Semantic Web Services for Satisfying SOA Requirements

Sami Bhiri[1], Walid Gaaloul[1], Mohsen Rouached[2], and Manfred Hauswirth[1]

[1] Digital Enterprise Research Institute (DERI), National University of Ireland, Galway
IDA Business Park, Galway, Ireland
[2] LORIA-INRIA-UMR 7503
BP 239, F-54506 Vandoeuvre-les-Nancy Cedex, France
{sami.bhiri,walid.gaaloul,manfred.hauswirth}@deri.org,
rouached@loria.fr

Abstract. Service oriented modeling is gaining acceptance among academia and industry as a computing paradigm for business and systems integration. Its strong decoupling between service provision and consumption enables much more flexible and cost-effective integration, within and across organizational boundaries, than existing middleware or workflow systems do. However, it also creates new requirements for handling effective service discovery, dynamic service interoperation and automation support for service composition. Web services have been emerging as the lead implementation of SOA upon the Web. The related technologies define common standards that ensure interoperability between heterogeneous platforms. Nevertheless, they fail in satisfying SOA requirements. Semantic Web services initiatives have then emerged with the objective of providing the foundation to overcome these requirements. The main idea is extending service description with machine interpretable information that software programs can reason over it. This chapter discusses how far Web services and semantic Web services initiatives satisfy SOA requirements.

Keywords: SOA, Web services, Semantic Web services, WSMO, OWL-S, IRS-III, METEOR-S.

1 Introduction

Service oriented modeling has been emerging as a paradigm for business and systems integration. Evolving from oriented object programming and component based modeling; it enables much more flexible and cost-effective integration, within and across organizational boundaries, than existing middleware or workflow systems do.

In recent years, Web services (a.k.a WS) have been emerging as the lead implementation of SOA upon the Web. WS have added a new level of functionality for service description, publication, discovery, composition and coordination extending the role of the Web from a support of information interaction to a middleware for application integration.

Nevertheless, current Web service technologies focus only on a syntactic level which hampers automation support for capability-based discovery, and dynamic

T.S. Dillon et al. (Eds.): Advances in Web Semantics I, LNCS 4891, pp. 374–395, 2008.

service composition and invocation. A common agreement is the need to semantically enrich WS description. Similar to the Semantic Web vision, the idea is making WS description more machine interpretable. A new breed of WS, called Semantic Web Services (a.k.a SWS), has then emerged with a promising potential to satisfy SOA requirements. The machine interpretable description of SWS provides the basis for capability-based service discovery, automatic service composition and dynamic service interoperation.

This chapter consists of three parts. First, we depict the concepts and principles of SOA. We highlight the dynamicity and flexibility it ensures and we discern the challenges it poses. In the second part, we present Web services as the key technology implementing SOA principles. We state in particular the main standards and technologies. And we investigate how far they satisfy SOA requirements. Finally, we exhibit SWS initiatives conceptual models and execution environments and inspect their capability to overcome SOA challenges.

2 Service Oriented Architecture

Service-oriented architecture (SOA) is a hot topic in enterprise computing as many IT professionals see the potential of SOA is dramatically speeding up the application development process. Gartner reports that "By 2008, SOA will be a prevailing software engineering practice, ending the 40-year domination of monolithic software architecture" [1] and that "Through 2008, SOA and web services will be implemented together in more than 75 percent of new SOA or web services projects." [1]. Thus, SOA has received significant attention recently as major computer and software companies such as HP, IBM, Intel, Microsoft, and SAP, have all embraced SOA, as well as government agencies such as DoD (US Department of Defense) and NASA.

In this section, we first remind the SOA genesis. After that, we describe SOA principles, terms and benefits. Thereafter we discuss SOA design and implementation requirements. Indeed, our aim is **to explore how successful existing Web services and semantic Web services implementation are to fulfill these requirements** (i.e. SOA requirements).

2.1 SOA Genesis

SOA can be viewed as an evolutionary computing architecture that closely mirrors the history of the industrial revolution. With SOA, computing architectures are expanding beyond object oriented self-sufficiency and now allowing for highly specialized and interoperable computing consumer/producer relationships [2]. Pre 1980, structured procedural programming was prevalent for assembling well structured software code into a software system. Procedural style APIs focus on the natural ability to solve problems via a functional process. The focus is primarily on how to get from point A to point B. This functional way of solving a problem is often a necessary first step when exploring an unfamiliar problem domain. Between 1980 and 1990, Object Oriented Programming (OOP) evolved and established its dominance in the software industry. OOP focuses on combining elements of the problem domain in the form of objects containing data and methods which help to solve the problem of how to get from point A to point B in a way that will also be good to get to point C (reusability).

However, OOP evolved prior to the common distributed computing environments that we have today. Between 1990 and 2000, enterprise tiered architectures evolved and demonstrated that combining methods with data between tiers worked against scalability and loose coupling of the enterprise system, thus the use of data transfer objects between tiers and the focus on the data model for communication between tiers of the enterprise system. Up to the year 2000, individual computing systems remained relatively self-sufficient.

The pre SOA tiered enterprise architectures and implementations did not provide a good solution for computing specialization and computing interdependence at a business or government level. SOA exploded from the evolution of the tiered enterprise architectures and pressures to provide specialized B2B interoperability. Only under the realm of SOA are the concepts of visibility, service descriptions, interaction, contracts and policies, real world effects, execution contexts, etc. combined to provide the architectures and implementations for the automated computing needs of modern computing consumer/producer relationships. SOA is a computing architecture that allows for complex relationships and specializations of computing services on a global scale.

In other words, service-orientation is a way of sharing functions (typically business functions) in a widespread and flexible way. Thanks to the high level dynamicity and flexibility it promises, SOA has been gaining ground as the key architecture for many kinds of applications like B2B interactions, enterprises application integration and grid computing. Indeed, in B2B applications for instance, enterprises can encapsulate and externalise their business processes as services. They can dynamically look for and interact with other services. They can collaborate on the fly to achieve common goals. And they can even establish dynamically virtual enterprises and create new services from existing ones.

2.2 SOA: Terms and Concepts

Service-orientation, as a means of separating things into independent and logical units, is a very common concept. A service-oriented architecture represents an abstract architectural concept defining an information technology approach or strategy in which applications make use of (perhaps more accurately, rely on) services available in a network such as the World Wide Web. It is an approach to building software systems that is concerned with loose coupling and dynamic binding between components (services) that have been described in a uniform way and that can be discovered and composed. Implementing a service-oriented architecture can involve developing applications that use services, making applications available as services so that other applications can use those services, or both. In fact, one way of looking at an SOA is as an approach to connecting applications (exposed as services) so that they can communicate with (and take advantage of) each other.

The fundamental elements of this computing approach are loosely coupled software components, called services. Services are autonomous platform-independent computational elements that can be described, published, discovered and accessed over network-accessible software module. Loosely coupling means that services interactions are neither hard coded (like in Object Oriented Programming), nor specified at design time (like in Component Based Modelling). On the contrary,

services are defined out of any execution context and interact on the fly without prior collaboration agreement.

A service provides a specific function, typically a business function, such as analyzing an individual's credit history or processing a purchase order. It is a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description. A service is provided by an entity – the service provider – for use by others, but the eventual consumers of the service may not be known to the service provider and may demonstrate uses of the service beyond the scope originally conceived by the provider. A service is accessed by means of a service interface, where the interface comprises the specifics of how to access the underlying capabilities. There are no constraints on what constitutes the underlying capability or how access is implemented by the service provider. Thus, the service could carry out its described functionality through one or more automated and/or manual processes that themselves could invoke other available services.

SOA uses the find-bind-execute paradigm as shown in Fig. 1. In this paradigm, service providers register their service in a public registry. This registry is used by consumers to find services that match certain criteria. If the registry has such a service, it provides the consumer with a contract and an endpoint address for that service. In general, entities (people and organizations) offer capabilities and act as service providers. Those with needs who make use of services are referred to as service consumers. In a typical service-based scenario, a provider hosts a network-accessible software module—an implementation of a given service—and defines a service description through which a service is published and made discoverable. A client discovers a service and retrieves the service description directly from the service, possibly from a registry or repository through metadata exchange. The client uses the service description to bind to the provider and invoke the service.

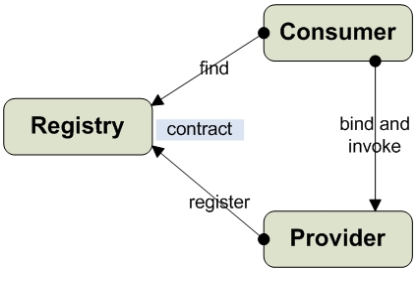

Fig. 1. Find-Bind-Execute paradigm

A service is opaque in that its implementation is typically hidden from the service consumer except for (1) the information and behavior models exposed through the service interface and (2) the information required by service consumers to determine whether a given service is appropriate for their needs. The consequence of invoking a service is a realization of one or more real world effects. These effects may include: (i) information returned in response to a request for that information, (ii) a change to the

shared state of defined entities, or (iii) some combination of (i) and (ii). Summarizing up, the following features are intrinsic for services in SOA in the sense that:

1. Services are software components with well-defined interfaces that are implementation-independent. An important aspect of SOA is the separation of the service interface (the what) from its implementation (the how). Such services are consumed by clients that are not concerned with how these services will execute their requests.
2. Services are self-contained and autonomous: The logic governed by a service resides within an explicit boundary. The service has control within this boundary and is not dependent on other services for it to execute its governance. Underlying logic, beyond what is expressed in the descriptions that comprise the contract, is invisible and irrelevant to service requesters.
3. Services are loosely coupled: Services are designed to interact without the need for tight, cross-service dependencies. What distinguishes SOA from other architecture paradigms is loose coupling. Loose coupling means that the client of a service is essentially independent of the service. The way a client (which can be another service) communicates with the service does not depend on the implementation of the service. Significantly, this means that the client does not have to know very much about the service to use it. For instance, the client does not need to know what language the service is coded in or what platform the service runs on.

2.3 SOA: Implementation Challenges

SOA-based integration provides a consistent way to access all applications within a company, and potentially outside the company. However, the value of SOA has perhaps been oversold as a methodology and has often been mistakenly promoted as a technology that will solve almost all IT problems. In this section, we present common design requirements and implementation challenges to ensure an efficient SOA implementation. Implementation solutions to handle these requirements will be discussed in the following Web services and semantic Web services sections.

In order to enable dynamic and seamless cooperation between different systems and organizations, implementing SOA poses new challenges to overcome. The largest barriers to adoption of SOA tend to be establishing effective SOA standards fulfilling the expected promises. Understandably, SOA-based technologies are seeking implementation solutions to help them meet the above promises in the most cost-effective way. The challenge represented by implementing an SOA is the actual implementations may fail to reach some design requirements. More explicitly, these challenges concern mainly (i) the dynamic service interoperation without prior collaboration agreement, (ii) dynamic service discovery and selection based on requester needs, (iii) and automatic service composition to achieve added-value business requirements [1].

Implementation standards developing services in SOA have to consider not simply for immediate benefit, but also for long-term and board benefit. Unlike objects or databases, a service is developed for use by its consumer, which may not be known at the time. To put it in another way, the existence of an individual service is not of much interest unless it is part of a larger collection of services that can be consumed

by multiple applications, and out of which new services can be composed and executed. Any collection of services needs common design, discovery, composition, and binding principles since they are typically not all developed at the same time. Thus, any SOA implementation MUST take into account the following design requirements to ensure the described set of SOA benefits and promises:

- ***Service Discovery:*** Service description should be visible to be discovered and understood by service consumers. Visibility refers to the capacity for those with needs and those with capabilities to be able to see each other [3]. It is the relationship between service consumers and providers that is satisfied when they are able to interact with each other. This is typically done by providing descriptions for such aspects as functions and technical requirements, related constraints and policies, and mechanisms for access or response. This is true for any consumer/provider relationship – including in an application program where one program calls another: without the proper libraries being present the function call cannot complete. In the case of SOA, visibility needs to be emphasized because it is not necessarily obvious how service participants can see each other. Thus, the initiator in a service interaction must be aware of the other parties. Visibility is promoted through the service description which contains the information necessary to interact with the service and describes this in such terms as the service inputs, outputs, and associated semantics. The service description also conveys what is accomplished when the service is invoked and the conditions for using the service. The service description allows prospective consumers to decide if the service is suitable for their current needs and establishes whether a consumer satisfies any requirements of the service provider. The descriptions need to be in a form (or can be transformed to a form) in which their syntax and semantics are widely accessible and understandable. Discovering services need not necessarily be fully automated (one can find many non-technical objections to fully automated discovery), but support for some richer discovery is than necessary. The main challenge of service discovery is using automated means to accurately discover services with minimal user involvement. This requires explicating the semantics of both the service provider and requester. It also involves adding semantic annotations to service definitions. Achieving automated service discovery requires explicitly stating requesters' needs—most likely as goals that correspond to the description of desired services—in some formal request language.
- ***Service composition:*** SOA provides a new way of application development by composing services. The service-oriented paradigm builds on the notion of composing virtual components into complex behavior. Thus, a consumer can use the functionality offered by multiple providers without worrying about the underlying differences in hardware, operating systems, programming languages, etc. Each service should be designed to satisfy a business task while possibly collaborating with applications or services provided by other entities. For services to interact, they need not share anything but a formal contract that describes each service and defines the terms of information exchange. Therefore a service composition task

involves the selection, and interoperation of Web services given a high-level semantic description of an objective using a formal contract.

- ***Interoperability for semantic heterogeneity:*** SOA is applied in an environment where the number of involved actors is more and more heterogeneous and distributed. Consumers and providers communicate and exchange data which may lead to interoperability problems on top of the data mismatch (in structure and meaning). There are three important facts about services that set the basis for their information processing requirements. First, services, especially at the business level, exchange information in the form of messages. Secondly, the information in those messages needs to conform to the enterprise information model and semantics. Third, there is often a transformation between the enterprise semantics and the internal information model of the service. So, the SOA platform must provide: message processing capabilities, integration with existing enterprise information models, definition of messages based on the information model, and information transformation capabilities.

Unlike OOP paradigms, where the focus is on packaging data with operations, the central focus of SOA is the task or business function getting something done. This distinction manifests itself by the fact that an object exposes structure but there is no way to express semantics other than what can be captured as comments in the class definition. SOA emphasizes the need for clear semantics. Especially, in the case of service interaction where the message and information exchanges are across boundaries, a critical issue is the interpretation of the data. This interpretation must be consistent between the participants involved in service interaction. Consistent interpretation is a stronger requirement than merely type (or structural) consistency – the attributes of the data itself must also have a shared basis. For successful exchange of address information, all the participants must have a consistent view of the meaning of the address attributes. The formal descriptions of terms and the relationships between them provide a firm basis for making correct interpretations for elements of information exchanged. Note that, for the most part, it is not expected that service consumers and providers would actually exchange descriptions of terms during their interaction but, rather, would reference existing descriptions – the role of the semantics being a background one – and these references would be included in the service descriptions. Specific domain semantics are beyond the scope of SOA reference model; but there is a requirement that the service interface enable providers and consumers to identify unambiguously those definitions that are relevant to their respective domains [3].

3 Realizing SOA with Web Services

People often think of Web services and Service-Oriented Architecture (SOA) in combination, but they are distinct in an important way. As discussed in the previous section, SOA represents an abstract architectural concept. It's an approach to building software systems that is based on loosely coupled components (services) that have been described in a uniform way and that can be discovered and composed. Web

services represent one important approach to realizing SOA. The World Wide Web Consortium (W3C), which has managed the evolution of the SOAP and WSDL specifications, defines Web services as follows:

A software system designed to support interoperable machine-to machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with XML serialization in conjunction with other Web-related standards.

Although Web services technology is not the only approach to realizing an SOA, it is one that the IT industry as a whole has enthusiastically embraced. With Web services, the industry is addressing yet again the fundamental challenge that distributed computing has provided for some considerable time: to provide a uniform way of describing components or services within a network, locating them, and accessing them. The difference between the Web services approach and traditional approaches (for example, distributed object technologies such as the Object Management Group – Common Object Request Broker Architecture (OMG CORBA), or Microsoft Distributed Component Object Model (DCOM)) lies in the loose coupling aspects of the architecture. Instead of building applications that result in tightly integrated collections of objects or components, which are well known and understood at development time, the whole approach is much more dynamic and adaptable to change. Another key difference is that through Web services, the IT industry is tackling the problems using technology and specifications that are being developed in an open way, utilizing industry partnerships and broad consortia such as W3C and the Organization for the Advancement of Structured Information Standards (OASIS), and based on standards and technology that are the foundation of the Internet.

3.1 Scope of the Architecture

Web services had its beginnings in mid to late 2000 with the introduction of the first version of XML messaging—SOAP, WSDL 1.1, and an initial version of UDDI [4] as a service registry. This basic set of standards has begun to provide an accepted industry-wide basis for interoperability among software components (Web services) that is independent of network location, in addition to specific implementation details of both the services and their supporting deployment infrastructure. Several key software vendors have provided these implementations, which have already been widely used to address some important business problems.

Developers are looking for enhancements that raise the level and scope of interoperability beyond the basic message exchange, requiring support for interoperation of higher-level infrastructure services. Most commercial applications today are built assuming a specific programming model. They are deployed on platforms (operating systems and middleware) that provide infrastructure services in support of that programming model, hiding complexity, and simplifying the problems that the solution developer has to deal with. For example, middleware typically provides support for transactions, security, or reliable exchange of messages (such as guaranteed, once-only delivery). On the other hand, there is no universally agreed standard middleware, which makes it difficult to construct applications from

components that are built using different programming models (such as Microsoft COM, OMG CORBA, or Java 2 Platform, Enterprise Edition (J2EE) Enterprise Java Beans). They bring with them different assumptions about infrastructure services that are required, such as transactions and security. As a consequence, interoperability across distributed heterogeneous platforms (such as .NET and J2EE) presents a difficult problem.

The Web services community has done significant work to address this interoperability issue, and since the introduction of the first Web services, various organizations have introduced other Web services–related specifications. Fig. 2 illustrates a population of the overall SOA stack with current standards and emerging Web services specifications that IBM, Microsoft, and other significant IT companies have developed. The remainder of this part provides a high-level introduction to these Web services specifications that realize more concretely the capabilities that are described in the SOA framework.

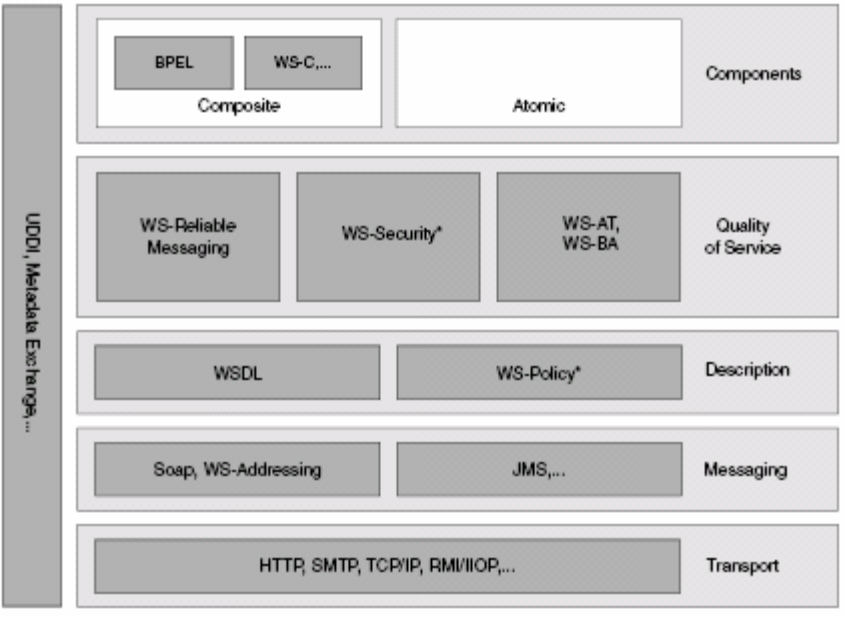

Fig. 2. Web services technologies

3.2 Web Service Transport

Web services are basically an interoperable messaging architecture, and message transport technologies form the foundation of this architecture. Web services are inherently transport neutral. Although you can transport Web services messages by using the ubiquitous Web protocols such as HyperText Transport Protocol (HTTP) or Secure HTTP (HTTPS) to give the widest possible coverage in terms of support for the protocols, you can also transport them over any communications protocol, using proprietary ones if appropriate. Although transport protocols are fundamental to Web

services and clearly are a defining factor in the scope of interoperability, the details are generally hidden from the design of Web services.

3.3 Web Service Messaging

The messaging services layer contains the most fundamental Web services specifications and technologies, including eXtensible Markup Language (XML), SOAP, and WS-Addressing [4]. Collectively, these specifications form the basis of interoperable messaging between Web services. XML provides the interoperable format to describe message content between Web services and is the basic language in which the Web services specifications are defined.

SOAP, one of the significant underpinnings of Web services, provides a simple and relatively lightweight mechanism for exchanging structured and typed information between services. SOAP is designed to reduce the cost and complexity of integrating applications that are built on different platforms.

WS-Addressing provides an interoperable, transport-independent way of identifying message senders and receivers that are associated with message exchange. WS-Addressing decouples address information from the specific transport used by providing a mechanism to place the target, source, and other important address information directly within the Web service message. This specification defines XML elements to identify Web services endpoints and to secure end-to-end endpoint identification in messages. This specification enables messaging systems to support message transmission through networks that include processing nodes such as endpoint managers, firewalls, and gateways in a transport neutral manner. WS-Addressing defines two interoperable constructs that convey information that transport protocols and messaging systems typically provide. These constructs normalize this underlying information into a uniform format that can be processed independently of transport or application. These two constructs are endpoint references and message information headers.

3.4 Web Service Description

Service description defines metadata that fully describes the characteristics of services that are deployed on a network. This metadata is important, and it is fundamental to achieving the loose coupling that is associated with SOA. It provides an abstract definition of the information that is necessary to deploy and interact with a service. Web Service Description Language (WSDL) [4] is perhaps the most mature of metadata describing Web services. It allows developers to describe the "functional" characteristics of a Web service—what actions or functions the service performs in terms of the messages it receives and sends. WSDL offers a standard, language-agnostic view of services it offers to clients. It also provides non-invasive future-proofing for existing applications and services and allows interoperability across the various programming paradigms, including CORBA, J2EE, and .NET.

3.5 Web Service Discovery

The Universal Description and Discovery Interface (UDDI) is a widely acknowledged specification of a Web service registry. It defines a metadata aggregation service and

specifies protocols for querying and updating a common repository of Web services information. Application developers can query UDDI repositories at well-known locations at design time to ascertain those services that might be compatible with their requirements. After they locate a directory, they can send a series of query requests against the registry to acquire detailed information about Web services (such as who provides them and where they are hosted) and bindings to the implementation. They can then feed this information into an assortment of development time tools to generate the appropriate runtime software and messages required to invoke the required service. Applications can also query UDDI repositories dynamically at runtime. In this scenario, the software that needs to use a service is told at execution time the type of service or interface it requires. Then it searches a UDDI repository for a service that meets its functional requirements, or a well-known partner provides it. The software then uses this information to dynamically access the service. Service discovery (publish/find) plays an important role in an SOA. It is possible to achieve this in other ways, but within a Web services world, UDDI provides a highly functional and flexible standard approach to Web service discovery.

WS-Policy proposes a framework that extends the service description features that WSDL provides. Having more refined service descriptions, qualified by specific WS-policies, supports much more accurate discovery of services that are compatible with the business application that is to be deployed. In a service registry (such as a UDDI registry), queries of WS-Policy-decorated services enable the retrieval of services that support appropriate policies in addition to the required business interface. For example, a query might request all services that support the credit Authorization WSDL interface (port type), use Kerberos for authentication, and have an explicitly stated privacy policy. This allows a service requester to select a service provider based on the quality of the interaction that delivers its business contracts.

3.6 Web Service Composition

Business Process Execution Language for Web services (WS-BPEL) [4] provides a language to specify business processes and how they relate to Web services. This includes specifying how a business process uses Web services to achieve its goal, and it includes specifying Web services that a business process provides. Business processes specified in BPEL are fully executable and are portable between BPEL-conformant tools and environments. A BPEL business process interoperates with the Web services of its partners, whether these Web services are realized based on BPEL or not. Finally, BPEL supports the specification of business protocols between partners and views on complex internal business processes. BPEL supports the specification of a broad spectrum of business processes, from fully executable, complex business processes over more simple business protocols to usage constraints of Web services. It provides a long-running transaction model that allows increasing consistency and reliability of Web service applications. Correlation mechanisms are supported that allow identifying statefull instances of business processes based on business properties. Partners and Web services can be dynamically bound based on service references.

3.7 SOA and Web Services: Need for Semantics

It is clear that Web services standards, both the core and extended specifications, contribute significantly to the ability to create and maintain service-oriented architectures on which to build new enterprise applications. However, despite their success, there still remain important challenges to be addressed in current SOA-based solutions. Service discovery, interoperation and composition are typically part of any development process based on SOA but, nothing is prescribed for effectively supporting these activities.

Discovery: To discover a Web service, the infrastructure should be able to represent the capabilities provided by a Web service and it must be able to recognize the similarity between the capabilities provided and the functionalities requested.

UDDI is the most well-known specification for an XML-based registry of service descriptions on the Web but the descriptions are syntactic only - the meaning is still open to interpretation by the user. Indeed, companies adopting UDDI for internal use have to define their own naming conventions and categorization structure and metadata, which inhibit adoption. Currently keyword-based search is the only means of finding relevant services. UDDI does not allow capability-based discovery of Web services. Support for some richer discovery than keyword-based search is necessary. Semantics bring closer the possibility of switching services dynamically by discovering them at runtime.

WSDL is less suitable for describing the semantics of a Web Service capability. This drawback affects not only the service discovery procedure but also service composition, invocation and interoperation. Indeed, WSDL files contain no information on the semantics of the described operations. It is up to the programmer or engineer to interpret the semantics from available descriptions in natural language. This type of interpretation becomes a challenge because the human factor inhibits automation of service discovery, selection, invocation and composition. Additionally, natural language descriptions are informal and can lead to different interpretations or even to failure to understand. This challenge can be overcome if formal and declarative semantic mark-up is used to complement service descriptions.

Composition: Similarly, service composition is mainly based on the syntactic descriptions provided by WSDL, which are necessary but not sufficient, since the semantics remain implicit and cannot be automatically processed. A number of approaches exist for modeling Web Service composition. Although these Web service composition languages are more suitable than the proprietary languages used in traditional workflow products, they lack the possibility to dynamically bind to Web services at run time. For example, WSCI and WS-BPEL describe how multiple Web services could be composed together to provide a more complex Web service. However, their focus remains on composition at the syntactic level and therefore, does not allow for automatic composition of Web services.

Service requesters have to bind specific services at design time which means they cannot take advantage of the large and constantly changing amount of Web services available. The services have to interoperate with each other seamlessly so that the combined results are a valid solution. Web services must be described and understood in a semantically consistent way in order to resolve terminological ambiguities and

misunderstandings, and to avoid the constant revision and redefinition of terms, concepts, and elements of the business. Such inconsistencies make applications not able to talk to each other, and subsequently result in slower response times when changes are needed. Business managers cannot get a clear view of their organization through these multiple un-integrated "languages".

Interoperation: The current Web services infrastructure focuses on syntactic interoperability. Syntactic interoperability allows Web services to identify only the structure of the messages exchanged, but it fails to provide an interpretation of the content of those messages. Indeed, Current standards like XML and XML Schema only solve the mismatch on the syntactical and structural level; solving the mismatch on the semantic level is usually handled on a case-by-case basis (for instance using custom adapters). Mismatches between interaction protocols are not dealt within current standards; semantics of the message exchange sequences are necessary to solve the mismatches on that level.

4 Realizing SOA with Semantic Web Services

4.1 Introduction

Web Services technology based on WSDL, SOAP and UDDI, define common standards that ensure interoperability between heterogeneous platforms. However, although low level interoperability is essential, SOA challenges as discussed in section 1 go beyond data formats and communication protocols interoperability. The purely syntactic focus of WS technologies makes service description non interpretable by the machine which hampers the automation of operations, inherent to SOA, such as service discovery, composition and invocation.

SWS initiatives have emerged with the objective of complementing the inter-operability ensured by Web services to deal with data and behavioral heterogeneity along with automation support for capability-based service discovery, and dynamic service composition and invocation. The basic and common principle of these initiatives is extending syntactic service descriptions with a semantic layer the machine can interpret and reason over it. Ontologies play a central role for defining this semantic extension. An ontology is a formal explicit specification of a shared conceptualization [5]. Ontologies define a common vocabulary and formal semantics by providing concepts, and relationships between them. Using a common vocabulary for describing services capability and behaviors ensures interoperability at data level. Formal semantics enables the application of powerful and well proven reasoning based techniques in order to enable capability-based service discovery and automatic service composition.

There are four main SWS initiatives namely WSMO/L/X Framework [6], OWL-S [7], IRS-III framework [8] and METEOR-S system [9]. The first three initiatives separate explicitly between the semantic and syntactic descriptions of a Web service and link them using the concept of grounding that maps abstract concepts and data types of the semantic description to concrete data formats and communication protocols at the

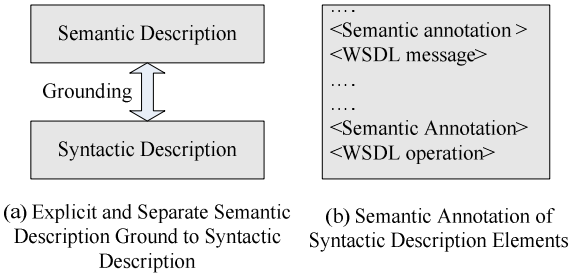

Fig. 3. Two approaches to semantically extend syntactic Web service descriptions

syntactic level (see Fig. 3 (a)). METOER-S, however, semantically annotate WSDL files by linking their elements to ontology concepts and relations (see Fig. 3 (b)).

While METEOR-S is agnostic as regards to the ontologies used for the semantic annotation, WSMO/L/X, OWL-S and IRS-III can be seen as fully fledged framework with three layers (see Fig. 4): (i) a conceptual model for describing Web services and related information, (ii) a formal language used for defining the conceptual model concepts, relations and axioms, and (iii) an execution environment, as a proof of concepts, showing the use of semantic description for carrying out goal-based service discovery and invocation, and automatic service composition. In the following, we present the conceptual model of each of these initiatives and give an overview of their execution environments.

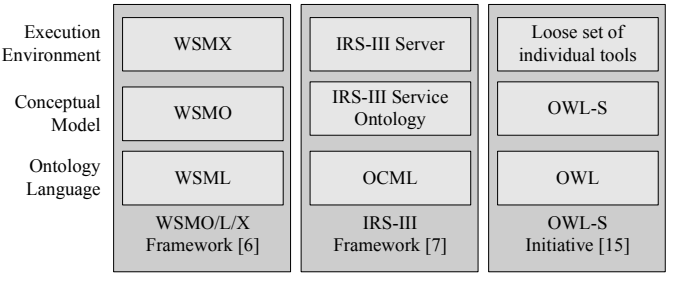

Fig. 4. WSMO/L/X, IRS-III and OWL-S constituent layers

4.2 WSMO/L/X Framework

WSMO [10] is an ontological conceptual model for describing various aspects related to SWS. WSMO refines and extends the Web Service Modeling Framework (WSMF) [11], by developing a set of formal ontology languages. WSMF is based on two complementary principles that WSMO inherits: strong decoupling between the various resources and a strong mediation to ensure the interoperation between these loosely coupled components. While WSMO provides the conceptual model for describing core elements of SWS, WSML [12] provides a formal language for writing, storing and communicating such descriptions.

4.2.1 Conceptual Model: WSMO

Following the main concepts identified in the WSMF, WSMO identifies four top level elements as the main concepts for describing several aspects of SWS, namely ontologies, Web services, goals and mediators (see Fig. 5).

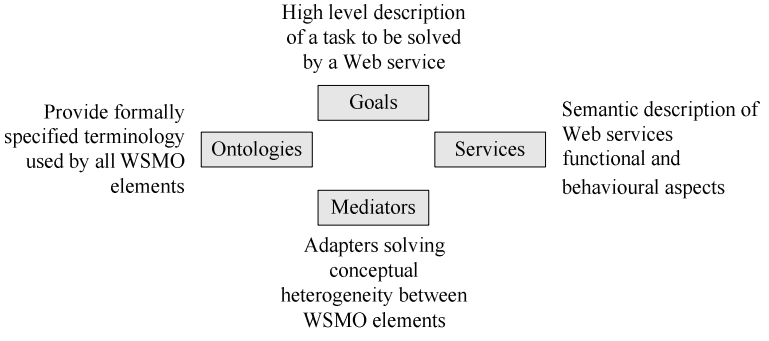

Fig. 5. WSMO top level elements [10]

Ontologies are used as the data model throughout WSMO. All resource descriptions as well as all data interchanged during service usage are based on ontologies. The core elements of an ontology are concepts (the basic entities of the agreed terminology), relations (model interdependencies between several concepts, and instances), instances, and axioms (define complex logical relations between the other elements defined in the ontologies) [6].

WSMO Service description consists of non-functional, functional, and behavioral aspects [10]. A service capability describes the provided functionality. A capability is described in terms of preconditions, assumptions, postconditions and effects. A service interface describes the behavioral aspects of the service in terms of choreography and orchestration. A service choreography details how to interact with the service from a user's perspective. An orchestration describes how the service works from the provider's perspective [10].

A WSMO goal is a high level description of a task required to be solved by Web services. Similar to a WSMO service, a goal consists of non functional properties, a capability describing the user objective and an interface reflecting the user behavior requirements.

Mediation in WSMO aims at resolving mismatches that may arise between different used terminologies (data level), or interaction protocols (process level). WSMO ensures dynamic interoperability by defining mediators during design time that will be used by mediation components during run time to resolve heterogeneity on the fly. A WSMO mediator can be seen as an adapter between WSMO elements defining the necessary mappings and transformations between the linked elements [6]. WSMO defines four types of mediators: OO mediators that resolve terminological mismatches between two ontologies, GG, WG and WW mediators that resolve mismatches respectively between two goals, a service and a goal, and two Web services.

4.2.2 Execution Environment: WSMX

WSMX [13] is an execution environment for dynamic discovery, selection, mediation, invocation and inter-operation of SWS. WSMX is the reference implementation of WSMO and therefore relies on it as conceptual model. A provider can register its service using WSMX in order to make it available to the consumers. A requester can find the Web Services that suit their needs and then invoke them in a transparent way [6].

WSMX exploits semantic service description to support capability-based discovery, not possible to perform having pure syntactic service description. In addition to the classical keyword-based discovery, WSMX supports functional, instance based and Quality of Service (a.k.a QoS) based discovery. Functional discovery reasons over service capabilities by matching them to the user goal capability. WSMX distinguishes different degrees of matching with the required goal [6]. Instance-based discovery considers instance level service descriptions and can dynamically fetch additional information during the discovery process. A Quality of Service based discovery provides a framework which matches specific QoS requirements of the requester with provided SWS.

WSMX implements data and process mediation as distinct components. The Data Mediation component in WSMX deals with heterogeneity problems that can appear at data level. Process level mediation deals with solving interaction protocols mismatches. Both components handle heterogeneity problems by applying the set of mappings rules, between the source and target WSMO element, defined during design-time.

WSMX conceptual architecture defines a distinct component for composition. However, no automatic composition is implemented yet as part of WSMX. Nevertheless, WSMO provides the required foundation for automatic service composition. Indeed, SUPER project [14] has released a composer component enabling automatic WSMO service composition by applying Artificial Intelligence (a.k.a AI) planning techniques.

4.3 OWL-S Initiative

4.3.1 Conceptual Model: OWL-S

OWL-S [7, 15] is an upper ontology for service description based on the Web Ontology Language (OWL) [16]. As shown in Fig. 6, an OWL-S service description consists in three interrelated parts: the service profile, the process model and the grounding. The service profile is used to describe what the service does; the process model is used to describe how the service is used; and the grounding is used to describe how to interact with the service. The service profile and process model are

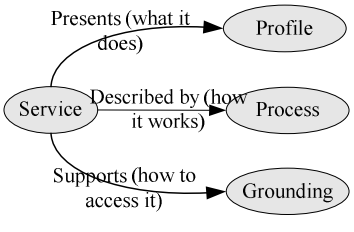

Fig. 6. Top level elements of OWL-S service ontology [7]

abstract descriptions of a service, whereas the grounding specifies how to interact with it by providing the concrete details related to message formats, and communication protocols.

A service profile describes functional, classification and non functional aspects of a service. Similar to WSMO, the capability of a Web service is represented as a transformation from the inputs and the preconditions of the Web service to the set of outputs produced, and the effects that may result from the execution of the service [15]. The classification aspect describes the type of service as specified in a domain-specific taxonomy. Non-functional aspects include service parameters like security, privacy requirements, and Quality of Service properties. OWL-S provides an extensible mechanism that allows the providers and the consumers to define additional service parameters.

The process model provides a more detailed view on how the service is carried out in terms of control and data flow. OWL-S distinguishes between atomic, composite and simple processes. An atomic process corresponds to a single interchange of inputs and outputs between a consumer and a provider. A composite process consists of a set of component processes linked together by control flow and data flow structures. The control flow is described using programming language or workflow constructs such as sequences, conditional branches, parallel branches, and loops. Data flow is the description of how information is acquired and used in subsequent steps in the process [15]. Simple processes can be used to provide abstracted, non invocable views of atomic or composite processes.

The grounding specifies the details of how a service can be accessed. Service grounding allows separating the abstract information described by the process model from the implementation details. Fig. 7 illustrates how the grounding is achieved in OWL-S. It specifies mapping atomic processes into WSDL operations. In addition, it specifies how to translate the messages described as OWL classes and instances to WSDL messages.

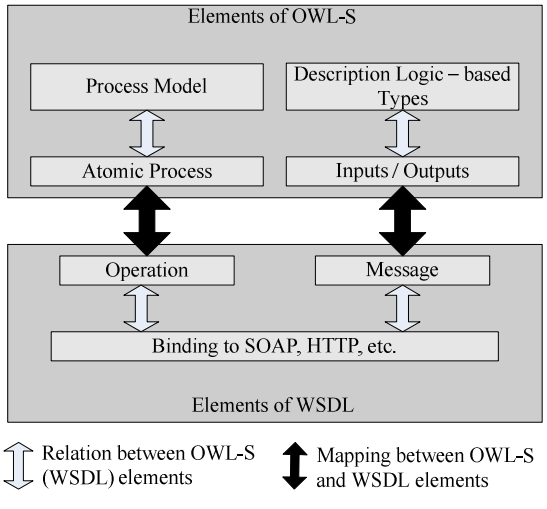

Fig. 7. Grounding in OWL-S [15]

4.3.2 OWL-S Tools

Unlike WSMO, OWL-S does not have a reference implementation like WSMX. Instead, there exists a collection of individual tools like OWL-S Editor [17], OWL-S/UDDI Matchmaker [18], OWL-S Virtual Machine [19], WSDL2OWL-S converter [20] and OWL-S2UDDI converter [18]. In the following we focus on the tools enabling dynamic service discovery and interoperation, and automatic service composition.

The OWL-S/UDDI matchmaker integrates OWL-S capability matching into the UDDI registry. OWL-S2UDDI converter converts OWL-S profile descriptions into corresponding UDDI advertisements, which can then be published in a UDDI registry. The OWL-S/UDDI registry enhances UDDI registry with OWL-S matchmaking functionalities. The matching engine contains five different filters for namespace comparison, word frequency comparison, ontology similarity matching, ontology subsumption matching, and constraint matching [6].

The OWL-S Virtual Machine enables to control the interaction between Web services according to their process models. Unlike WSMO and WSMX, OWL-S conceptual model and implementation do not consider mediators as first class citizens. OWL-S assumes the existence of external mechanisms that can handle heterogeneity at data and process level [6].

Several approaches have been proposed for automatic service composition based on OWL-S description [15]. [21] considers OWL-S process model as abstract workflow which is expanded and refined using automated reasoning machinery. [22, 23] use Hierarchical Task Network (HTN) planning to perform automated Web Service composition. Other planning techniques that have been applied to the composition of OWL-S services, are classical STRIPS-style planning [24], extended estimated-regression planning [25], and Planning as Model Checking [26].

4.4 IRS-III Framework

IRS-III (Internet Reasoning Service) is a framework for creating and executing SWS [8]. It acts as a semantic broker between a client application and deployed Web services by supporting capability-based invocation. A client sends a request encapsulating the desired goal and, by exploiting the semantic description of Web services, IRS-III framework: (a) discovers potentially relevant Web services; (b) selects the set of Web services which best fit the incoming request; (c) mediates any mismatches at the conceptual level; and (d) invokes the selected Web services whilst adhering to any invocation constraints.

4.4.1 Conceptual Model: Service Ontology

IRS-III service ontology defines the conceptual model of IRS-III framework. It extends the core epistemological framework of its previous IRS-II framework [27] by incorporating WSMO conceptual model. Different from WSMO, IRS-III service ontology uses its own ontology language, OCML [28]. While there are some differences between IRS-III and WSMO conceptual models, IRS-III service ontology defines the same concepts for describing SWS namely goals, Web service capability and interface (choreography and orchestration), and mediators.

4.4.2 Execution Environment: IRS-III Server

IRS-III server is the main element of IRS-III framework handling capability-based discovery and dynamic invocation. IRS-III framework includes also IRS-III publishing platform, IRS-III browser and IRS-III API. The publishing platform enables ease publication and deployment of Web services. IRS-III browser provides a goal-centric invocation mechanism to end users. IRS-III API facilitates the integration of IRS-III framework with other SWS platforms.

Similar to WSMX, IRS-III mediation approach consists of defining mediators which provide declarative mappings for solving different types of conceptual mismatches. These mediator models are created at design time and used at runtime. The mediation handler (part of IRS-III server) interprets each type of mediator accordingly during selection, invocation and orchestration of Web services [8].

Like WSMX, IRS-III does not support explicitly automatic service composition; however, it shares with it the same results of SUPER project about automatic service composition by applying AI planning techniques since both of them rely on the same conceptual model WSMO.

4.5 METEOR-S

METEOR-S project addresses the usage of semantics to support the complete lifecycle of Semantic Web processes [9] using four kinds of semantics - data, functional, non-functional and execution semantics. The data semantics describe the data (inputs/outputs) of the Web services. The functional semantics describe the functionality of a Web services (what it does). The non-functional semantics describe the non-functional aspects like Quality of Service and business rules. The execution semantics model the behavior of Web services and processes. Unlike above initiatives, METEOR-S does not define a fully fledged conceptual model for SWS description. It rather follows a light-weight approach by extending WSDL files with semantic annotation. The semantic annotation is achieved by mapping WSDL elements to ontological concepts. WSDL-S [29], METEOR-S specification for WSDL annotation, was one of the main works that influenced SAWSDL [30] the W3C standard for WSDL and XML schema semantic annotation.

METEOR-S framework provides a tool for creating SWS [31], a publication and discovery module [32], a composition module [33] and an execution environment. The GUI based tool provides support for semi-automatic and manual annotation of existing Web services or source code with domain ontologies. The publication and discovery module provides support for semantic publication and discovery of Web services. It provides support for discovery in a federation of registries as well as a semantic publication and discovery layer over UDDI. The composition module consists of two main sub-modules - the constraint analysis and optimization sub-module that deal with correctness and optimization of the process on the basis of QoS constraints. METEOR-S framework doesn't define components dedicated to deal with data and behavioral heterogeneity problems that may arise during services interactions.

4.6 SOA and Semantic Web Services: A Step Forward

The objective of SWS initiatives is providing the means to automate capability-based service discovery, service composition and invocation. The main idea is extending

syntactical service description with additional information which can be understood and processed by the machine. Ontologies play a central role in the semantic service description. Using ontologies does not only bring user requirements and service advertisements to common conceptual space, but also helps to apply reasoning mechanisms [9]. Thus software programs are able to understand service descriptions and reason over them.

Indeed, as regards to **service discovery** ontology-enhanced search engine can exploit semantic service descriptions to implement matchmaking techniques, much more powerful than keyword-based ones, based on information retrieval, AI, and software engineering to compute both the syntactical and semantic similarity among service capability descriptions. Regarding **dynamic service interoperation**, software agents can leverage the computer-interpretable service description to understand what input is necessary to the service call, what information will be returned, and how to execute the service automatically. Mediation is a pillar for solving heterogeneity problems on the fly. Data and process mediators have been recognized as first class citizens within WSMO and IRS-III. Concerning **dynamic service composition**, semantic markup of Web services provides the necessary information to select and compose services. Software programs, based on AI planning, software synthesis and model checking, can be written to manipulate these representations, together with a specification of the objectives of the task, to achieve the task automatically [7].

In spite of the undeniable advancement realized by SWS, some issues still remain to be addressed. Indeed the great success of SWS is due to their semantic descriptions that rely on ontologies. However, service providers and requesters may use different terminologies to describe their requirements and services. Therefore, mediation is a key point in all SWS operations (discovery, composition, invocation) in order to resolve the terminological mismatches. Mediators can be seen as adapters at an ontological level enabling to migrate from one conceptualization to another. Consequently SWS initiatives have the same drawbacks as any adapter-based solution. Mediators are often defined at design time manually. In addition mapping between two conceptualizations is not always straightforward. Furthermore, mediators must be maintained each time one or both of the involved ontologies change. Another problems SWS face concerns the computing complexity, especially in terms of response time, of machine reasoning techniques which hamper the application of these techniques in context where soft real-time response is required for user interactions. Furthermore, plans generated by AI planning techniques are relatively simple compared to real composition models, defined by BPEL for instance.

5 Conclusion

In this chapter, we discussed how far Web services technologies and SWS initiatives satisfy SOA requirements. Loose coupling between service provision and consumption has led to new challenges for ensuring seamless and cost effective integration. These challenges concern effective service discovery, dynamic service interoperation, and automation support for service composition. By defining a set of standards, Web services technologies ensure low level interoperability, an essential first step yet not enough. Indeed, human intervention is still heavily required to

resolve data and behavioral mismatches, to find the right services, and to select and compose appropriate services. SWS initiatives extend the level of interoperability to deal with data and behavioral heterogeneity using the concept of mediation. They also provide the foundation for (i) capability-based service discovery which is much more efficient than keyword-based one, (ii) dynamic service interoperation and (iii) automatic service composition. In spite of the remarkable results achieved by these initiatives, some open issues still need to be resolved. These issues are mainly related to mediator definition and update, and the computing complexity of machine reasoning techniques.

Acknowledgments. This work was supported by the Lion project supported by Science Foundation Ireland under grant no. SFI/02/CE1/I131, and by SUPER project funded by the EU under grant no. FP6-026850.

References

1. Erl, T.: Service-Oriented Architecture (SOA): Concepts, Technology, and Design. Prentice-Hall, Englewood Cliffs (2005)
2. Service Oriented Architecture Modeling, http://www.soamodeling.org
3. MacKenzie, C.M., Laskey, K., McCabe, F., Brown, P.F., Metz, R., Hamilton, B.A.: OASIS Reference Model for Service Oriented Architecture V 1.0, http://www.oasis-open.org/committees/download.php/19361/soa-rm-cs.pdf
4. Weerawarana, S., Curbera, F., Leymann, F., Storey, T., Ferguson, D.F.: Web services platform architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More. Prentice-Hall, Englewood Cliffs (2005)
5. Gruber, T.R.: A translation approach to portable ontology specifications. Knowledge Acquisition 5, 199–220 (1993)
6. Roman, D., de Bruijn, J., Mocan, A., Lausen, H., Bussler, C., Fensel, D.: WWW: WSMO, WSML, and WSMX in a nutshell. In: 1st Asian Semantic Web Conference, pp. 516–522. Springer, Beijing (2006)
7. Martin, D., Burstein, M., McDermott, D., et al.: OWL-S 1.2 Release, http://www.daml.org/services/owl-s/1.2/
8. Domingue, J., Cabral, L., Galizia, S., Tanasescu, V., Gugliotta, A., Norton, B., Carlos, P.: IRS-III: A broker-based approach to semantic Web services. J. Web Sem. 6(2), 109–132 (2008)
9. Verma, K., Gomadam, K., Sheth, A.P., Miller, J.A., Wu, Z.: The METEOR-S Approach for Configuring and Executing Dynamic Web Processes. LSDSIS technical report, http://lsdis.cs.uga.edu/projects/meteor-s/
10. Roman, D., Lausen, H., Keller, U., et al.: Web Service Modelling Ontology, http://www.wsmo.org/TR/d2/v1.4/
11. Fensel, D., Bussler, C.: The Web Service Modeling Framework (WSMF). Electronic Commerce Research and Applications 1(2), 113–137 (2002)
12. Steinmetz, S., Toma, I.: Web Service Modeling Language, http://www.wsmo.org/TR/d16/d16.1/v1.0/
13. Shafiq, O., Moran, M., Cimpian, E., Mocan, A., Zaremba, M., Fensel, D.: Investigating Semantic Web Service Execution Environments: A comparison between WSMX and OWL-S tools. In: 2nd International Conference on Internet and Web Applications and Services. IEEE Computer Society, Mauritius (2007)

14. Semantic Utilised for Process Management within and between Enterprises, http://www.ip-super.org
15. David, L., Martin, D.L., Burstein, M.H., McDermott, D.V., McIlraith, S.A., Paolucci, M., Sycara, K.P., McGuinness, D.L., Sirin, E., Srinivasan, N.: Bringing Semantics to Web Services with OWL-S. World Wide Web Journal 10(3), 243–277 (2007)
16. McGuinness, D.L., van Harmelen, F.: OWL Web Ontology Language Overview, http://www.w3.org/TR/2004/REC-owl-features-20040210/
17. Elenius, D., Denker, G., Martin, D., et al.: The OWL-S Editor—A development tool for semantic web services. In: 2nd European Semantic Web Conference, pp. 78–92. Springer, Heraklion (2005)
18. Srinivasan, N., Paolucci, M., Sycara, K.: An efficient algorithm for OWL-S based semantic search in UDDI. In: Cardoso, J., Sheth, A.P. (eds.) SWSWPC 2004. LNCS, vol. 3387, pp. 96–110. Springer, Heidelberg (2005)
19. Paolucci, M., Ankolekar, A., Srinivasan, N., et al.: The DAML-S virtual machine. In: 2nd International Semantic Web Conference, pp. 335–350. Springer, Sanibel Island (2003)
20. Paolucci, M., Srinivasan, N., Sycara, K., et al.: Toward a semantic choreography of web services: From WSDL to DAML-S. In: 1st International Conference on Web Services, pp. 22–26. CSREA Press, Las Vegas (2003)
21. McIlraith, S., Son, T.: Adapting golog for composition of semantic web services. In: 8th International Conference on Principles of Knowledge Representation and Reasoning, pp. 482–493. Morgan Kaufmann, Toulouse (2002)
22. Sirin, E., Parsia, B., Wu, D., et al.: HTN Planning for Web Service Composition using SHOP2. Journal of Web Semantics 1(4), 377–396 (2004)
23. Nau, D., Au, T.C., Ilghami, O., et al.: SHOP2: An HTN planning system. J. Artif. Intell. 20, 379–404 (2003)
24. Sheshagiri, M., desJardins, M., Finin, T.: A planner for composing services described in DAML-S. In: Workshop on Web Services and Agent-Based Engineering, Melbourne (2003)
25. McDermott, D.: Estimated-regression planning for interactions with web services. In: 6th International Conference on Artificial Intelligence Planning Systems, Toulouse (2002)
26. Sirin, E., Parsia, B., Hendler, J.: Filtering and selecting semantic web services with interactive composition techniques. IEEE Intell. Syst. 19(4), 42–49 (2004)
27. Motta, J., Domingue, L., Cabral, M.: IRS-II: a framework and infrastructure for semantic Web services. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 306–318. Springer, Heidelberg (2003)
28. Motta, E.: An overview of the OCML modelling language. In: 8th Workshop on Knowledge Engineering Methods and Languages, Karlsruhe, pp. 21–22 (1998)
29. Miller, J., Verma, K., Rajasekaran, P., Sheth, A., Aggarwal, R., Sivashanmugam, K.: WSDL-S: Adding Semantics to WSDL - White Paper, http://lsdis.cs.uga.edu/library/download/wsdl-s.pdf
30. Kopecky, J., Vitvar, T., Bournez, C., Farrell, F.: SAWSDL: Semantic Annotations for WSDL and XML Schema. IEEE Internet Computing 11, 60–67 (2007)
31. Patil, A., Oundhakar, S., Sheth, A., Verma, K.: METEOR-S Web service Annotation Framework. In: 13th International World Wide Web Conference, pp. 17–22 (2004)
32. Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S., Miller, J.: METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services. J. Inf. Technol. and Management 6(1), 7–39 (2005)
33. Cardoso, J., Sheth, A.O.: Semantic E-Workflow Composition. J. Intell. Inf. Syst. 21(3), 91–225 (2003)