et al., 1995]. In F-Logic, as mentioned in Section 3.4, there is not clear distinction between classes and instances. The F-Logic based approach of OWL Full$^-$ determine a non separation between classes and instances, which is different from OWL DL$^-$ where this separation is very clear. OWL Full$^-$ stays in Horn fragment of First-order logic.

   OWL Full$^-$ syntax is define on top of OWL DL$^-$ syntax. This is not the abstract syntax of OWL Full, because it allows arbitrary use of RDF triples, which cannot be captured in the OWL abstract syntax. OWL Full$^-$ imposes some restrictions on the use of RDF triples in order be able to extend the OWL DL$^-$ abstract syntax. OWL Full$^-$ has three major limitations:

   - lack of a notion of value constraints
   - lack of cardinality constraints
   - lack of negation.

   OWL Full$^-$ has a more appropriate layering on top of RDFS. This is done by defining an alternative semantics for RDFS based on F-Logic translation defined in [de Bruijn et al., 2004b].

We conclude by summarizing the most important features of OWL$^-$:

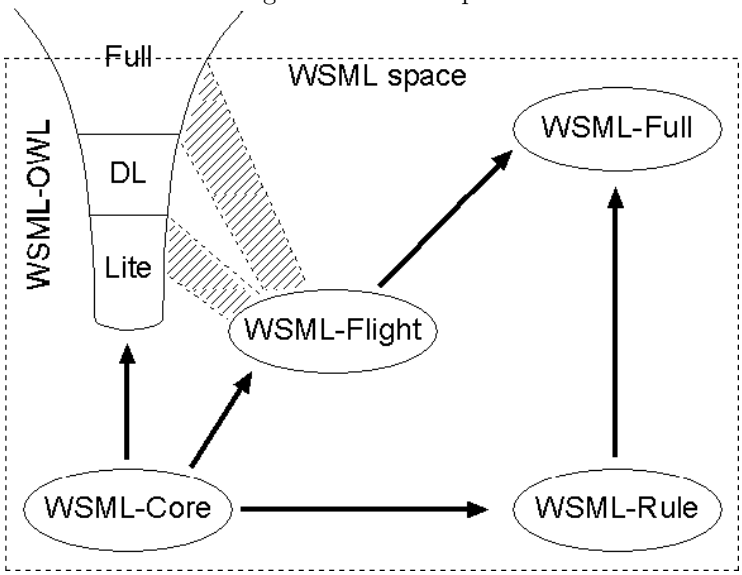
- efficient reasoning over instances.
- direct translation into Description Logics and Logic Programming.
- several non-intuitive features of OWL (e.g. derivation of equality are removed).
- extensions in both the DL and LP direction are straightforward.
- the Full species is properly layered.

3.6.3 WSMO Languages

A complete list of WSMO languages that can be used for representing knowledge about a static World can be found in [Oren et al., 2004]. In Figure 3 the different WSML Languages and the relation between them are presented. The basis for all WSML languages is WSML-Core. This langauge, described in more detail in the end of this section can be extended in WSML Space in two directions. One direction is to extend it by rules support. In this way a new language, called WSML-Rule is define. The second direction is to extend WSML-Core with support for Web Ontology Langauge OWL. The new language called WSML-DL follows the layering approach of OWL, distinguishing

the Lite, DL and Full species. Another language in the WSML Space is WSML-Flight. This language is an extension of WSML-Core with several feature from OWL Full and constrains. By unifying WSML-Flight and WSML-Rule a new language was defined: WSML-Full. Our focus, in the context of SWF, is on WSML-Core language that has an attractive computational characteristic and though less expressive, it provides enough expressivity for SWF needs.

Figure 3: WSML Space



WSMO-Core The WSML working group[7] have created a new ontology modeling language called WSML-Core [de Bruijn et al., 2004a] which is based on semantic of OWL^- and the conceptual model of WSMO[8]. This language has the least expressive power, but on the other hand it has the most powerful computational characteristics. As was mention before, WSML-Core was created with the WSMO conceptual model in mind. The modelling elements used in WSML-Core are:

1. *Ontologies*
   Ontologies in WSML-Core are simply defined by using the **ontology** keyword.

2. *Namespace*
   Namespaces in WSML-Core are introduced using the **namespace** keyword optionally followed by a list by the default namespace and a number of namespace definitions. Namespaces provides web compatibility by defining a space of coherent vocabulary.

3. *Non-functional properties*
   Non-functional properties are introduced using the **nonFunctionalProperties** keyword followed by a list of properties and property values. They are used to describe the non-functional aspects of the item.

4. *Import Ontology*
   Following WSMO modular approach for ontologies design, ontologies can

[7] http://www.wsml.org
[8] http://www.wsmo.org

be imported(imported ontologies). This can be done in WSML-Core using **importOntologies** keyword followed by a list of URIs identifying the ontologies being imported.

5. *Mediators*
   Mediators are connectors between ontologies with mediation facilities for handling heterogeneities. The (optional) mediation block is identified by the **usedMediators** keyword, followed by one or more identifiers of WSMO mediators.

6. *Concepts*
   A concept definition in WSML-Core starts with the **concept** keyword followed by the identifier of the concept.

7. *Relations*
   Relations are used to express relationship between concepts in the domain. In WSML-Core a relation definition starts with the **relation** keyword followed by the identifier of the relation and an optional specification of super-relation through the **subRelationOf** keyword.

8. *Instances*
   Instances represents specific members of the concepts. In WSML-Core a instance definition starts with the **instance** keyword followed by the identifer of the instance, the **memberOf** keyword and the name of the concepts of which the instance is a member.

9. *Axiom*
   Axioms represent model sentences that are always true. In WSML-Core an axiom definition starts with the axiom keyword, followed by the name of the axiom, the optional **nonFunctionalProperties** block and then by the logical expression preceded with the **definedBy** keyword.

10. *Datatypes*
    WSML-Core allow user-defined datatypes predicates and user-defined datatypes. A datatype expressions starts with the **function** keyword followed by the name of the function then by an optional **nonFunctionalProperties** block. This syntactic construct is followed by an optional range and the then by an optional set of parameters and a datatype expression preceded by the **definedBy** keyword.

The logical expressions provided in WSML-Core are:

1. **Simple Logical Expression**
   (a) *Relational Expressions*
   A Relational Expression consists of a predicate identifier followed by the comma-separated arguments of the predicate, enclosed by parentheses.
   (b) *Molecules* Molecules in WSML-Core have one of the following type:
   - Concept molecule
     A concept molecule is either a concept membership assertion of the form :/ **memberOf C**, either an attribute value list of the form :/[A_1 **hasValues** v_1,...,A_n **hasValues** v_n].
   - Instance molecule
     A concept molecule is either a subconcept assertion of the form C **subConceptOf** D, either an attribute definition list of the form C[A_1 **ofType** D_1,...,A_n **ofType** D_n].

(c) *Datatype predicates*
A datatype predicate consists of a predicate identifier and parenthesis-delimited, comma-separated list of arguments. The number of arguments depends on the arity of the predicate and all the arguments must be data values.

2. **Complex Logical Expressions**

(a) *Compound Logical Expressions*
A compound logical expression consists of several molecules and/or (datatypes) predicates, separated by the **and** keyword.

(b) *Formulas*
A formula contains two logical expressions interconnected by an implication symbol.

The syntax of WSML-Core is basically a slight restriction of WSML. Because is based on semantics of OWL Lite$^-$ some extensions were added to cover epistemology of OWL Lite- (e.g. transitive, symmetric, inverse properties). The semantics of WSML-Core is defined through a translation to OWL Lite$^-$ Abstract Syntax, and falls in intersection of Description Logic and Datalog.

WSML-Core represent the most suitable langauge for representing knowledge about a static world in SWF project. The rationale for this choice is the high computational characteristic of this language, characteristic required within SWF. The base of WSML-Core is the intersection of Description Logics and Datalog, which offers to this language. WSML-Core allows compatibility with Descriptions Logic and also easy rules extensions. WSML-Core is part of WSMO effort therefore the description language for representing knowledge about a static world in SWF and the mechanisms that work upon this language are aligned with WSMO.

4 Dynamics and Interaction Protocols

The purpose of this section is to evaluate the representational formalism for dynamic and interaction protocols used within SWF. This will be provided in the next version of the deliverable.

5 Conclusions

In this deliverable we have evaluated the languages for describing, representing, and handling different kinds of technological constructs that are relevant in the Semantic Web Fred project. As a framework for semantic web services SWF uses WSMO, by apply description elements and technologies developed within WSMO. The rationale of this choice has been provided in Chapter 2. For the static knowledge representation, the WSML languages, more precisely WSML-Core represent the more complete and appropriate choice, as it was explained in Chapter 3. A second, updated version of this deliverable will contain an evaluation of the representational formalisms for dynamic and interaction protocols used within SWF.

References

[Abiteboul and Kanellakis, 1989] Abiteboul, S. and Kanellakis, P. C. (1989). Object identity as a query language primitive. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 159–173, Portland, Oregon, USA.

[Baader et al., 2003] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The Description Logic Handbook*. Cambridge University Press.

[Bechhofer et al., 2002] Bechhofer, S., Horrocks, I., and Turi, D. (2002). Instance store - database support for reasoning over individuals. Available from http://instancestore.man.ac.uk/instancestore.pdf.

[Beckett, 2003] Beckett, D. (2003). RDF/XML syntax specification (revised). Recommendation 10 February 2004, W3C.

[Berners-Lee, 2003] Berners-Lee, T. (2003). Web services - semantic web: Semantic web stack. http://www.w3.org/2003/Talks/0521-www-keynote-tbl/.

[Calvanese et al., 1998] Calvanese, D., Giancomo, G. D., and Lenzerini, M. (1998). On the decidability of query containment under constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 149–158.

[Curbera et al., 2002] Curbera, F., Goland, Y., Klein, J., Leymann, F., Roller, D., Thatte, S., and Weerawarana, S. (2002). Business process execution language for web services. Technical report, BEA Systems & IBM Coporation & Microsoft Corporation. Available from http://www.ibm.com/developerworks/library/ws-bpel/.

[de Bruijn et al., 2004a] de Bruijn, J., Foxvog, D., Oren, E., and Fensel, D. (2004a). WSML-Core. Working Draft D16.7v0.1, WSMO. Available from http://www.wsmo.org/2004/d16/d16.7/v0.1/.

[de Bruijn et al., 2004b] de Bruijn, J., Polleres, A., Lara, R., and Fensel, D. (2004b). OWL$^-$. Working Draft D20.1v0.2, WSMO. Available from http://www.wsmo.org/2004/d20/d20.1/v0.2/.

[Decker et al., 1999] Decker, S., Erdmann, M., Fensel, D., and Studer, R. (1999). *Ontobroker: Ontology based Access to Distributed and Semi-Structured Information*. Kluwer Academic.

[Fensel and Bussler, 2002] Fensel, D. and Bussler, C. (2002). The web service modeling framework WSMF. *Electronic Commerce Research and Applications*, 1(2):113–137.

[Fensel et al., 2001] Fensel, D., van Harmelen, F., Horrocks, I., McGuinness, D. L., and Patel-Schneider, P. (2001). OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2).

[Grosof et al., 2003] Grosof, B., Horrocks, I., Volz, R., and Decker., S. (2003). Description logic programs: Combining logic programs with description logic. *World Wide Web Conference, Budapest*.

[Guizhen Yang and Zhao, 2003] Guizhen Yang, M. K. and Zhao, C. (2003). FLORA-2: A rule-based knowledge representation and inference infrastructure for the semantic web. In *Proceedings of the Second International Conference on*

Ontologies, Databases and Applications of Semantics (ODBASE), Catania, Sicily, Italy.

[Horrocks and Patel-Schneider, 2003] Horrocks, I. and Patel-Schneider, P. F. (2003). Reducing OWL entailment to description logic satisfiability. In *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, Sanibel Island, Florida.

[Horrocks and van Harmelen, 2001] Horrocks, I. and van Harmelen, F. (2001). Reference description of the daml+oil (march 2001) ontology markup language. Technical report, DAML. http://www.daml.org/2001/03/reference.html.

[Kifer et al., 1995] Kifer, M., Lausen, G., and Wu, J. (1995). Logical foundations of object-oriented and frame-based languages. *JACM*, 42(4):741–843.

[Lara et al., 2004] Lara, R., Polleres, A., Lausen, H., Roman, D., de Bruijn, J., and Fensel, D. (2004). A comparison of WSMO and OWL-S.

[McGuinness, 2003] McGuinness, D. L. (2003). Ontologies come of age. In Fensel, D., Hendler, J., Lieberman, H., and Wahlster, W., editors, *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*, chapter 6, pages 171–194. MIT Press.

[McGuinness and van Harmelen, 2004] McGuinness, D. L. and van Harmelen, F. (2004). Owl web ontology language overview (february 2004). Technical report, W3C. http://www.w3.org/TR/owl-features/.

[Oren et al., 2004] Oren, E., Lausen, H., and de Bruijn, J. (2004). Languages for WSMO. Working Draft D16.0v0.2, WSMO. Available from http://www.wsmo.org/2004/d16/d16.0/v0.2/.

[Patel-Schneider et al., 2004] Patel-Schneider, P. F., Hayes, P., and Horrocks, I. (2004). OWL web ontology language semantics and abstract syntax. Recommendation 10 February 2004, W3C.

[Roman et al., 2004] Roman, D., Lausen, H., and Keller, U. (2004). Web service modeling ontology standard (WSMO-standard). Working Draft D2v1.0, WSMO. Available from http://www.wsmo.org/2004/d2/v1.0/.

[The OWL Services Coalition, 2004] The OWL Services Coalition (2004). OWL-S 1.1 beta release. Available at http://www.daml.org/services/owl-s/1.1B/.

[Volz, 2004] Volz, R. (2004). *Web Ontology Reasoning with Logic Databases*. PhD thesis, AIFB, Karlsruhe.

[Weibel et al., 1998] Weibel, S., Kunze, J., Lagoze, C., and Wolf, M. (1998). Dublin core metadata for resource discovery.