Survey on semantic data stores and reasoning engines

VIP Project - ANR-09-COSI-03 - Milestone 1.2.1

Alban Gaignard, alban.gaignard@i3s.unice.fr
Johan Montagnat, johan@i3s.unice.fr

Version 1.1, September 14, 2010

Revision table

version 1.1	September 14, 2010
version 1.0	July 30, 2010

Abstract

This document aims at reviewing semantic technologies and their applications in the context of the VIP project. The first part focuses on background information describing the Semantics area and its connections to the Semantic Web and to the Knowledge Engineering area. Then, in section 2, some of the most popular semantic data stores are reviewed followed by popular semantic reasoning engines in section 3. Section 5 describes the main approaches aimed at bridging Web Services to the Semantic Web. Finally, in section 6, we focus on the VIP project use cases and highlight the more suitable approaches.

Contents

1 Background information on Semantics

1.1 Semantic data

1.1.1 The Semantic Web as a foundational technological layer

Representing the semantics of data is a need that has recently emerged from the Semantic Web area which intends to bring meaning to the unprecedented and tremendous amount of data published over the Web.

Tim Berners-Lee presents the *Semantic Web* [8] as *"not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation"*. He also conditions its success to automated reasoning: *"computers must have access to structured collections of information and sets of inference rules that they can use to conduct automated reasoning"*. The Semantic Web is based on *Ontologies* - languages to formally describe domain specific concepts and their relations - coupled to reasoning engines, to perform domain fact deductions.

Representing the semantics of data generally consists in associating meaningful and structured metadata (*i.e.* descriptive information) to raw data. Such metadata (i) classifies raw data through well-defined concepts and (ii) link raw data together through well-defined relationships. The main objective is that metadata can be automatically processed in order, for instance, to help users in retrieving relevant information from a jungle of resources[1] spread over the Internet. But some technical means are needed to address (i) the unambiguous and unique identification of resources over the Internet, (ii) the description and linking of these resources, and (iii) the conceptualization of a domain to formally express the meaning of the resources (*i.e.* the setup of an ontology).

A lot of standardization efforts have been made through the W3C and led to the definition of three frameworks that answer the above mentioned needs, namely URI (Universal Resource Identifier), RDF (Resource Description Framework), and OWL (Web Ontology Language) which constitute the main bricks of the Semantic Web, and by extension provide a strong support to the Knowledge Engineering area.

1.1.2 Linking data

In order to share information, URIs are the *de facto* standard to unambiguously identify information. No particular assumptions are made on the resources identified by a URI. They might be abstract or concrete, and are potentially not accessible over the Internet. Standard URLs are specific URIs used to locate resources over a network, and thus URLs syntax can be used to define URIs. For instance "Saint-Malo" is an ambiguous name for a city and might refer to either a french city or a canadian city. We are able to uniquely identify "Saint-Malo" with two URIs such as "http://canada/quebec/saint-malo" and "http://france/bretagne/saint-malo".

As soon as resources are well identified through URIs, RDF can be used to attach descriptions to these resources and link them together. The main idea of RDF is to describe the semantics of the data by expressing simple statements of the form *Subject – Predicate – Object* which can also be considered as simple sentences involving a subject, a verb, and a complement. Note that when several statements are grouped together, the objects of some statements may also act as a subject of another statements which lead to a graph-like representation. Indeed, RDF statements can be considered as a directed labelled graph where *Subjects* and *Objects* define the nodes and *Predicates* define the labelled and directed edges.

Let us consider an example from the neuro-imaging domain where a medical image has been acquired from a scanner and is processed to extract brain tissues.

Figure 1 represents an RDF graph linking neuro-imaging resources identified by their URIs. It first states that a medical image has been acquired from a scanner, then, that this image has been processed by a brain extraction tool, and finally, that brain tissues have been extracted from the "Bet" brain extraction tool.

1.1.3 Classifying data

So far, we have seen how RDF can be used to attach descriptive information to data and to establish relationship between data. But another aim of RDF is also to allow user communities to define controlled vocabularies (or terminologies) by describing the domain specific concepts and how they are related to each others. RDFS, which stands for "RDF Schema", is an extension of RDF[2] and provides a language to define such vocabularies. Two main constructs are provided and aim at describing *Classes* and *Properties*.

[1]typically data or services

[2]as "XML Schema" extends XML to support the definition of complex types

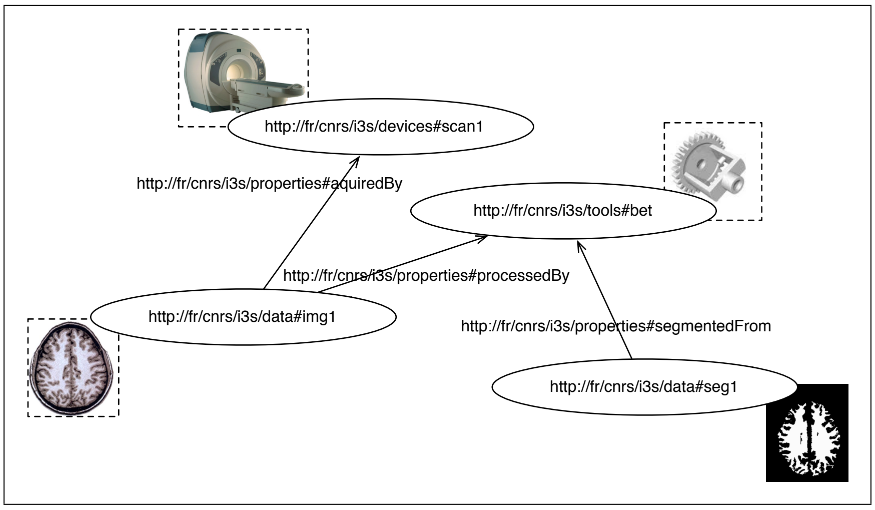
http://fr/cnrs/i3s/devices#scan1
http://fr/cnrs/i3s/properties#aquiredBy
http://fr/cnrs/i3s/tools#bet
http://fr/cnrs/i3s/properties#processedBy
http://fr/cnrs/i3s/data#img1
http://fr/cnrs/i3s/properties#segmentedFrom
http://fr/cnrs/i3s/data#seg1

Figure 1: Linking raw and processed data to the equipment and processing tool

Classes are used to group resources together. Resources that belong to a particular class are called *instances* or *individuals* of that class. Hierarchies of classes can be constructed through the definition of sub-classes.

Properties are used to define relationships between classes, and in the same way, it is also possible to define hierarchies of properties through sub-properties. Properties are characterized by (i) their *domain* – the set of classes from which the property can be attached, in other world its applicability domain – and by (ii) their *range* – the set of classes to which the property can be attached.

The following example shows a simple controlled vocabulary covering the previous example (figure 1).

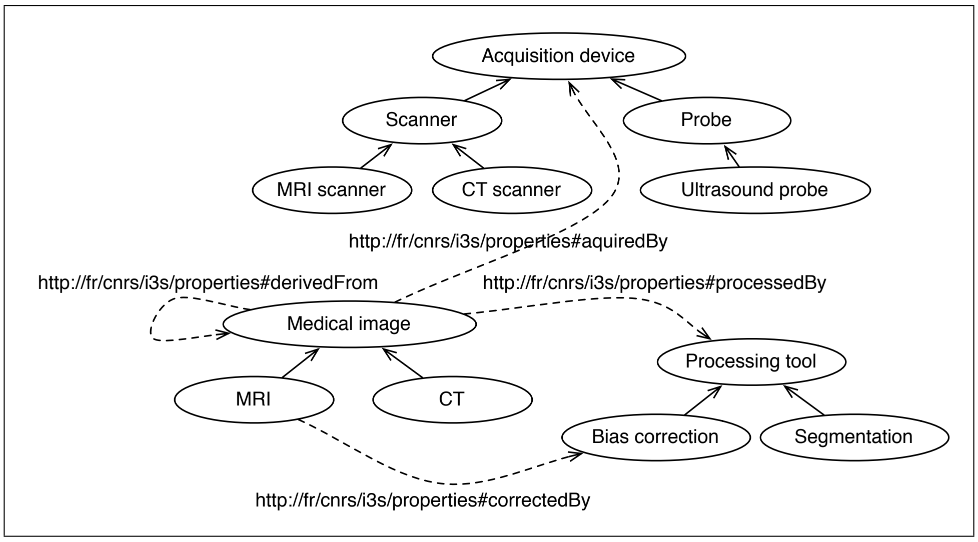
Acquisition device
Scanner
Probe
MRI scanner
CT scanner
Ultrasound probe
http://fr/cnrs/i3s/properties#aquiredBy
http://fr/cnrs/i3s/properties#derivedFrom
http://fr/cnrs/i3s/properties#processedBy
Medical image
Processing tool
MRI
CT
Bias correction
Segmentation
http://fr/cnrs/i3s/properties#correctedBy

Figure 2: A sample controlled vocabulary expressed in RDFS

Figure 2 illustrates how RDFS can be used to define the concepts of a domain and their relationships. The rounded boxes represent the domain classes. Note that they are also identified through URIs but for readability, we only show their label. Plain arrows represent *rdfs:subClassOf* properties and constitute the several hierarchies of concepts. Dashed arrows represent domain-specific properties. For instance *processedBy* and *correctedBy* differ by their *rdfs:domain* and *rdfs:range*. More precisely *correctedBy* can only be applied to MRI images and bias correction tools, whereas *processedBy* is applicable for any medical image and any processing tool. In addition we can state in RDFS than *correctedBy* is a sub property of *processedBy*. In the remainder of the document we will refer to both RDF or RDFS standards by the RDF(S) notation.

But sometimes, for some user communities, these means are not sufficient to precisely model their domain, and they usually develop a domain ontology. An ontology can be defined as a set of *class axioms*, *property axioms*, and *facts about individuals* (or *instances*). While *axioms* aim at formally defining the semantics of classes or properties, *facts* are assertions (or statements) describing the instances. Note that *axioms* are used to place constraints on classes and on their relationships (properties). The OWL language [25], is a standard framework to build ontologies. OWL ontologies are written and stored in RDF syntax so that they really appear as an extension of the existing RDF(S) stack. OWL brings new constructs, through *axioms* and allows richer exploitation through the possible entailments operationalized by reasoning engines. This document does not detail the whole extensions, with regards to the RDF(S) language, but rather illustrates some of them through the previous examples.

- *TransitiveProperty*: the "derivedFrom" property could be stated as a transitive property so that it can be automatically inferred that if A "derivedFrom" B and B "derivedFrom" C then A "derivedFrom" C.
- *SymmetricProperty*: we could also imagine a *consume* property starting from the processing tool class and directed to the medical image class, which is the symmetric of the *processedBy* property. This would be useful, if we consider a set of instance related by only one of these two properties, to generate the symmetric property with an inference engine, and thus to complete the set of relationships.
- *disjointWith*: this construction helps in validating the nature of a set of instances. For instance we could state that MRI and CT are disjoint classes so that when validating a set of instances, the system can check that none of the instances belong to both MRI and CT classes (similar to an exclusive disjunction, also known as the XOR logical operator).
- *unionOf*: OWL classes, generally stated as "complex classes", can be defined through axioms representing basic set operators such as union, intersection or complement. We can easily imagine that the *Acquisition device* class can be defined as the union of the two classes *Scanner* and *Probe*, so that an owl reasoner infer that all instances of *Scanner* and *Probe* classes also belong to the *Acquisition device* class.
- *cardinality*: another interesting feature is the ability to define some constraints, for instance through property restrictions such as cardinality. We could imagine that a medical image (*i.e.* an instance of the *Medical image* class) cannot be acquired from two different devices. This restriction can be stated by the *maxCardinality* set to 1 for the *acquiredBy* property.

OWL constructs allow richer description of domain-specific knowledge but the counterpart is that possible deductions through reasoners become more complex and time consuming. More insights will be given on the possible entailments and the impact on their calculation in sections 1.2.1 and 1.2.3.

The following table summarizes the different level of knowledge covered by RDF, RDFS and OWL languages:

Knowledge on concepts	RDFS classes	OWL classes
Knowledge on individuals	RDF data	OWL instances

We have seen that the two standardized languages of the Semantic Web, RDF(S) and OWL, are the cornerstones in the process of building semantic applications either for storing knowledge about the domain-specific concepts or for the data itself. OWL being designed upon the existing RDF stack of the W3C, OWL ontologies are expressed using RDF triples. It seems thus natural to lean on RDF tools to express semantic data, either considering simple conceptualizations through RDF(S) or complex conceptualizations through OWL ontologies.

1.2 Semantic reasoning

1.2.1 Description Logics

In few words, Description Logics (DL) [52, 4] are used to provide a logical formalism for Ontologies and Semantic Web by modeling *concepts*[3], *roles*[4], *individuals* and their relationships. More generally, DL are used as a foundation for building *Knowledge Bases* from two main components, the terminological knowledge box (TBox) and the assertional knowledge box (ABox). The TBox refers to statements regarding the terminology itself. It describes the hierarchy of concepts, and, the relationships between concepts, whereas the ABox refers to statements regarding the belonging of individuals to concepts, and the relationships between individuals. In description logics, both TBox and ABox constitute a knowledge base. While the TBox tend to be more static in the sense that the definition of concepts is rather fixed, the ABox tend to be more dynamic in the sense that the population of individuals is changing and subject to extension through DL reasoning.

What makes DL sound "exotic" for classical computer system users or programmers, is that DL are based on the "open world" assumptions. Classical systems are based on "closed world" assumptions which means that if a fact is not known to be true, it is considered as *false*. On the contrary, "open world" assumptions mean that, ABox being considered as incomplete, if a fact is not known to be true (in ABox), it does not mean that the fact is *false*, and it becomes potentially *true*. It becomes then much more complex to determine if something is *true*.

The web ontology language OWL [30, 24] have a lot of similarities with DL knowledge bases. Indeed, OWL *axioms* can be considered as statements of the TBox while OWL *facts* can be considered as statements of the ABox. Finally, an OWL ontology, encompassing axioms and facts, can be bound to a DL Knowledge Base.

Being much more expressive than RDF(S) languages and allowing richer inferences, OWL lead to computational issues such as decidability (related to the "open world" assumption). Three variant of OWL are thus proposed to tackle these computational issues:

[3]which abstract *classes* in RDFS or OWL
[4]which abstract, with their relationships, *properties* in RDFS or OWL

(i) *OWL-DL* which has a readable DL-like syntax allows for decidable inferences, (ii) *OWL-Lite* a subset of *OWL-DL* with an even more simpler syntax with tractable inferences, and finally (iii) *OWL-Full*, a syntactic and semantic extension of RDF(S) without any usage restriction. Regarding the computational complexity of these variants, *OWL-Full* is undecidable, *OWL-DL* is non-deterministic and exponential time complexity, and finally *OWL-Lite* is more tractable, with a deterministic and exponential time complexity.

1.2.2 Conceptual graphs

Conceptual graphs are another formalism introduced by John Sowa [66] to capture the underlying meaning of data, to link them together and to identify patterns between them [56]. Conceptial graphs (CG) are appealing because their formalism is logically precise, their representation is human-readable, and finally they are computationally tractable. RDF graphs can easily be bound to Conceptual Graphs as they share a similar principle: concept nodes are connected to relation nodes and form together a labelled graph.

Projection is the key operation providing reasoning capabilities in CG. It consists in calculating a specialization/generalization relation between two graphs. It is generally stated that a "general" graphs "projects" into a more "specialized" graph. The projection operation lead to the identification of patterns in data which is particularly suited for Information Retrieval issues.

1.2.3 Reasoning tasks

As presented in section 1.1.1, one of the objectives of the Semantic Web (and by extension, the Knowledge Engineering area) is to lean on reasoning engines to exploit semantically annotated data and therefore deduce new knowledge specific to a domain covered by the underlying ontology. Based on different constructs and level of expressivity, the languages of the Semantic Web allow several kinds of deductions with different complexities and calculation costs.

Inherited from the Description Logics area, reasoning tasks target the meaningful exploitation of knowledge bases by means of validation or deduction of new knowledge. Reasoning tasks are generally split between the TBox and the ABox as some are related to perform inferences or validation at the conceptual level, or other ones are more related to validate instances. Some of the main reasoning tasks are briefly introduced below and illustrated through the TBox graphically defined in figure 2.

Reasoning tasks regarding the TBox of a knowledge base:

- *Subsumption*: is a key reasoning task which consists in determining if a concept is more general than another one. For instance, the class "Acquisition device" *subsumes* the class "Scanner" as a scanner is a specialization of an acquisition device.
- *Classification*: consists in determining the appropriate place of a newly defined concept into a hierarchy of concepts linked together by subsumption relations. For instance if we define a "Medical device" as the union of "Probe", "Scanner", this class should appear as a sub-class of "Acquisition device" and as a super-class of both "Probe" and "Scanner".
- *Logical implication*: consists in checking if relationships between concepts are logical consequences of the assertions of the TBox.
- *Inference*: is a reasoning task related to the extension of relationships in the TBox, and consists in propagating transitivity, symmetry, functionality or inversibility of relationships.

- *Equivalence*: consists in testing if two classes or properties, potentially defined by different set of assertions or axioms, are the same. The OWL *sameAs* construct is used in this sense to state equivalence.
- *Satisfiability*: is a key reasoning task with the following underlying idea. A newly added concept in the TBox must make sense with regards to the TBox and does not lead to any contradictory knowledge. The logical foundation of satisfiability is that an interpretation (the set of deductible facts) of the concept must satisfy all the axioms (all the assertions) of the TBox.

Reasoning tasks regarding the ABox of a knowledge base:

- *Instance checking*: is the main reasoning task over the ABox (and also the TBox, by extension), and validates that an individual belongs to a specified concept. *Instance checking* is the foundation for the other following reasoning tasks.
- *Knowledge base consistency*: consists in verifying that each concept of the TBox admits at least one individual.
- *Realization*: which, for a given individual, find the most specific concept it belongs to. For instance, given an instance of the "Medical image" class, it is not necessary useful to qualify it with an intermediate concept in the hierarchy of medical image classes, such as MR image. It is much more useful to qualify it with the most precise concept such as its proper modality, T1-weighted for instance, taking into account that all abstract super-classes are also considered when asserting that a medical image belongs to the class "T1-weighted".
- *Retrieval*: which finds all the individuals belonging to a given concept. This reasoning task is particularly important when searching information into a knowledge base. We can easily imaging some basic queries aiming at finding all T1-weighted MRIs registered into the knowledge base.
- *Satisfiability*: the underlying idea is similar to satisfiability at the concept definition level (TBox) in the sense that a newly created instance should not lead any contradiction concerning other instances or concerning the concepts of the TBox. For instance if we consider the *acquiredBy* property of figure 2 with its cardinality constraint (maximum one device), linking an instance of a medical image with two acquisition devices would lead to an un-satisfiability issue.

Unlike OWL, RDF(S) does not implements any Description Logics constructs, but the semantic specification of RDF(S) [28] defines a full set of valid inferences with regards to an RDF graph or its underlying RDF schema. Thirteen rules define the semantics of RDFS and the can be applied to infer new RDF triples through the inferred closure of the source graph.

1.2.4 Rule-based reasoning

The use of rule engines, also known as inference engines is a general data-driven and declarative paradigm to deduce new conclusions from a set of both data and inference rules. An inference rule is made of two parts, the *antecedent* and the *consequent*, and is generally expressed through an *If* clause and a *Then* clause. Inference engines are the lower layer of reasoners as they internally use inference rules to perform reasoning tasks such as *subsumption*, *classification*, *instance checking*, etc.

The forward chaining is one of the two main strategies to perform reasoning based on inference rules. Forward chaining consists in starting from the whole available data and applying iteratively the inference rules. The produced data is then merged into the original set of data and the set of available rules is again executed to produce additional data. This process continues until a termination goal is reached.

The backward chaining is the main alternative to forward chaining when performing automated reasoning on inference rules. Unlike forward chaining which iterates over the *antecedent* of the rules to reach one of the goals (the *consequent*), backward chaining starts from the goals, and search for a rule which has an *antecedent* known to be true. If it is not the case, this *antecedent* is added to the list of goals and the engine goes for a new iteration.

Being goal-driven, the backward chaining strategy is well adapted to prove facts. On the other side, forward chaining is more considered as a data-driven approach allowing the inference engine to produce new knowledge, based on the existing data set.

The W3C proposes SWRL[29], the Semantic Web Rule Language, to combine OWL with RuleML, the Rule Markup Language. SWRL rules represent implications between the *antecedent* and the *consequent* clauses. Each clause is made of a set of atoms interpreted as a conjunction. Each atom may represent (i) the belonging of a variable to an OWL class, (ii) the existence of an OWL property between two variables, (iii) similarity or difference between two variables. A variable represents either an OWL individual or an OWL data value. As a standard language for representing semantic rules, it is generally supported by a majority of reasoners.

We have seen that forward chaining engines iteratively produce knew facts into the knowledge base through the calculation of its inferred closure. But in order to be queried or retrieved, the results of this inferred closure must be made available. Most of the time, semantic stores provide such a capability generally known as *materialization*. With the assumption that adding new facts to the knowledge base never implies that existing facts are retracted (also known as *monotonic* entailments) some tools provide a *total materialization* strategy which consists in computing the inferred closure after each modification of the knowledge base. The main benefits are the consistency of the knowledge base and efficiency of queries (no reasoning needed at query/retrieval time) but on the other hand, addition, updates or deletions of new facts are costly in terms of computing resources.

2 Semantic data stores

2.1 Semantic data persistence

RDF being the *de facto* standard for expressing semantic data, three main directions can be envisaged to permanently store RDF triples. RDF stores are generally built on top of (i) file storage, or (ii) relational database management systems (RDBMS), or (iii) native graph model storage. Via their proper API, RDF stores generally propose in-memory RDF statement management, but since it is a non-persistent storage, this capability will not be addressed in this section.

File-system back-ends might be useful for development purpose because of an easy deployment and the simplicity of storing a small set of RDF statements into a unique file. But this approach requires to manually manage RDF files and is therefore not suitable to efficiently store, load, and request a large set of RDF triples. RDF stores usually operate a native storage back-end with optimized storage and request procedures.

RDBMS back-ends appear to be a convenient and scalable approach with regards to the growing size of the RDF triples. Indeed, the RDF store can be independent of the underlying RDBMS and benefits from its replication, or clustering mechanisms. But the RDF store become also impacted by the different performances or sensitivities of these RDBMS.

But the counterpart of RDBMS back-ends become more apparent when considering reasoning on a knowledge based stored on a relational database, especially using rule-based engines. These reasoning engines dynamically generates database accesses to propagate the effects of inference rules which dramatically impact the performances of the reasoning engine.

With native storage back-ends, RDF stores are able to implement optimizations strategies to more efficiently store or retrieve RDF triples. Some of them natively adopt a graph structure which better fit the way of representing RDF statements, and thus increase the overall performances.

2.2 Semantic data querying

Following the widely adopted Semantic Web stack from the W3C, semantic data is usually represented as RDF triples. The low-level representation of RDF is XML and it could be possible to use standard XML query languages to retrieve information. Moreover, XML query languages are not suitable to represent graph structures. This fact motivates the design of specific query languages adapted to the graph structure of RDF data. RDF query languages generally adopt a SQL-like syntax to retrieve data.

SPARQL[57] (which stands for "SPARQL Protocol and RDF Query Language"), standardized through a W3Cs recommendation, is the most popular RDF query language. The queries are composed of two clauses. The first one specifies the kind of the query. The second clause, the *WHERE* clause, consists in defining triple patterns through variables to identify the target data. Queries may include conjunctions, disjunctions, or optionality. Four kinds of queries are available through the SPARQL language:

- *SELECT*: returns all or a subset of the values of variables bound in the query pattern match (the *WHERE* clause).
- *CONSTRUCT*: returns new RDF triples (defined in the *CONSTRUCT* clause) corresponding to the bound variables of the query pattern (the *WHERE* clause).
- *ASK*: returns *true* or *false* with respect to the matching of the query pattern.
- *DESCRIBE*: returns a single RDF graph describing all the resources found through the query pattern match.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX myontology: <http://fr/cnrs/i3s/myontology#>

SELECT ?image WHERE {
  ?image myOntology:acquiredBy ?device .
  ?device rdf:type myOntology:Scanner .
}
```

Figure 3: sample SPARQL query selecting all medical images acquired by a Scanner device

Figure 3 illustrates a sample SPARQL query to retrieve the URIs of all medical images acquired by a scanner acquisition device. Variables are prefixed by a question mark. The first template triple searches for *acquiredBy* properties. For each of them, the engine searches for all devices of type *Scanner*. Finally the source of the remaining *acquiredBy* property are returned as resources matching the whole *WHERE* clause.

(TODO subgraph extraction queries) [2]

2.3 Existing stores

2.3.1 Jena framework

Jena [47, 15] is a widely adopted framework covering most of the concerns related to the development of Semantic Web applications. Jena provides a unified programming environment to address the management of RDF(S) and OWL ontologies. Moreover, it provides some interesting features regarding the persistence of semantic data and the reasoning through its inference engine, or external state-of-the-art engines.

Jena provides two subsystems to persist RDF and OWL data, namely SDB and TDB. While TDB focuses on high-performance through a proper indexing and storage engine in a local filesystem, SDB is built on top of a classical relational database and benefits from non-functional properties provided by the database engine. These two components do not let the user specify or tune a schema for storing semantic data but use their own schemas to efficiently perform storage or retrieval. Different levels of caching are used to transparently let the user manage its Jena model without taking care of the persistency layer.

Jena can turn reasoning capabilities on or off. For each category of reasoning, a specification of ontology model is available. Such specifications cover ontology languages such as OWL-{Lite, DL, Full}, RDFS, or DAML+OIL[5], with different kinds of reasoners (transitive class-hierarchy inferences, or rule-based reasoners). Based on these reasoning capabilities, Jena distinguishes between the asserted model (or base model, *i.e.* without inferences) and the inferred model, mainly because depending on the application, it is not always useful to store (materialize) inferred statements which could be considered as "virtual" statements.

Some limitations of the Jena SDB persistence engine have been pointed out in [31] (TODO).

2.3.2 Joseki

Joseki is a web server to access semantic data through the SPARQL RDF query language [57] and the SPARQL protocol [23]. The SPARQL protocol consists in defining a standard web service interface for SPARQL query processors. Joseki implements both SPARQL/-Query and SPARQL/Update [61] protocols and thus provides a web service interface to query and update RDF graphs. RDF data are internally managed through the Jena framework. (TODO questions about support of multiple user connections, concurrency)

2.3.3 Owlgres

Owlgres [67] is a DL reasoner (DL-Lite family [11] of Description Logics, with its corresponding OWL2 QL Profile recommended by the W3C [50]), with a relational database backend. This engine does not show recent development activities (first and last release in May 2008). The engine is released with a simple Joseki endpoint and some scripts to load ontologies into the relational PostgreSQL database. This semantic repository has

[5]the OWL predecessor

some limitations regarding its reasoning capabilities (no transitive properties, no cardinality restrictions, no individual equality assertions) but performs queries with logarithmic time with respect to the size of the data.

2.3.4 Virtuoso

Virtuoso is a commercial[6], general-purpose, data and application container, with extensive SPARQL and RDF support [19]. It additionally provides a Jena interface. Virtuoso addresses scalability for both semantic data storage and entailment production. In its commercial distribution, Virtuoso also addresses information distribution and heterogeneity issues through the setup of a unified data model on top of its Virtual Database Engine.

With regards to the inference scalability [54], whereas common approaches consists in beforehand materializing deduced facts, Virtuoso performs as much as possible runtime and on-demand inferencing. To guarantee a response in a fixed time, incomplete results may be returned through partial evaluation of SPARQL queries. Virtuoso also provides a realtime implementation of the *owl:sameAs* property, and supports transitivity through the rewriting of transitive SPARQL subqueries. While one subquery starts from the source, the second subquery starts from the target, and the first intersection of paths produces a first partial result.

2.3.5 Allegrograph

Allegrograph is a commercial product providing a high-performance disk-based storage with standard HTTP and SPARQL interfaces. Multiple RDF triple stores, can be assembled into a virtual single store, thus enabling the federation of distributed stores. Allegrograph provides some limited but effective and practical reasoning through its RDFS++ engine. RDFS++ supports the following predicates: *rdf:type, rdfs:subClassOf, rdfs:range, rdfs:domain, rdfs:subpropertyof, owl:sameAs, owl:inverseOf, owl:TransitiveProperty,* and *owl:hasValue*. (Todo ? AllegroGraph provide dynamic materialization) This engine can be topped with a Prolog interface allowing to perform rule-based reasoning. The Prolog engine can also be used on top of RDF data to allow more complex deductions based on domain-specific reasoning.

2.3.6 Sesame

Sesame [10] is an open-source, community-supported framework to store and query RDF(S) data. It can be connected to several storage mechanisms such as standard file systems or relational databases through its SAIL API (Storage and Inference Layer). The repository can be queried through the standard SPARQL query language and provides reasoning capabilities through the support of an RDFS inferencer (also provided by SAIL). Sesame additionally provides a standard web service interface which implements the SPARQL protocol. Sesame is extensible through a plugin architecture.

In the context of the Sesame engine development, a specific query language SeRQL (Sesame RDF Query Langugage) has been proposed. According to its authors, some original features has been proposed such as the support for graph transformation, basic set operators (*union*, *intersection*, and *minus*). SPARQL and SeRQL have been developed during simultaneous initiative but now some merge effort have been achieve to benefits, in SPARQL, from features of SeRQL and conversely.

[6]with an open-source distribution

Alibaba is an extension to the Sesame RDF repository which allows to bind Java objects and classes to RDF triples and OWL classes. Alibaba also exposes them via an HTTP server. A federation feature is additionnaly provided with the set up of virtual RDF stores based on multiple distributed RDF data sources. Based on the SPARQL Protocol, the set up of such federations is therefore read-only.

2.3.7 OWLIM

OWLIM [32] is a high-performance semantic repository published as an extension of Sesame through a SAIL implementation. Two versions are available, SwiftOWLIM, the free edition and BigOWLIM, the "enterprise" edition (usable for free in a research context). Both editions provide a persistence strategy guaranteeing data preservation and consistency, and support for reasoning with OWL dialects, covering most of OWL-Horst[7], OWL-Lite, RDFS, and OWL2 RL[8]. SwiftOWLIM is more dedicated to prototyping as it loads the full content of the knowledge base in central memory and performs fast reasoning and query evaluation. On the other hand, BigOWLIM scales over billions of triples, performs query and reasoning optimizations, but still, performances remain slower than for the free edition.

2.3.8 Tupelo

The Tupelo middleware [21] aims at supporting large-scale e-Science by proposing a decentralized semantic data repository. Tupelo has been driven by general principles to enhance metadata interoperability. For instance, (i) when re-located, data and meta-data should retain its meaning (underlying model/ontology), (ii) metadata should be automatically interpreted in order to scale, and (iii) the knowledge on how data has been processed is often more valuable than the data itself.

To implement these principles, Tupelo leans on semantic web technologies for representing metadata, on content management system technics to handle data and on the open provenance model [49] to record complex processes and data provenance. Tupelo offers to categories of operations, covering both metadata management – through the assertion, the deletion, and the search of statements – and data management allowing users to read, write and delete large binary objects (BLOBs) globally identified by their URIs. Some interfaces are provided to connect to RDF triple stores (Jena, Sesame), local or remote file systems (ssh), or HTTP servers.

Distribution issues are handled through the ability of globally identifying resources across several data and metadata providers. Semantic data can be linked across repositories through different identification policies and mappings. Moreover computational inferences are available through the extension of the middleware with dedicated plugins, similarly to the Jena framework.

2.4 Stores comparison

Rohloff *et al.* [59] and Bizer & Shultz [9] evaluated the most popular RDF Stores in terms of capabilities and performances over large set of semantic data. The LUBM [26] methodology and datasets is dedicated to benchmarking RDF store and has been used to measure both load and query[9] time.

[7]a decidable rule extension to OWL

[8]an OWL2 profile dedicated to scalable reasoning (polynomial time) through a rule engine

[9]the set of queries also cover reasoning soundness (returns correct responses to a query) and completeness (returns all of the correct responses to a query)

	Storage			Reasoning					
Repositories	*In-memory*	*RDBMS*	*Native*	*RDFS*	*DL*	*Rule-based*	*Materialization*	*Performance*	*Price*
Jena	✓	✓	✓	✓	≈OWL-Lite	✓	dynamic	⊖	free
OWLGres	-	✓	-	✓	DL-Lite	-	?	⊕/simple	free
Virtuoso	✓	✓	✓	subsomption	same/equivalent		dynamic	⊕⊕	non-free
AllegroGraph	-	-	✓	RDFS++	< OWL-Lite	ext.	dynamic	?	non-free
Sesame	✓	✓	✓	✓	-	-	-	⊕	free
SwiftOWLIM	✓	✓	-	✓	≈OWL-Lite	✓	static	⊕/small	free
BigOWLIM	-	✓	-	✓	≈OWL-Lite	✓	static	⊖/large	non-free
Tupelo	ext.	ext.	ext.	-	-	-	-	?	free

Table 1: Comparison of several existing stores regarding their storage back-end and reasoning capabilities

Rohloff *et al.* chose to evaluate, among others, Sesame, {Swift,Big}OWLIM, and AllegroGraph through the LUBM benchmark. Their results confirmed that SwiftOWLIM was not designed to address large datasets, and Jena/Sesame with a MySQL backend was not suitable for handling large datasets (more than 100M triples). "Winners" are Sesame + BigOWLIM, Sesame + DAMLDB[10] and Jena + DAMLDB. The authors also noticed that Jena + DAMLDB provided faster answers to low complexity queries.

From the single client use case, Bizer & Shultz conclude that Sesame has good performances for small datasets (1 Million triples), and Virtuoso is faster for large datasets (25M to 100M triples). For simultaneous queries from multiple clients, Bizer & Shultz showed that Virtuoso outperforms its competitors, namely Sesame and Jena SDB/TDB. Regarding the overall performances, it is confirmed that Jena SDB/TDB is out-of-competition.

Table 1 summarizes the capabilities and the overall performances of popular semantic repositories. In particular, we highlight the kinds of reasoning and their limitations with regards to the OWL-* families, and their materialization strategy. Virtuoso shows good performance but has limited reasoning capabilities. Sesame + OWLIM shows a good coverage of storage and reasoning capabilities (with static materialization) with in one hand, high performances in the context of small datasets, and on the other hand a slower behavior in the context of large datasets. Performance of AllegroGraph could not be evaluated in the study of Rohloff *et al.* mainly due to a buggy early release. Finally Jena shows slow performances but is highly documented, extensible, and can be plugged to state-of-the-art DL reasoners.

3 Semantic reasoning

3.1 Existing reasoners

3.1.1 DIG Interface

The DIG interface [5] provides a high-level unified interface to access Description Logic reasoner capabilities through HTTP communication of XML messages. In addition to the standardization motivation, this initiative aims at providing a single reasoning component

[10]dedicated and optimized database for storing DAML content, predecessor of OWL content

accessible through web-based systems with a Service Oriented Architectures flavor. Some assumptions are made regarding the management of client connections. For instance, a DIG reasoner is not aware of multi-client connections and no guarantees are given concerning concurrent updates or queries on the knowledge base. Moreover connections are stateless and clients are not identified by the reasoner. The DIG interface defines two main operations, namely *Tells* and *Asks*:

- *Tells* requests provide a unified mean to realize OWL constructs over a knowledge base ;
- *Asks* requests realize Description Logic inferences and cover satisfiability, subsumption, instance checking, relationship finding within a subsumption hierarchy.

The DIG interface provides some extensibility mechanism in order to support engine-specific reasoning capabilities. The DIG interface is spread over the most known DL reasoners such as FaCT++[69], KAON2[51] Pellet[65] or Racer[27].

3.1.2 Jena Reasoners

The Jena framework comes with a complete set of reasoners: (i) an RDFS reasoner, (ii) an OWL reasoner, (iii) a transitive reasoner and finally (iv) a generic rule-based reasoner.

Three variants are available for the RDFS reasoner. The Simple reasoner only implements the transitive closure of the *subClassOf* and *subPropertyOf* relations, the entailments regarding the *range* and *domain* of properties and the implication of *subPropertyOf* and *subClassOf*. It is not complete if we consider the RDFS specification but it is considered as very useful in practice. The *Default* RDFS reasoner additionally includes the axiomatic rules and the *Full* reasoner implements almost all the RDFS axioms and closure rules.

The Jena OWL reasoner supports only the OWL-Lite language and is implemented through a rule-based engine. It is possible to perform OWL-DL reasoning by plugging an external DL reasoner through the DIG interface. More precisely, the Jena OWL reasoner applies rules to propagate OWL implications over instance data (*i.e.* the ABox of a DL knowledge box). Class reasoning, *i.e.* reasoning over the TBox of a DL knowledge base, is implemented similarly through the generation of an instance. Inferences are computed at the instance level, and class reasoning is deduced from these inferences. This OWL rule reasoner is well adapted for applications involving large set of semantic instance data with relatively simple underlying class definitions. As a superset of RDFS entailments, all RDFS constructs are supported plus a large set of OWL entailments. Three implementations are provided with different support for OWL entailments with different impacts on the cost and efficiency of reasoning, namely *full*, *mini* and *micro* implementations. It is to be noticed that the aim of the rule-based OWL-Lite reasoner is to be correct but not complete (all deduced facts are valid, but all deducible facts might not be found).

Jena also provides a "transitive" reasoner aimed at traversing class and property hierarchies defined by transitive and symmetric properties of *rdfs:subPropertyOf* and *rdfs:subClassOf*. This engine is useful to perform a high-performance transitive closure over class and property hierarchies, and much more efficient than using the rule-based engines (Built-in RDFS or OWL reasoners).

The underlying rule-based reasoner of the RDFS and OWL reasoner is available to perform general purpose rule-based reasoning. Forward and backward chaining are supported and an hybrid strategy is also available. The hybrid strategy consists in first saturating the knowledge base with the forward chaining rules, and generating backward chaining rules relevant to the data set. Then semantic queries can be performed through the backward chaining engines.

3.1.3 Pellet

Pellet [65] is presented as a sound and complete OWL-DL reasoner. All reasoning tasks are reduced to consistency checking (through the so called *tableaux* algorithm) and cover TBox and ABox. The underlying knowledge base can be interfaced with Jena, OWL API, or standard DIG applications. TODO Optimizations and incremental reasoning (materialization ?)

3.1.4 Fact++ (TODO)

[69]

3.1.5 Kaon2 (TODO)

[51]

3.1.6 HermiT (TODO)

[63]

3.1.7 Racer (TODO)

[27]

3.2 Reasoners comparison (TODO)

4 Semantic Web Services Challenges

Semantic web services (SWS) has emerged has a research domain on its own. SWS aims at leaning on semantic technologies to enhance service oriented architectures. In particular, benefits are expected in the processes of discovery and composition of services, but also in the process of mediating syntactically incompatible services.

4.1 Semantic Web Services

Nowadays, organizations heavily rely on *Service-Oriented Architecture (SOA)* to improve their agility in a quickly evolving and competing world. Enterprises need to cost-effectively and almost instantly react to evolving markets. Similarly scientific communities need to continuously react to novel conceptual or technical approaches to tackle their information processing challenges. Inside or outside organizational boundaries, sharing is also a major issue to foster collaborations. In that direction SOA provide some solutions by focusing on loosely-coupled processes, by focusing on their description and therefore on reusability.

More precisely, Service-Oriented Architectures provide software infrastructures to publish and consume, over a network, units of information processing, *i.e. Services*. The World Wide Web has evolved to allow the distribution and the consumption of such services through Web Service standards, over established web protocols. Web services rely mainly on message exchanges between service providers and service consumers. Messages are encoded in XML and conform to the Simple Object Access Protocol (SOAP) standard. The Web Service Description Language (WSDL) standard focus mainly on describing the SOAP messages a Web Service can understand and produce as results. Web Services are interoperable in the sense that a consumer doesn't have to know how the provider implemented the service, but only needs to know the service requirements, through its WSDL file, in order to provide the needed SOAP messages for invocation, and for interpretation of results.

But Web Services do not provide any external understanding about the nature of the performed information processing so that, even supported by service orchestration tools or languages, elaborating a flow of services (workflow), must be down manually with a clear understanding of how the data is processed. The nature of data transformation is highly dependent on the domain in which the service is created or used. Another reason that prevents automation when designing workflows is the lack of description about (i) requirements on domain entities prior to service invocation, and (ii) effects on domain entities when the service invocation is finished. All this additional descriptive information involving domain specific concepts is generally called semantic description, in opposition to syntactic description provided by WSDL.

Semantic-Web technologies are good candidates to address issues raised by coherently managing a huge variety of published services over the Web or institutional networks. Among others, improving the reliability of search tasks, providing guidance when plugging together available services, or addressing several degrees of (in)compatibility between services are challenges driving the Semantic Web Services (SWS) area. In that sense, the SWS Challenge initiative [55] promotes Semantic Web Service technologies and aims at developing their common understanding through a set of use cases.

The *Annotation* task is a prerequisite to bridge Web Services and the Semantic Web. Annotating a service consists in using an ontology to bind technical concepts, usually elements describing web services (including their messages), to domain specific concepts. This is a costly task mainly because the expert performing the annotation need both technical and domain specific knowledge. For instance, let us consider a medical image processing tool performing a de-noising operation. From a technical or syntactical point of view, the service might be implemented by an executable binary taking as input a raw file and producing as output another raw file containing the de-noised image. But from a semantic point of view, this de-noising service might implement a kind of algorithm characterizing how the image is processed. The service might additionally require a specific medical image format, and a specific modality of acquisition, for instance ultrasound. Moreover, the resulting de-noised image should conserve the same modality. In other world, even de-noised, the image still has been acquired through an ultrasound device. Providing automation in the generation of such new annotation takes part in the SWS challenges.

Once annotated, the service can be searched through natural terms for domain experts. This task is generally known as semantic *Discovery* process. Capabilities of services can therefore be classified with reasoners so that semantic search engines expose most adequate services. Search requests can also involve the description of information to process, or to be processed, in order to retrieve the service candidates most able to consume or produce this kind of information. Service discovery activities rely on so-called matchmaking algorithms, generally operationalized through inference rules and reasoners.

Whereas Web Service Choreography languages address complex issues such as global web services coordination and collaboration, we only consider *Composition* of services with the goal of providing a guidance to the workflow designer when plugging together services. Considering a given semantic web service S, the process of composition rely mainly on discovering either candidate services able to produce semantically compatible data as inputs of S, or able to consume data semantically compatible with data produced by S. Semantic compatibility will be presented later on.

Mediation challenges consist in, considering two heterogeneous services which are semantically pluggable, taking into account syntactic mismatches. Services able to perform data transformations or conversions to conform to syntactic constraints are not always available, and it might be not desirable to expose them as services. Some semantic web service approaches therefore consider providing data transformation guidelines or implementations to fulfill syntactic mismatches.

4.2 From service models to semantic web services

OWL-S is one of the major initiatives to bring Semantics to Web Services [45, 44]. OWL-S is a set of notations based on OWL [48], the Semantic Web ontology language. This framework is composed of three linked ontologies, *(i)* the *profile ontology* which describe what the service does, *(ii)* the *process ontology* which describe how the service is used and *(iii)* the *grounding ontology* which describes the service interface. High-level objectives are to provide a standard[11] representational framework for Web Services, to support automation of Web Services management and to be comprehensive enough to cover the whole lifecycle of service tasks. Key concepts used to describe Services are *inputs*, *outputs*, *preconditions*, and *results* which refer to *effects* (postconditions) on outputs specifications. The goal of the *profile* sub-ontology is to support capability-based discovery. The *profile* defines functional, classification, and non-functional aspects of a service. The functional aspect represents actually the information transformation between the inputs and the outputs, and the domain transformation between the preconditions and the effects. The classification aspect represents the type of a service in a taxonomy. The non-functional aspect represent transverse concerns such as security or quality of service, that can be defined "by extension" using service parameters that refer to another ontology. The *process* sub-ontology is another workflow language based on atomic processes (without internal structure), and composite processes (with both control flow and data flow). The *grounding* sub-ontology links *(i)* each *atomic process* to a deployed Web Service, and more precisely, to a WSDL[12] [14] operation, and *(ii)* abstract *inputs* and *outputs* to WSDL messages. Other works on the applications of OWL-S are shown in terms of service enactment, service discovery, service composition (planning techniques). Related approaches to OWL-S such as WS-BPEL, ebXML, CDL, SWSF, WSMO or WSDL-S are presented in [1].

WSMO (Web Service Modeling Ontology) is another major initiative to address SWS challenges. Roman & al. propose an unified framework composed of a conceptual model for Semantic Web Services, a formal language for their description, and an execution environment hosting SWS [60]. WSMO is proposed to describe *Web Services*, *User Goals*, and to tackle the interoperability issues through *Ontologies* (potentially domain specific) and *Mediators*. The usage of such mediators is specific to WSMO. They capture mappings that cannot be expressed through ontologies and are used as computing elements to resolve mismatches when plugging together WSMO elements. They are able to address the heterogeneity of ontologies, of goals, of web services, and also the one from web services to user goals. WSML is a syntactic framework for a set of layered languages covering the description of ontologies, web services, user goals and mediators. The execution environment WSMX originally designed to be a testbed for WSMO provides support for SWS discovery and invocation but also provides an internal workflow engine and a resource manager able to store/retrieve general data produced within the environment.

Chabeb & Tata propose in [13] a semantic annotation framework (using the Eclipse Modeling Framework) for web services able to generate descriptions using SAWSDL, OWL-S or WSMO recommendations based on an extension of the SAWSDL standard. This extension is composed of a technical ontology aimed at describing service concepts including non-functional properties, and a domain ontology which describes the semantics of the

[11]from both Semantic Web and Web Services standards

[12]WSDL is the *de facto* standard to describe Web Services, but the grounding sub-ontology of OWL-S can be exploited with other service implementations such as UPnP.

service. The contribution is presented as a more expressive framework than the SAWSDL recommendation of the W3C.

4.3 From legacy web services to semantic web services

Derived from WSDL-S, the World Wide Web Consortium (W3C) released in 2007 the SAWSDL standard [20], aimed at defining Semantic Annotations for WSDL and XML Schema. Martin et al. argue that a joint usage of OWL-S and SAWSDL [46] is a good way to tackle SWS challenges. SAWSDL is indeed presented as lightweight but too modest to address SWS objectives. SAWSDL introduces the annotations *modelReference* (for both WSDL and XSD) and {*lifting,lowering*}*SchemaMapping* (only for XSD). Some limitations of the *modelReference* annotation are pointed out. For instance, if we consider two WS operations with two input parameters sharing the same XML Schema type but two different semantic referents A and B, the two services become ambiguous and it deserves the discovery process. Indeed the two services cannot be distinguished if the two parameters p_1 and p_2 have for semantic referent respectively A and B for the first service and p_1 and p_2 have for semantic referent respectively B and A. OWL-S classes (*input*, *output*, and *process*) should be used as an intermediate description between a WS and a domain ontology. The authors conclude by giving a set of recommendations for annotating web services with both SAWSDL and OWL-S.

Similarly to the joint usage of SAWSDL and OWL-S presented before, Vitvar & al. propose WSMO-Lite [70], a bottom-up service modeling approach to benefit from the SAWSDL standard in WSMO-fashioned Semantic Web Services. This framework is lighter than WSMO: *mediators* are considered as provided by the infrastructure, similarly to *user goals* which are supposed to be provided by the discovery mechanism. Contrary to WSMO, WSMO-Lite does not constraint to use WSML but allows the usage of any ontology written in an RDF syntax. This lightened version no more focuses on semantics of pre/post-conditions and effects as they are domain specific. The main difference between the OWL-S approach to conform to the SAWSDL standard, is that they chose to reduce WSMO to a minimal model for SWS.

Derouiche & Nicole propose to extend the Windows Workflow Foundation with SAWSDL documents to guarantee semantic type correctness in scientific workflows [17]. Their contribution consists in adding a new "Semantic Web Service" activity responsible of annotating web services (WS) annotation. The semantic type checking is performed by extending the binding activity (data flows between services) of the framework with calls to the introduced "Semantic Web Service" activity. Structural mismatch are solved using *lifting* and *lowering* schema mappings in conjunction with XSLT transformations. The authors also refer to a prototype from the Taverna workbench used to address semantic mismatch detection and resolution in workflows, but the binding of concrete and semantic datatypes is not addressed and prevent the support for workflow enactment.

Lécué & Moreau also tackle semantic and syntactic web services composition in a quite similar approach [41]. Their motivation is based on the abundance of approaches dealing with service discovery but the lack of these addressing syntactic message transformation to perform effective mediation. As background information, they describe *causal links* and *data integration* in the Web. Causal links are defined in a formal model [40] and represent semantic connections between functional output and input parameters of a service. From an operational point of view, the semantic annotation of these parameters might be realized using SAWSDL, WSMO capability or OWL-S profile. The matching / binding of parameter is described and categorized between five matching types: *exact, plugin, subsume, intersection, and disjoint* which allows at design time, to qualify several levels of (in)compatibilities. Data integration in the Web is considered through the use of XSL technologies to transform XML messages. The proposed semantic matchmaker consists in, being assumed that the control flow is already determined, scoring all causal links between parameters. For syntactic adaptations, no guidelines are provided by the SAWSDL standard. The proposed syntactic engine automatically generates, from the semantic links, an XSL transformation to adapt XML messages between services trough a syntactic and structural analysis of XML schemas (Boukottaya's phd thesis).

Often syntactically described through WSDL, Grid Services are also desired to be semantically discovered [62]. Thamarai Selvi & al. propose a bottom-up approach leading to a matchmaking system not only based on input/output description of services but also based on their functionalities. The proposed framework is based on OWL-S tools to generate an OWL-S instance of a service from its WSDL. The similarity distance between services is computed through a *Domain ontology* which describes input and output datatypes, and a *Function ontology* which describes the capabilities of services. The authors conclude with a favorable evaluation showing that the elimination of irrelevant services, by considering also their functionalities, is more efficient that by considering only their parameters.

The FUSION project [36] aims at supporting enterprise application integration through Semantic Web Services technologies, namely UDDI [53], SAWSDL and OWL. Service discovery requirements cover functional and non-functional properties. *Inputs*, *Outputs*, *Preconditions* and *Effects* (IOPE) are generally considered but FUSION only focus on *Inputs* and *Outputs* (*i.e.* information transformations between services and not state-wise conditions) at both message and schema levels.The classification of the service within a classification scheme is considered as a non-functional property. The proposed architecture is independent from any standard implementation of UDDI, or OWL knowledge bases. Software components UDDI4J, SAWSDL4J, WSDL4J, OWL-API and Pellet reasoner are the basis of a set of web services used to publish and discover services.

The METEOR-S research project is another major initiative in the SWS area [64]. The approach is based on a peer-to-peer middleware to address service discovery and publication. SAWSDL is used for both services annotation (through *modelReferences*) and data mediation (through schema *lifting/lowering*). A new approach, gaining more and more popularity, is introduced as a lightweight manner to address SWS. It is based on RESTful web services and microformats for their description. RESTful web services are often presented as simple Web API through HTTP (GET, PUT, POST, DELETE, etc.) without any description language like WSDL which prevents discovery capabilities. This need is therefore addressed through the recommended usage of the hRESTs microformat [35], which can be directly embedded within web pages and searched through regular web search engines.

Cardoso et al. review the state-of-the-art in Web Service Discovery and propose a new algorithm to tackle semantic discovery [12]. The novelty of the approach consists in (i) considering a judgment process on both similar and distinguishing features (Tversky's model), (ii) not only considering inputs and outputs but also the functionality of the service, and (iii) considering potentially different semantic references. This work is based on the METEOR-S infrastructure. The authors review the three main approaches developed in the SWS research area, namely OWL-S, WSMO, and SAWSDL annotations, with a focus on SAWSDL. Four discovery approaches are covered: (i)*IO matching*, (ii) *Multi-level matching* by exploiting functional and non-functional service information, (iii) *Graph-based matching* based on directed graph representing service descriptions and linked concepts from domain ontologies, and (iv) *Syntactic matching* based on information retrieval techniques. Two tools (Eclipse plugins) resulting from the METEOR-S project are used in this context: Radiant for service annotation and Lumina for service discovery/advertisement.

Built upon the Semantic Web search engine CORESE[16], Lo & Gandon propose an approach to address enterprise application integration through Semantic Web Services

[42]. To annotate services, their approach relies on the *profile* and *grounding* ontologies, and on the *input* and *output* of the OWL-S *process* ontology. From the user point of view, a Semantic Web Portal embedding CORESE is used to discover services trough SPARQL queries. This portal is also dedicated to assist the user during Web Service composition tasks. The CORESE engine can therefore be seen as a semantic UDDI, acting as a Web Service Broker in classical SOA architectures. Two cases are considered for interactive WS composition addressing inputs availability or desired outputs. Sequences of services (a set of service combinations) can also be discovered to achieve several data transformations between available inputs and desired outputs. The composability is computed by the engine through the execution of *production rules*.

To address semantic mediation between business partners (one of the SWS challenges) Küster & König-Ries [37] propose the DIANE Service Description (DSD). Contrary to OWL-S, DSD focuses on a clear distinction between service offers and requests. Instead of instantiating an ideal service and searching through the reasoner for the closest "real-world" service, DSD specify a service request with a fuzzy set of acceptable services and thus copes with user's preferences, in other worlds, copes with what is really needed. DSD is used to solve one of the SWS Challenge mediation problem in an original way: the mediation issue is viewed as a discovery issue. The DIANE middleware both captures service requests, and service offers, performed by stakeholders, and its matchmaker discovers suitable services to handle the requests. The DIANE middleware can also handle the invocation of services through grounding information of the service description offer and through their proper data lifting/lowering between xml messages and ontological DSD data. In order to promote SWS sharing and evaluation, the OPOSsum portal [38] has been developed and allows, among others, categorizing services, editing semantic annotations based on several standards (OWL-S, WSML, WSDL).

4.4 Hybrid semantic and syntactic service discovery

Hybrid approaches to semantic discovery of services are presented in [33] and [34]. Adding structural similarity measurement (through information retrieval techniques) to logic-based matchmakers improves significantly the discovery process.

As we have seen previously in bottom-up SWS initiatives, the approaches rely heavily on manually annotating WSDL, and providing a binding with a domain ontology. Based on manually adding textual annotations to message parts of WSDL, Vouros et al. [71] propose a method to automatically bind WSDL input and output messages to classes of a domain ontology.

4.5 Applied Semantic Web Services

In the context of the SIMDAT Pharma Grid project, Qu et al. propose a semantics-enabled service discovery framework based on OWL-S descriptions and ontology reasoning [58]. The service discovery component is a major requirement for biologist end-users to conduct *in silico* experiments, at a high level of abstraction. OWL-S has been chosen for its maturity in comparison with WSMO and for its expressivity in comparison with WSDL-S, from which has been derived SAWSDL. The authors highlight that "non-semantic" information, for instance non-functional infrastructure requirements such as site memory or bandwidth need to be added in an early stage of the service discovery process in order to avoid time consuming reasoning.

Built upon the ^{my}Grid ontology, a bioinformatics service and domain ontology, FETA [43] is a service discovery framework characterized by a light-weight semantic support and a semi-automatic approach. Three main are actors are distinguished in this framework:

both *knowledge engineers* and *service annotators* provide semantic enhanced web services consumed by *scientists*. During the annotation process, services are registered in a UDDI repository (used as an intermediate component within the FETA architecture) and an XML description file is produced to link the service to concepts of the ontology. To cope with the light-weight requirements, they avoid usage of Description Logics reasoning and predefine a set of RDF queries covering the major needs of the bioinformatics community. Feta is part of the ^{my}Grid technologies and referenced in a work focusing on discovery and reusing of scientific workflows [73]. In a bioinformatics context, the authors highlight that for usability purpose they do provide full description logic reasoning. The ^{my}Grid ontology is described in [72] with relevant informations on the usability of semantic technologies by scientists to build in silico experiments. In particular, two kind of services are distinguished, namely *domain* services aimed at performing a scientific function, and *shim* services ("helper" services) aimed at gluing *domain* services. While complex Description Logics reasoning is needed to precisely and automatically select *shim* services aimed at resolving mismatches, simple RDFS descriptions are more suitable to service discovery. Indeed, RDFS descriptions are simpler and it is preferable to retrieve a shortlist of candidate services and let scientists realize the final decision. Authors also reported that OWL created a barrier of adoption for the user communities and that OWL flavors should be hidden to the end-users.

Belhajjame & al. overview in [7] the management of Metadata in the scientific workflow design and enactment workbench, Taverna. Metadata are presented as valuable structured and descriptive information about (i) workflows and their related entities, and (ii) workflow executions. Main Objectives are to enhance the components of workflow systems and the usage of such systems. The classification of workflow entities with domain specific concepts is described. Annotations are proposed to cover both tasks and parameters of processing units with benefits on focused (domain based) browsing and also on accurate description by domain experts. Additionally, *parameter instances* are proposed to populate a repository of testing data to prevent regressions, and assert availability and reliability of processing units. Metadata describing the workflow execution, namely provenance metadata, is also described through both process and data point of view. The metadata life cycle is described through *creation*, *maintenance* and *curation* tasks, and take into account third-party constraints such as service description importation, metadata storage in RDF/XML repositories, references to external domain ontologies. SWS challenges and provenance metadata usage are investigated through an implementation within the Taverna workbench.

Semantically annotating services is a costly a time consuming task. Belhajjame & al. propose an automatic annotation framework based on existing annotated workflows from the ^{my}Grid repository [6]. The derivation of a set of annotations for input or output parameters consist in inferring from existing compatible connections between outputs and inputs, the semantics type candidates. An annotation algorithm is proposed and as a favorable side effect, the detection of conflicts lead to the curation of already annotated workflows, by excerpting errors, in workflows or in annotations. An annotation workbench as been implemented, covering both manual annotation (enhanced by hints with derived annotations), annotation conflicts identification and resolution, and workflow inspection.

The BioMOBY project aims at providing interoperability for biological data centers and analysis center. SAWSDL has been used in this context and this real-world application is one of the few existing initiatives [22]. The focus is on interoperability and therefore on schema mapping annotations of SAWSDL, implemented through XSLT stylesheets. The entry-point is a SAWSDL Proxy servlet, in front of a web service provider, a semantic registry, and a schema mapping server.

Larvet et al. propose in [39] a process covering the semantization of legacy web services using SAWSDL. The framework assists the service developer by providing a way to add semantic tags to the service inputs, outputs, or to its goal, avoiding manipulation of ontologies or XML files. Few details are shown about concepts retrieving and services publishing.

4.6 Comparisons and applicability in a collaborative neuroscience platform

		Challenges					
Approaches	Formalism	Annotation	Discovery	Composition	Mediation	Targeted end-users	Year
WSMO	•	x	x	x	x	s	2005
OWL-S	•	x	x	x	-	s	2004
Taverna	•/+	x	x	x	-	d	2008
Lécué & al.	+	-	x	x	x	s	2008
METEOR-S	+	x	x	-	x	s	2008
hRESTs	+	x	x	-	-	s	2008
Cardoso & al.	+	x	x	-	-	s	2008
OWLS-MX	+	x	x	-	-	s	2009
SAWSDL-MX	+	x	x	-	-	s	2009
SIMDAT	+	x	x	-	-	d	2008
FETA	•/+	x	x	-	-	s/d	2005
BioMOBY	•	x	-	-	x	d	2008
CORESE	+	-	x	x	-	s	2005
DIANE	+	-	x	-	x	s	2008
Derouiche & al.	+	-	-	x	x	s	2007
SAWSDL	+	x	-	-	-	s	2007
WSMO-Lite	+	x	-	-	-	s	2008
Yasa4wsdl	•	x	-	-	-	s	2008
Larvet & al.	+	x	-	-	-	d	2008
Vouros & al.	+	x	-	-	-	s	2008
Selvi & al.	+	-	x	-	-	s	2006
FUSION	+	-	x	-	-	s	2008

Table 2: Despite most of approaches covering annotation and discovery, none of them address all challenges and target the domain experts.

Through a large number of initiatives, this review shows a huge interest in enhancing web services with semantic web technologies. Table 2 categorizes the existing approaches by highlighting the tackled challenges. Criteria also cover the use of proper (•) or external (+) formalism (language, model or ontology), and the targeted end-users: service experts (s) or domain experts (d).

A consensus appears, relying on the SAWSDL standard to bridge the gap between traditional web service and semantically enhanced services. Such a technological consensus could minimize the software development efforts and foster its adoption. However SAWSDL features cannot, alone, tackle SWS challenges. Reasoning engines and precise enough descriptions of services are needed, being orginally addressed by OWL-S and WSMO approaches. Domain experts seemed to be reluctant to jump into the WSMO world and rather preferred the lighter framework OWL-S, which aims at modeling first and which leans on third party OWL reasoners.

Section 4.5 focuses on approaches driven by demanding user communities, especially from the Bioinformatics area. A lot of work has been achieved and should be considered for SWS applications in Neuroinformatics fields. Approaches should rely on SAWSDL and OWL-S for a technical description of services, and additionnaly a Neuroinformatics domain ontology to address at least service features classification, and parameters compatibility in workflows.

Neuroscientists involved in the VIP project are building such a Neuroinformatics domain ontology, representing both knowledge on manipulated medical data and simulation parameters, simulated images, and knowledge on their dedicated processing or simulation tools, exposed as services. They also rely on large scale grid infrastructure to support data storage and computing tasks described in their workflows similarly to *"in silico"* experiments in the SIMDAT Pharma Grid or workflows in the ^{my}Grid environment. The VIP platform could thus benefits from these semantic approaches to assist users in handling simulation results, but also to assist them during the design of new simulation workflows.

5 Close-up to the Virtual Imaging Platform (VIP)

5.1 Handling of simulation results

In the context of the Virtual Imaging Platform, the semantic annotation of simulation runs consists in producing RDF triples all along the execution of simulation workflows. Several software components are in charge of realizing this process, namely the *Semantic client*, the *Workflow engine*, the *Annotation store*, and finally the *Reasoning engine*. The *Workflow engine* will produce "on-the-fly" provenance information that will be stored as semantic annotation into the the *Annotation store*. For each workflow output, domain-specific annotations (based on the VIP ontology) will be produced. This result annotation process will be based on the analysis of the provenance of the simulation result and will be supported by the *Reasoning engine* to determine the nature of the result (in terms of classes), and its relationships to other entities involved in the simulation experiment (in terms of properties).

As it has already be mentioned, the semantic annotation process might lean on some reasoning tasks to infer annotations from the underlying VIP ontology. A forward-chaining rule engine may be useful in order to produce new annotations from existing annotations, and based on domain-specific (VIP) inference rules.

Any of the previously presented technology focusing large datasets is relevant with regards to the storage of VIP semantic data. The materialization need to be carefully considered. If the knowledge base does not evolves to often, the inferred knowledge can be computed at each insertions/deletions of new triples so that semantic queries remain fast (static materialization). On the other hand, if insertions/deletions are frequent, the system cannot be responsive since it needs to recompute all the inferences dynamically for each semantic query (dynamic materialization). A trade-off needs to be found between fast queries, or fast updates of the knowledge base.

Semantically browsing the knowledge base highly depends on the kind of proposed queries, and in particular if these queries need to beforehand perform some reasoning task. As they tend to propose scalable and computationally tractable reasoning, built-in reasoners provided by state-of-the-art semantic repositories might be limited in comparison with state-of-the-art Description Logic reasoners such as those presented in section 3.1.

5.2 User assistance

5.2.1 Adapting simulation experiments

Another facet of the use of semantic technologies concerns the user assistance in the set up of simulation experiments. Indeed, semantic technologies can be used to help user in realizing appropriate choices depending on the nature of simulation workflow components or the nature of their parameters. Three categories of assistance are being considered:

1. *Parameter documentation*: consists in giving to the user relevant information, based on the description of its underlying concept defined in the VIP ontology. Such assistance highly depends on both TBox and ABox reasoning. Indeed, a *realization* operation must be performed to retrieve its more specific concepts and extract their documentation (ABox reasoning task). Later on, these concepts need to be classified to extract the documentation of its "subsumers", i.e., generalized concepts (TBox reasoning task). Almost all presented semantic repositories, with built-in reasoning engines, are capable of realizing such tasks.
2. *Parameter suggestion*: consists in proposing to users appropriate parameter values or value ranges in order to let them evolve in a "constrained world", tending to avoid "no-win" simulations. This kind of user assistance leans on *instance checking* and reasoning over values and data range restrictions. Since the corresponding "owl:hasValue", "owl:allValuesFrom" and "owl:dataRange" constructs are out of the scope of OWL-Lite, DL reasoners should be considered for this context. Nevertheless Jena supports validation for "range" or a "allValuesFrom" violations for datatype properties, and also support the "owl:hasValue" construct. A more challenging task consists in suggesting appropriate parameter values from the understanding of the other fixed parameters of the simulation. This could be achieved through rule-based reasoning since the "side-effect" of parameters are formalized into a set of inference rules.
3. *Parameter checking*: aims at validating all simulation parameters in order to advertise users on possibly "no-win" computations. It consists in (i) leaning on the constraints expressed within the VIP ontology, if they can be embedded within the ontology, and (ii), consists in expressing rules to validate the consistency of the parametrization of simulation experiments. While DL reasoners are suitable for (i), since they support almost all restriction constructs of OWL (cardinality, value restrictions, etc.), a general rule-based inference engine is needed to compute consistency through a set of rules.

5.2.2 Integrating new models or simulators

During the exploitation of the VIP platform, users will face some extensibility issues, for instance when they will intend to work with a more precise organ model, or they will update some components of a simulation workflow, or integrate a completely new simulator. Semantic Web Services approaches presented in section 4 should be considered to assist users at workflow design-time, especially for the following tasks:

1. *Annotation*: simulation workflow components should be semantically annotated (manual operation), to precisely describe the nature of their function and their parameters. This is necessary to allow services, for instance model converters, for being advertised properly. Moreover, during the publication of such annotated tool, a precise version number should be given to ease the update of annotated services.
2. *Discovery*: a semantic registry of simulation workflow components should be available to retrieve the most appropriate component at workflow design-time, through a semantic classification of these component, or the execution of pre-defined SPARQL queries.
3. *Composition*: at workflow design-time, users should be advertised about suitable candidate services (semantically compatible) able to consume the data produced by a selected workflow component. Moreover pre-processing components such as converters or any "helper" components for simulation workflows could be advertised as it is proposed in the ^{my}grid environment.
4. *Validation*: As it has been proposed for parameter checking, it could be interesting – at workflow design-time – to validate the overall consistency of a simulation workflow, based on the VIP ontology and an appropriate reasoning engine.

References

[1] A. Ankolekar, D. Martin, D. McGuinness, S. McIlraith, M. Paolucci, and B. Parsia. OWL-S' Relationship to Selected Other Technologies [http://www.w3.org/submission/owl-s-related], November 2004.

[2] K. Anyanwu, A. Maduko, and A. Sheth. Sparq2l: towards support for subgraph extraction queries in rdf databases. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 797–806, New York, NY, USA, 2007. ACM.

[3] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[4] F. Baader and W. Nutt. Basic description logics. In Baader et al. [3], pages 43–95.

[5] S. Bechhofer, T. Liebig, M. Luther, P. Patel-schneider, B. Suntisrivaraporn, A.-y. Turhan, T. Weithöner, D. Euro-labs, and T. Dresden. DIG 2.0 – towards a flexible interface for description logic reasoners. In *Proceedings of the second international workshop OWL: Experiences and Directions*, 2006.

[6] K. Belhajjame, S. M. Embury, N. W. Paton, R. Stevens, and C. A. Goble. Automatic annotation of web services based on workflow definitions. *TWEB*, 2(2), 2008.

[7] K. Belhajjame, K. Wolstencroft, O. Corcho, T. Oinn, F. Tanoh, A. William, and C. A. Goble. Metadata management in the taverna workflow system. In *International Conference on Computational Science (4)*. IEEE Computer Society, 2008.

[8] T. Berners-Lee, J. Hendler, and L. Ora. *The Semantic Web*. Scientific American, 2001.

[9] C. Bizer and A. Schultz. The berlin sparql benchmark. *International Journal On Semantic Web and Information Systems*, 2009.

[10] J. Broekstra, A. Kampman, and F. V. Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. pages 54–68. Springer, 2002.

[11] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The dl-lite family. *J. of Automated Reasoning*, 39(3):385–429, 2007.

[12] J. Cardoso, J. Miller, and S. Emani. Web Services Discovery Utilizing Semantically Annotated WSDL. *Reasoning Web*, pages 240–268, 2008.

[13] Y. Chabeb and S. Tata. Yet another semantic annotation for wsdl (yasa4wsdl). pages 462–467, October 2008.

[14] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1 [http://www.w3.org/tr/wsdl], March 2001.

[15] H.-P. D. Company. Jena – A Semantic Web Framework for Java[http://jena.sourceforge.net/], 2009.

[16] O. Corby, R. Dieng-Kuntz, and C. Faron-Zucker. Querying the semantic web with corese search engine. In R. L. de Mántaras and L. Saitta, editors, *ECAI*, pages 705–709. IOS Press, 2004.

[17] K. Derouiche and D. A. Nicole. Semantically resolving type mismatches in scientific workflows. In *OTM Workshops (1)*, pages 125–135, 2007.

[18] C. Dolbear, A. Ruttenberg, and U. Sattler, editors. *Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions, collocated with the 7th International Semantic Web Conference (ISWC-2008), Karlsruhe, Germany, October 26-27, 2008*, volume 432 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[19] O. Erling and I. Mikhailov. RDF Support in the Virtuoso DBMS. In S. Auer, C. Bizer, C. Müller, and A. V. Zhdanova, editors, *Conference on Social Semantic Web*, volume 113 of *LNI*, pages 59–68. GI, 2007.

[20] J. Farrell and H. Lausen. Semantic Annotations for WSDL and XML Schema [http://www.w3.org/tr/sawsdl], August 2007.

[21] J. Futrelle, J. Gaynor, J. Plutchak, J. D. Myers, R. E. McGrath, P. Bajcsy, J. Kastner, K. Kotwani, J. S. Lee, L. Marini, R. Kooper, T. McLaren, and Y. Liu. Semantic middleware for e-science knowledge spaces. In *MGC '09: Proceedings of the 7th International Workshop on Middleware for Grids, Clouds and e-Science*, pages 1–6, New York, NY, USA, 2009. ACM.

[22] P. M. K. Gordon and C. W. Sensen. Creating bioinformatics semantic web services from existing web services: A real-world application of sawsdl. In *ICWS*, pages 608–614, 2008.

[23] K. Grant Clark, L. Feigenbaum, and E. Torres. SPARQL Protocol for RDF [http://www.w3.org/tr/rdf-sparql-protocol/], January 2008.

[24] B. C. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, and U. Sattler. Owl 2: The next step for owl. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(4):309 – 322, 2008. Semantic Web Challenge 2006/2007.

[25] W. O. W. Group. Owl 2 web ontology language document overview [http://www.w3.org/tr/owl2-overview/], October 2009.

[26] Y. Guo, Z. Pan, and J. Heflin. Lubm: A benchmark for owl knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2-3):158–182, October 2005.

[27] V. Haarslev and R. Möller. Racer system description. In *IJCAR '01: Proceedings of the First International Joint Conference on Automated Reasoning*, pages 701–706, London, UK, 2001. Springer-Verlag.

[28] P. Hayes and B. McBride. RDF semantics [http://www.w3.org/tr/rdf-mt], February 2004.

[29] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML [http://www.w3.org/submission/swrl/], May 2004.

[30] I. Horrocks, P. F. Patel-Schneider, and F. V. Harmelen. From shiq and rdf to owl: The making of a web ontology language. *Journal of Web Semantics*, 1:2003, 2003.

[31] J.-D. Kim, H. Shin, D. Jeong, and D.-K. Baik. Jena storage plug-in providing an improved query processing performance for semantic grid computing environment. In *Computational Science and Engineering Workshops, 2008. CSEWORKSHOPS '08. 11th IEEE International Conference on*, pages 393 –398, 16-18 2008.

[32] A. Kiryakov, D. Ognyanov, and D. Manov. OWLIM –A Pragmatic Semantic Repository for OWL. *Web Information Systems Engineering –WISE 2005 Workshops*, pages 182–192, 2005.

[33] M. Klusch, B. Fries, and K. Sycara. OWLS-MX: A hybrid Semantic Web service matchmaker for OWL-S services. *Web Semant.*, 7(2):121–133, 2009.

[34] M. Klusch, P. Kapahnke, and I. Zinnikus. Hybrid Adaptive Web Service Selection with SAWSDL-MX and WSDL-Analyzer. In *ESWC 2009 Heraklion: Proceedings of the 6th European Semantic Web Conference on The Semantic Web*, pages 550–564, Berlin, Heidelberg, 2009. Springer-Verlag.

[35] J. Kopecký, K. Gomadam, and T. Vitvar. hRESTS: An HTML Microformat for Describing RESTful Web Services. *Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*, 1:619–625, 2008.

[36] D. Kourtesis and I. Paraskakis. Combining SAWSDL, OWL-DL and UDDI for Semantically Enhanced Web Service Discovery. In *5th European Semantic Web Conference (ESWC2008)*, pages 614–628, June 2008.

[37] U. Küster and B. König-ries. Semantic mediation between business partners - a sws-challenge solution using diane service descriptions. *Web Intelligence and Intelligent Agent Technology, International Conference on*, 0:139–143, 2007.

[38] U. Küster, B. König-Ries, and A. Krug. OPOSSum - an online portal to collect and share semantic service descriptions. In *Proceedings of the 5th European Semantic Web Conference (ESWC08), Poster Session*, Tenerife, Canary Islands, Spain, June 2008.

[39] P. Larvet, B. Christophe, and A. Pastor. Semantization of legacy web services: From wsdl to sawsdl. In *ICIW '08: Proceedings of the 2008 Third International Conference on Internet and Web Applications and Services*, pages 130–135, Washington, DC, USA, 2008. IEEE Computer Society.

[40] F. Lecue and A. Léger. A formal model for semantic web service composition. In *5th International Semantic Web Conference (ISWC2006)*, 2006.

[41] F. Lécue, S. Salibi, P. Bron, and A. Moreau. Semantic and syntactic data flow in web service composition. *Web Services, IEEE International Conference on*, 0:211–218, 2008.

[42] M. Lo and F. Gandon. Integrating dynamic resources in corporate semantic web: an approach to enterprise application integration using semantic web services. Technical Report 5663, INRIA, August 2005.

[43] P. Lord, P. Alper, C. Wroe, and C. Goble. Feta: A light-weight architecture for user oriented semantic service discovery. In *European Semantic Web Conference*, pages 17–31. Springer Berlin / Heidelberg, 2005.

[44] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. OWL-S: Semantic Markup for Web Services [http://www.w3.org/submission/owl-s], Novembre 2004.

[45] D. Martin, M. Burstein, D. Mcdermott, S. Mcilraith, M. Paolucci, K. Sycara, D. L. Mcguinness, E. Sirin, and N. Srinivasan. Bringing semantics to web services with owl-s. *World Wide Web*, 10(3):243–277, September 2007.

[46] D. Martin, M. Paolucci, and M. Wagner. Bringing semantic annotations to web services: Owl-s from the sawsdl perspective. In *ISWC/ASWC*, pages 340–352, 2007.

[47] B. McBride. Jena: a semantic web toolkit. *Internet Computing, IEEE*, 6(6):55 – 59, nov/dec 2002.

[48] D. L. Mcguinness and F. van Harmelen. OWL Web Ontology Language Overview [http://www.w3.org/tr/owl-features], February 2004.

[49] L. Moreau, B. Clifford, J. Freire, Y. Gil, P. Groth, J. Futrelle, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, Y. Simmhan, E. Stephan, and J. V. den Bussche. The open provenance model — core specification (v1.1). *Future Generation Computer Systems*, December 2009.

[50] B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. OWL 2 Web Ontology Language Profiles [http://www.w3.org/tr/owl2-profiles/], October 2009.

[51] B. Motik and U. Sattler. A comparison of reasoning techniques for querying large description logic aboxes. pages 227–241. Springer, 2006.

[52] D. Nardi and R. J. Brachman. An introduction to description logics. In Baader et al. [3], pages 1–40.

[53] OASIS. Introduction to UDDI: Important Features and Functional Concepts. Technical report, OASIS, 2004.

[54] E. Orri and M. Ivan. SPARQL and Scalable Inference on Demand[http://virtuoso.openlinksw.com/dataspace/dav/wiki/main/vosscalableinference], 2009.

[55] C. Petrie, T. Margaria, H. Lausen, and M. Zaremba. Introduction to the first year of the semantic web services challenge. *Semantic Web Services Challenge*, pages 1–11, novembre 2008.

[56] S. Polovina. An introduction to conceptual graphs. In *ICCS '07: Proceedings of the 15th international conference on Conceptual Structures*, pages 1–14, Berlin, Heidelberg, 2007. Springer-Verlag.

[57] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF [http://www.w3.org/tr/rdf-sparql-query], January 2008.

[58] C. Qu, F. Zimmermann, K. Kumpf, R. Kamuzinzi, V. Ledent, and R. Herzog. Semantics-Enabled Service Discovery Framework in the SIMDAT Pharma Grid. *IEEE Transactions on Information Technology in Biomedicine*, 12(2):182–190, 2008.

[59] K. Rohloff, M. Dean, I. Emmons, D. Ryder, and J. Sumner. An evaluation of triple-store technologies for large data stores. *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*, pages 1105–1114, 2007.

[60] D. Roman, J. de Bruijn, A. Mocan, H. Lausen, J. Domingue, C. Bussler, and D. Fensel. Www: Wsmo, wsml, and wsmx in a nutshell. pages 516–522. 2006.

[61] A. Seaborne, G. Manjunath, C. Bizer, J. Breslin, S. Das, I. Davis, S. Harris, K. Idehen, O. Corby, K. Kjernsmo, and B. Nowack. SPARQL Update [http://www.w3.org/submission/sparql-update/], July 2008.

[62] S. T. Selvi, R. Balachandar, K. Vijayakumar, N. Mohanram, M. Vandana, and R. Raman. Semantic discovery of grid services using functionality based matchmaking algorithm. *Web Intelligence, IEEE / WIC / ACM International Conference on*, 0:170–173, 2006.

[63] R. Shearer, B. Motik, and I. Horrocks. Hermit: A highly-efficient owl reasoner. In Dolbear et al. [18].

[64] A. P. Sheth, K. Gomadam, and A. Ranabahu. Semantics enhanced Services: METEOR-S, SAWSDL and SA-REST. *IEEE Data Eng. Bull.*, 31(3):8–12, 2008.

[65] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical owl-dl reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51 – 53, 2007. Software Engineering and the Semantic Web.

[66] J. F. Sowa. *Conceptual structures: information processing in mind and machine*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[67] M. Stocker and M. Smith. Owlgres: A scalable owl reasoner. In Dolbear et al. [18].

[68] L. Temal, M. Dojat, G. Kassel, and B. Gibaud. Towards an ontology for sharing medical images and regions of interest in neuroimaging. *J. of Biomedical Informatics*, 41(5):766–778, 2008.

[69] D. Tsarkov and I. Horrocks. Fact++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.

[70] T. Vitvar, J. Kopecky, J. Viskova, and D. Fensel. WSMO-Lite Annotations for Web Services. In *5th European Semantic Web Conference (ESWC2008)*, pages 674–689, June 2008.

[71] G. A. Vouros, A. G. Valarakos, and K. Kotis. Uncovering wsdl specifications' data semantics. In R. L. Hernandez, T. D. Noia, and I. Toma, editors, *SMRR*, volume 416 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[72] K. Wolstencroft, P. Alper, D. Hull, C. Wroe, P. W. Lord, R. D. Stevens, and C. A. Goble. The mygrid ontology: bioinformatics service discovery. *IJBRA*, 3(3):303–325, 2007.

[73] C. Wroe, C. A. Goble, A. Goderis, P. W. Lord, S. Miles, J. Papay, P. Alper, and L. Moreau. Recycling workflows and services through discovery and reuse. *Concurrency and Computation: Practice and Experience*, 19(2):181–194, 2007.