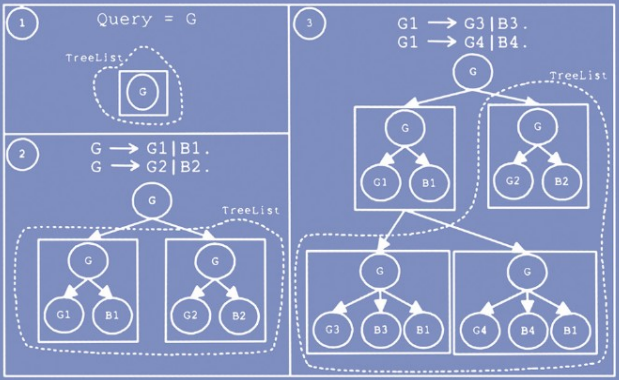
Hot Topics

LNCS 3387

Jorge Cardoso
Amit Sheth (Eds.)

Semantic Web Services and Web Process Composition

First International Workshop, SWSWPC 2004
San Diego, CA, USA, July 2004
Revised Selected Papers



Springer

Jorge Cardoso   Amit Sheth (Eds.)

Semantic Web Services and Web Process Composition

First International Workshop, SWSWPC 2004
San Diego, CA, USA, July 6, 2004
Revised Selected Papers

Springer

Volume Editors

Jorge Cardoso
Universidade da Madeira, Departamento de Matemática e Engenharias
Funchal, 9000-390 Portugal
E-mail: jcardoso@uma.pt

Amit Sheth
University of Georgia, LSDIS Lab, Computer Science Department
415 Boyd GSRC, DW Brooks Dr., UGA, Athens, GA 30602-7404, USA
E-mail: amit@cs.uga.edu

Library of Congress Control Number: 2004117658

CR Subject Classification (1998): H.3, H.4, H.2, I.2, H.5, C.2

ISSN 0302-9743
ISBN 3-540-24328-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11376736 06/3142 5 4 3 2 1 0

Preface

This book constitutes the refereed proceedings of the 1st International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004, held at the Westin Horton Plaza Hotel, San Diego, California, USA, July 6, 2004, in conjunction with the IEEE International Conference on Web Services (ICWS 2004).

The workshop intended to bring researchers, scientists from both industry and academics, and representatives from different communities together to study, understand, and explore the phases that compose the lifecycle of Semantic Web processes. The workshop presented what can be achieved by the symbiotic synthesis of two of the hottest R&D and technology application areas, Web services and the Semantic Web, as recognized at the 12th International World Wide Web conference (WWW 2003) and in the industry press.

The emphasis of the workshop was mainly on Web services, Web processes and semantics which are important movements emerging in the World Wide Web. Web services and Web processes promise to ease several current infrastructure challenges, such as data, application, and process integration. Web services are truly platform-independent and allow the development of distributed, loosely coupled applications, a key characteristic for the success of dynamic Web processes.

The 9 revised full papers presented were carefully reviewed and selected from 20 submissions, after a double-blind review process. In addition, we were honored by the presence of two distinguished invited speakers, namely Prof. Munindar Singh (North Carolina State University, USA) and Prof. Boualem Benatallah (University of New South Wales, Australia). The workshop also included a panel entitled "Do Academic Research and Industry Differ in the Role and Approach to the Use of Semantics for Web Processes?" where John Miller (University of Georgia), Jeff Pollock (Network Inference Ltd., USA), Jianwen Su (UC, Santa Barbara, USA), Ryusuke Masuoka (Fujitsu, USA), and Shishir Garg (France Telecom, France) participated.

We would like to express our sincere gratitude to all the authors, who provided the rich material discussed at the workshop, and the members of the Program Committee who reviewed and assessed the scientific merit of each submitted paper, thus ensuring high quality standards.

July 2004

Jorge Cardoso
Amit Sheth

Organization

SWSWPC 2004 was organized by the Department of Mathematics and Engineering, University of Madeira, Portugal and by the Department of Computer Science, University of Georgia, USA, in cooperation with the 2004 IEEE International Conference on Web Services (ICWS 2004).

Program Committee

Conference Chair	Jorge Cardoso (University of Madeira, Portugal)
Program Chairs	Jorge Cardoso (University of Madeira, Portugal)
	Amit Sheth (University of Georgia, USA)
Organizing Chairs	Jorge Cardoso (University of Madeira, Portugal)
	Amit Sheth (University of Georgia, USA)
	Leonid A. Kalinichenko (Russian Academy of Sciences, Russia)
	Francisco Curbera (IBM, USA)

Referees

R. Akkiraju
C. Bussler
J. Cardoso
F. Curbera
S. Damodaran
A. Dogac
S. Garg
L. Kalinichenko
M. Little
R. Masuoka
J. Mischkinsky
M. Nez
L. Obrst
M. Paolucci
D. Plexousakis
A. Sheth
S. Staab
R. Studer
S. Thatte
K. Verma

Sponsoring Institutions

Network Inference, Inc., USA (http://www.networkinference.com/)

Table of Contents

Introduction to Semantic Web Services and Web Process Composition

Jorge Cardoso[1] and Amit Sheth[2]

[1] Departement of Mathematics and Engineering,
University of Madeira, Funchal, Portugal
jcardoso@uma.pt

[2] Large Scale Distributed Information Systems (LSDIS) Lab,
Department of Computer Science,
University of Georgia, GA, USA
amit@cs.uga.edu

Abstract. Systems and infrastructures are currently being developed to support Web services. The main idea is to encapsulate an organization's functionality within an appropriate interface and advertise it as Web services. While in some cases Web services may be utilized in an isolated form, it is normal to expect Web services to be integrated as part of Web processes. There is a growing consensus that Web services alone will not be sufficient to develop valuable Web processes due the degree of heterogeneity, autonomy, and distribution of the Web. Several researchers agree that it is essential for Web services to be machine understandable in order to support all the phases of the lifecycle of Web processes. This paper deals with two of the hottest R&D and technology areas currently associated with the Web — Web services and the Semantic Web. It presents how applying semantics to each of the steps in the Semantic Web Process lifecycle can help address critical issues in reuse, integration and scalability.

1 Introduction

E-commerce and e-services have been growing at a very fast pace. The Web coupled with e-commerce and e-services is enabling a new networked economy [1]. The scope of activities that processes span has moved from intra-enterprise workflows, predefined inter-enterprise and business-to-business processes, to dynamically defined Web processes among cooperating organizations.

There is a remarkable range for growth in trade through electronic interactions, simply because it can eliminate geographical distances in bringing buyers and sellers together. With the Internet dissemination and the e-commerce growth there is a shift from the traditional off-line distribution process based on organization's catalogs to on-line services. A shift that is marked by isolated initiatives guided by the business-to-customer and business-to-business promise of increased profit margins and reduced commission values. This leads us to the present situation where we can find diverse and numerous groups of on-line systems, most of them focused in one or in a few types of products. Therefore, organizations are increasingly faced with the challenge of managing e-business systems and e-commerce applications managing Web services, Web processes, and semantics. Web services promise universal interoperability and integration. The key to achieving this relies on the efficiency of discovering appropriate Web services and composing them to build complex processes. We will start this section by explaining what semantics are and their role and relationships with ontologies. We then explain the purpose of each of the Web process lifecycle phases.

2 Semantic Web Process Lifecycle

Semantic Web services will allow the semi-automatic and automatic annotation, advertisement, discovery, selection, composition, and execution of inter-organization business logic, making the Internet become a global common platform where organizations and individuals communicate among each other to carry out various commercial activities and to provide value-added services.

In order to fully harness the power of Web services, their functionality must be combined to create Web processes. Web processes allow representing complex interactions among organizations, representing the evolution of workflow technology. Semantics can play an important role in all stages of Web process lifecycle. The main stages of the Web process lifecycle are illustrated in Figure 1.

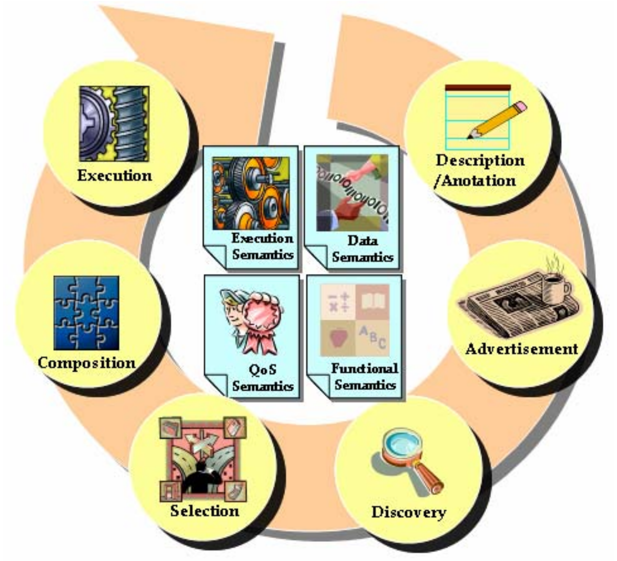


Fig. 1. Web process lifecycle and semantics.

The lifecycle of semantic Web processes includes the description/annotation, the advertisement, the discovery, the selection, the composition of Web services that makeup Web processes, and the execution of Web processes. All these stages are significant for the Web process lifecycle and their success.

2.1 Semantics and Ontologies

There is a growing consensus that Web services alone will not be sufficient to develop valuable and sophisticated Web processes due the degree of heterogeneity, autonomy, and distribution of the Web. Several researchers agree that it is essential for Web services to be machine understandable in order to allow the full deployment of efficient solutions supporting all the phases of the lifecycle of Web processes.

The idea and vision of the "Semantic Web" [2] catches on and researchers as well as companies have already realized the benefits of this great vision. Ontologies [3] are considered the basic building block of the Semantic Web as they allow machine supported data interpretation reducing human involvement in data and process integration.

An ontology "is a formal, explicit specification of a shared conceptualization. *Conceptualization* refers to an abstract model of phenomena in the world by having identified the relevant concepts of those phenomena. *Explicit* means that the type of concepts used, and the constraints on their use are explicitly defined. *Formal* refers to the fact that the ontology should be machine readable. *Shared* reflects that ontology should capture consensual knowledge accepted by the communities" [4].

When the knowledge about a domain is represented in a declarative language, the set of objects that can be represented is called the universe of discourse. We can describe the ontology of a program by defining a set of representational terms. Definitions associate the names of entities in the universe of discourse (e.g. classes, relations, functions or other objects) with human-readable text describing what the names mean and formal axioms that constrain the interpretation and well-formed use of these terms.

A set of Web services that share the same ontology will be able to communicate about a domain of discourse. We say that a Web service commits to an ontology if its observable actions are consistent with the definitions in the ontology.

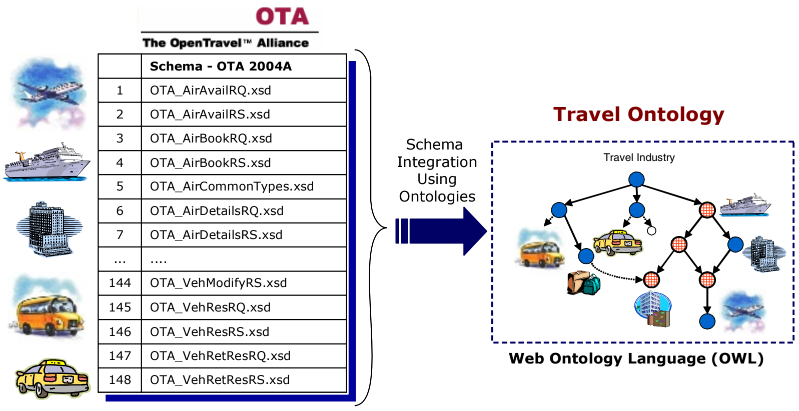
Example: Benefits of Ontologies for the Travel Industry. The Web has permanently changed the manner travel packages can be created. Consumers can now acquire packages from a diversity of Web sites including online agencies and airlines. With the spread of Web travel, a new technology has surfaced for the leisure travel industry: dynamic packaging. For the development of dynamic packaging solutions it is necessary to look in detailed at the technology components needed to enhance the online vacation planning experience. By transitioning from a third-party service in most markets, dynamic packaging engines can better tailor its package offerings, pricing and merchandising to consumer demand.

Currently, the travel industry has concentrated their efforts on developing open specifications messages, based on eXtensible Markup Language (XML), to ensure that messages can flow between industry segments as easily as within. For example, the OpenTravel Alliance (OTA) [5] is an organization pioneering the development and use of specifications that support e-business among all segments of the travel industry. The cumulative effort of various teams, individuals, associations, companies, and international organizations, including air, car, cruise, rail, hotel, travel agencies, tour operators and technology providers, has produced a fairly complete set of XML-based specifications for the travel industry (more than 140 XML specification files exist).

The current development of open specifications messages based on XML, such as the OTA schema, to ensure the interoperability between trading partners and working groups is not sufficiently expressive to guaranty an automatic exchange and processing of information. The development of a suitable ontology for the tourism industry is indispensable and will serve as a common language for travel-related terminology and a mechanism for promoting the seamless exchange of information across all travel industry segments.

The development of such an ontology can be used to bring together autonomous and heterogeneous Web services, Web processes, applications, data, and components residing in distributed environments. Semantics allow rich descriptions of Web services and Web processes that can be used by computers for automatic processing in various tourism related applications. The deployment of ontologies help articulate a well-defined set of common data elements or vocabulary that can support communication across multiple channels, expedite the flow of information, and meet travel industry and customer needs.

For the travel industry, the simplest form to construct an ontology is to retrieve rich semantic interrelationships from the data and terminology present in the XML-based OTA specifications already implemented [5] and available to organizations. This procedure is illustrated in Figure 2.


OTA
The OpenTravel™ Alliance
Schema Integration Using Ontologies
Travel Ontology
Travel Industry
Web Ontology Language (OWL)

	Schema - OTA 2004A
1	OTA_AirAvailRQ.xsd
2	OTA_AirAvailRS.xsd
3	OTA_AirBookRQ.xsd
4	OTA_AirBookRS.xsd
5	OTA_AirCommonTypes.xsd
6	OTA_AirDetailsRQ.xsd
7	OTA_AirDetailsRS.xsd
...	....
144	OTA_VehModifyRS.xsd
145	OTA_VehResRQ.xsd
146	OTA_VehResRS.xsd
147	OTA_VehRetResRQ.xsd
148	OTA_VehRetResRS.xsd

Fig. 2. Ontology for the travel industry

One possible language to construct such an ontology is using the Web Ontology Language (OWL) [6] designed by the World Wide Web Consortium (W3C). The OWL is designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content by providing additional vocabulary along with a formal semantics. It can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms.

OWL is appropriate to develop an ontology for the travel industry since it is intended to be used when the information used by Web services needs to be processed by applications, as opposed to situations where the content only needs to be presented to humans.

The development of such an ontology lead to the spearhead and foster the cross-industry consensus needed to establish and maintain the most effective and widely used specifications designed to electronically exchange business data and information among all sectors of the travel industry.

This effort represents what can be achieved by the symbiotic synthesis of two of the hottest R&D and technology application areas: Web services and the semantic Web, as recognized at the Thirteenth International World Wide Conference (2004) and in the industry press. The intelligent combination of Web services and the semantic Web can start off a technological revolution with the development of semantic Web processes [7]. These technological advances can ultimately lead to a new breed of Web-based applications for the travel industry.

2.2 Semantics for Web Services

In Web services domain, semantics can be classified into the following types [8] illustrated in Figure 1:

- Functional Semantics
- Data Semantics
- QoS Semantics and
- Execution Semantics

These different types of semantics can be used to represent the capabilities, requirements, effects and execution of a Web service. In this section we describe the nature of Web services and the need for different kind of semantics for them.

Functional Semantics. The power of Web services can be realized only when appropriate services are discovered based on the functional requirements. It has been assumed in several semantic Web service discovery algorithms [9] that the functionality of the services is characterized by their inputs and outputs. Hence these algorithms look for semantic matching between inputs and outputs of the services and the inputs and outputs of the requirements. This kind of semantic matching may not always retrieve an appropriate set of services that satisfy functional requirements. Though semantic matching of inputs and outputs are required, they are not sufficient for discovering relevant services. For example, two services can have the same input/output signature even if they perform entirely different functions. A simple mathematical service that performs addition of two numbers taking the numbers as input and produce the sum as output will have the same semantic signature as that of another service that performs subtraction of two numbers that are provided as input and gives out their difference value as output. Hence matching the semantics of the service signature may result in high recall and low precision. As a step towards representing the functionality of the service for better discovery and selection, the Web services can be annotated with functional semantics. It can be done by having an ontology called Functional Ontology in which each concept/class represents a well-defined functionality. The intended functionality of each service can be represented as annotations using this ontology.

Data Semantics. All the Web services take a set of inputs and produce a set of outputs. These are represented in the signature of the operations in a specification file. However the signature of an operation provides only the syntactic and structural details of the input/output data. These details (like data types, schema of a XML complex type) are used for service invocation. To effectively perform discovery of services, semantics of the input/output data has to be taken into account. Hence, if the data involved in Web service operation is annotated using an ontology, then the added data semantics can be used in matching the semantics of the input/output data of the Web service with the semantics of the input/output data of the requirements. Semantic discovery algorithm proposed in [9] uses the semantics of the operational data.

QoS Semantics: After discovering Web services whose semantics match the semantics of the requirements, the next step is to select the most suitable service. Each service can have different quality aspect and hence service selection involves locating the service that provides the best quality criteria match. Service selection is also an important activity in web service composition [10]. This demands management of QoS metrics for Web services. Web services in different domains can have different quality aspects. For organizations, being able to characterize Web processes based on QoS has several advantages: a) it allows organizations to translate their vision into their business processes more efficiently, since Web processes can be designed according to QoS metrics, b) it allows for the selection and execution of Web processes based on their QoS, to better fulfill customer expectations, c) it makes possible the monitoring of Web processes based on QoS, and d) it allows for the evaluation of alternative strategies when Web process adaptation becomes necessary.

Execution Semantics. Execution semantics of a Web service encompasses the ideas of message sequence, conversation pattern of Web service execution, flow of actions, preconditions and effects of Web service invocation, etc. Some of these details may not be meant for sharing and some may be, depending on the organization and the application that is exposed as a Web service. In any case, the execution semantics of these services are not the same for all services and hence before executing or invoking a service, the execution semantics or requirements of the service should be verified.

Some of the issues and solutions with regard to execution semantics are inherited from traditional workflow technologies. However, the globalization of Web services and processes result in additional issues. In e-commerce, using execution semantics can help in dynamically finding partners that will match not only the functional requirements, but also the operational requirements like long running interactions and complex conversations. Also, a proper model for execution semantics will help in coordinating activities in transactions that involve multiple parties.

3 Phases of the Web Process Lifecycle

As stated previously, the lifecycle of semantic Web processes includes the description/annotation, the advertisement, the discovery, and the selection of Web services, the composition of Web services that makeup Web processes, and the execution of Web processes. In this section, we discuss the characteristics of each of these stages.

3.1 Semantic Web Service Annotation

Today, Web service specifications are based on standards that only define syntactic characteristics. Unfortunately, it is insufficient, since the interoperation of Web services/processes cannot be successfully achieved. One of the most recognized solutions to solve interoperability problems is to enable applications to understand methods and data by adding meaning to them.

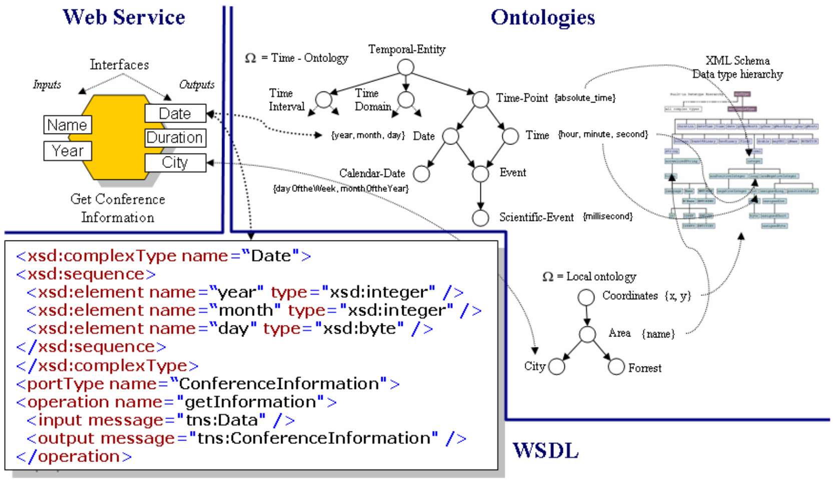
Many tools are available to create Web services. Primarily programs written in Java or any object oriented language can be made into Web services. In technical terms any program that can communicate with other remote entities using SOAP [11] can be called a Web service. Since the development of Web services is the first stage in the creation of Web services, it is very important to use semantics at this stage. During Web service development data, functional and QoS semantics of the service needs to be specified.

All the Web services (operations in WSDL file [12]) take a set of inputs and produce a set of outputs. These are represented in the signature of the operations in a WSDL file. However the signature of an operation provides only the syntactic and structural details of the input/output data.

To effectively perform operations such as the discovery of services, semantics of the input/output data has to be taken into account. Hence, if the data involved in Web service operation is annotated using an ontology, then the added data semantics can be used in matching the semantics of the input/output data of the Web service with the semantics of the input/output data of the requirements.

The Meteor-S Web Service Annotation Framework (MWSAF) [13] provides a framework and a tool to achieve automatic and semi-automatic annotation of web services using ontologies.

Figure 3 illustrates one solution to annotate WSDL interfaces with semantic metadata based on relevant ontologies [14]. A Web service invocation stipulate an input interface that specifies the number of input parameters that must be supplied for a proper Web service realization and an output interface that specifies the number of outputs parameters to hold and transfer the results of the Web service realization to other services.



```
<xsd:complexType name="Date">
<xsd:sequence>
  <xsd:element name="year" type="xsd:integer" />
  <xsd:element name="month" type="xsd:integer" />
  <xsd:element name="day" type="xsd:byte" />
</xsd:sequence>
</xsd:complexType>
<portType name="ConferenceInformation">
<operation name="getInformation">
  <input message="tns:Data" />
  <output message="tns:ConferenceInformation" />
</operation>
```

Fig. 3. Semantic annotation of a Web service specified with WSDL

3.2 Semantic Web Service Advertisement

After the service is developed and annotated, it has to be advertised to enable discovery. The UDDI registry is supposed to open doors for the success of service oriented computing leveraging the power of the Internet. Hence the discovery mechanism supported should scale to the magnitude of the Web by efficiently discovering relevant services among tens and thousands (or millions according to the industry expectations) of the Web services.

The present discovery supported by UDDI is inefficient as services retrieved may be inadequate due to low precision (many services you do not want) and low recall (missed the services you really need to consider). Effectively locating relevant services and efficiently performing the search operation in a scalable way is what is required to accelerate the adoption of Web services. To meet this challenge, Web service search engines and automated discovery algorithms need to be developed. The discovery mechanisms supported need to be based on Web services profiles with machine process-able semantics.

3.3 Semantic Web Service Discovery

This stage is the process of discovering appropriate services before selecting a specific Web service to and binding it to a Web processes [15]. The search of Web services to model e-commerce applications differs from the search of tasks to model traditional processes. One of the main differences is in terms of the number of Web services available to the composition process. In the Web, potentially thousands of Web services are available. One of the problems that need to be solved is how to efficiently discover Web services [10].

The discovery of Web services has specific requirements and challenges as compared to previous work on information retrieval systems and information integration systems. Several issues need to be considered:

- Precision of the discovery process. The search has to be based, not only on syntactic information, but also on data, functional, and QoS semantics.
- Enable the automatic determination of the degree of integration of the discovered Web services and a Web process host.
- The integration and interoperation of Web services differs from previous work on schema integration due to the polarity of the schema that must be integrated [10].

Typically, a cluster of Web services that match initial requirements is constructed. In the next phase (semantic Web service selection), we selected, from the cluster, the Web service that more closely matches our requirements. The cluster which contains the list of other services, which also match the requirements, is maintained. This is because a service may be chosen later in case of failure or breach of contract.

3.4 Semantic Web Service Selection

Web service selection is a need that is almost as important as service discovery. After discovering Web services whose semantics match the semantics of the requirement, the next step is to select the most suitable service. Each service can have different quality aspect and hence service selection involves locating the service that provides the best quality criteria match.

Service selection is also an important activity in Web service composition [10]. This demands management of QoS metrics for Web services. Web services in different domains can have different quality aspects. These are called *Domain Independent*

QoS metrics. There can be some QoS criteria that can be applied to services in all domains irrespective of their functionality or specialty. These are called *Domain Specific* QoS metrics. Both these kind of QoS metrics need shared semantics for interpreting them as intended by the service provider. This could be achieved by having an ontology (similar to an ontology used for data semantics) that defines the domain specific and domain independent QoS metrics.

3.5 Semantic Process Composition

The power of Web services can be realized only when they are efficiently composed into Web process. This requires a high degree of Interoperability among Web services. Interoperability is a key issue in e-commerce because more and more companies are creating business-to-customer and business-to-business links to better manage their value chain. In order for these links to be successful, heterogeneous systems from multiple companies need to interoperate seamlessly. Automating inter-organizational processes across supply chains presents significant challenges [16].

Compared to traditional process tasks, Web services are highly autonomous and heterogeneous. Sophisticated methods are indispensable to support the composition of Web process. Here again, one possible solution is to explore the use of semantics to enhance interoperability among Web services.

This stage involves creating a representation of Web processes. Many languages like BPEL4WS [17], BPML [18] and WSCI [19] have been suggested for this purpose. The languages provide constructs for representing complex patterns [20] of Web service compositions. While composing a process, four kinds of semantics have to be taken into account. The process designer should consider the functionality of the participating services (functional semantics), data that is passed between these services (data semantics), the quality of these services, the quality of the process as a whole (QoS semantics) and the execution pattern of these services, the pattern of the entire process (Execution semantics). Since Web process composition involves all kind of semantics, it may be understood that semantics play a critical role in the success of Web services and in process composition.

3.6 Execution of Web Processes

Web services and Web processes promise to ease several of nowadays infrastructure challenges, such as data, application, and process integration. With the emergence of Web services, workflow management systems (WfMSs) become essential to support, manage, enact, and orchestrate Web processes, both between enterprises and within the enterprise. Several researchers have identified workflows as the computing model that enables a standard method of building Web process applications and processes to connect and exchange information over the Web [21].

Execution semantics of a Web service encompasses the ideas of message sequence (e.g., request-response, request-response), conversation pattern of Web service execution (peer-to-peer pattern, global controller pattern), flow of actions (sequence, parallel, and loops), preconditions and effects of Web service invocation, etc.

Traditional formal mathematical models (Process Algebra [22]), concurrency formalisms (Petri Nets [23], state machines [24]) and simulation [25] techniques) can be used to represent execution semantics of Web services. Formal modeling for workflow scheduling and execution are also relevant [26]. With the help of execution semantics process need not be statically bound to component Web services. Instead, based on the functional and data semantics a list of Web services can be short listed, QoS semantics can be used to select the most appropriate service, and execution semantics the service can be bound to a process and used to monitor process execution.

3.7 Semantic Web Process QoS

New trading models, such as e-commerce, require the specification of QoS metrics such as products or services to be delivered, deadlines, quality of products, and cost of service. To enable adequate QoS management, research is required to develop mechanisms that semantically specify, compute, monitor, and control the QoS of the products or services to be delivered [10, 27].

In e-commerce and e-business Web processes, suppliers and customers define a binding agreement between the two parties, specifying QoS items such as services to be delivered, deadlines, and cost of services. The management of QoS metrics of semantic Web processes directly impacts the success of organizations participating in e-commerce. Therefore, when services or products are created or managed using Web processes, the underlying WfMS must accept the specifications and be able to estimate, monitor, and control the QoS rendered to customers.

A comprehensive QoS model that allows the description of Web processes components from a QoS perspective have already been developed [28]. One of the models includes three dimensions: time, cost, and reliability. The QoS model is coupled with an algorithm (the SWR algorithm [28]) to automatically compute the overall QoS of Web processes. These developments can be easily applied to automatically compute the duration, cost, and reliability of Web processes.

4 Ongoing Work

The industrial research related to semantic Web services depends on the ongoing development of open standards that ensure interoperability between different implementations. Several initiatives have been conducted with the intention to provide platforms and languages that will allow easy integration of heterogeneous systems. The standardization efforts for the technologies that underlie Web services include Simple Object Access Protocol (SOAP)[29], Web Services Description Language (WSDL)[12], Universal Description, Discovery and Integration (UDDI)[30], and process description languages, such as Business Process Execution Language for Web Services (BPEL4WS)[17] (from Microsoft, IBM, BEA).

Recently, the Semantic Web Services Initiative (SWSI)[31], an initiative of academic and industrial researchers has been composed to create infrastructure that combines Semantic Web and Web Services to enable the automation in all aspects of Web services. In addition to providing further evolution of OWL-S [32], SWSI will also be a forum for working towards convergence of OWL-S with the products of the WSMO[33]/WSML[34]/WSMX[35] research effort.

WSMO is a complete ontology for the definition of Semantic Web Services. It follows the WSMF as a vision of Semantic Web Services. WSML is a family of languages that allow Semantic Web Service designers to define Semantic Web Services in a formal language. The WSMX provides a standard architecture for the execution of Semantic Web Services.

Besides these major standards and initiatives, there are two ongoing projects being developed in the US, the LSDIS METEOR-S project [36], and in Europe, the DERI SWWS project [37].

The METEOR-S (METEOR for Semantic Web services) project is focused on the usage of semantics for the complete lifecycle of semantic Web processes, namely, annotation, discovery, composition, and execution.

DERI [38] is currently working on a project titled Semantic Web enabled Web Services (SWWS). DERI researchers recognize that to use the full potential of Web services and the technology around UDDI, WSDL and SOAP, it is indispensable to use semantics, since current technologies provide limited support for automating Web service discovery, composition and execution. Important objectives of the SWWS initiative include providing a richer framework for Web Service description and discovery, as well as, providing scalable Web Service mediation middleware.

5 Conclusions

Systems and infrastructures are currently being developed to support Web services. The main idea is to encapsulate an organization's functionality within an appropriate interface and advertise it as Web services.

While in some cases Web services may be utilized in an isolated form, it is normal to expect Web services to be integrated as part of Web processes. There is a growing consensus that Web services alone will not be sufficient to develop valuable Web processes due the degree of heterogeneity, autonomy, and distribution of the Web.

For example, a new requirement for the travel industry is the ability to dynamically compose travel packages from the aggregation and orchestration of distributed Web services. Current approaches, using XML-based specification messages, are not sufficient to enable the creation of dynamic travel packages. One solution is the use of ontologies to overcome semantic problems that arise from the autonomy, heterogeneity, and distribution of Web services.

Several researchers agree that it is essential for Web services to be machine understandable in order to support all the phases of the lifecycle of Web processes. In this paper we have presented a set of challenges that the emergence of semantic Web processes has brought to organizations. Designing semantic Web processes entails research in two areas: Web services and the Semantic Web. We have presented how applying semantics to each of the steps in the Semantic Web Process lifecycle can help address critical issues in reuse, integration and scalability.

References

1. Sheth, A.P., W.v.d. Aalst, and I.B. Arpinar, Processes Driving the Networked Economy. IEEE Concurrency, 1999. 7(3): p. 18-31.
2. W3C, W3C Semantic Web Activity. http://www.w3.org/2001/sw/. 2004.
3. Uschold, M. and M. Gruninger, Ontologies: Principles, methods and applications. Knowledge Engineering Review, 1996. 11(2): p. 93-155.

4. Gruber, T.R., Toward principles for the design of ontologies used for knowledge sharing. International Journal of Human-Computer Studies, 1995. 43(5-6): p. 907-928.
5. OTA, OpenTravel Alliance. www.opentravel.org, 2004.
6. OWL, Web Ontology Language (OWL). http://www.w3.org/2004/OWL/, 2004, World Wide Web Consortium (W3C).
7. Cardoso, J. and A.P. Sheth. Semantic Web Processes: Semantics Enabled Annotation, Discovery, Composition and Orchestration of Web Scale Processes. in Fourth International Conference on Web Information Systems Engineering (WISE'03). 2003. Roma, Italy.
8. Sivashanmugam, K., et al., Metadata and Semantics for Web Services and Processes, in Datenbanken und Informationssysteme (Databases and Information Systems). Festschrift zum 60. Geburtstag von Gunter Schlageter, W. Benn, et al., Editors. 2003: Hagen, Germany. p. 245-271.
9. Paolucci, M., et al. Importing the Semantic Web in UDDI. in Proceedings Web Services, E-Business and Semantic Web Workshop, CAiSE 2002. 2002. Toronto, Canada.
10. Cardoso, J. and A. Sheth, Semantic e-Workflow Composition. Journal of Intelligent Information Systems (JIIS). 2003. 21(3): p. 191-225.
11. Graham, S., et al., Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI. 2002: SAMS.
12. Christensen, E., et al., W3C Web Services Description Language (WSDL). http://www.w3c.org/TR/wsdl, 2001.
13. Patil, A., et al. MWSAF - METEOR-S Web Service Annotation Framework. in 13th Conference on World Wide Web. 2004. New York City, USA.
14. Cardoso, J., F. Curbera, and A. Sheth. Tutorial: Service Oriented Archiectures and Semantic Web Processes. in The Thirteenth International World Wide Web Conference (WWW2004). 2004. New York, USA.
15. Verma, K., et al., METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services. Journal of Information Technology and Management (in print), 2004.
16. Stohr, E.A. and J.L. Zhao, Workflow Automation: Overview and Research Issues. Information Systems Frontiers, 2001. 3(3): p. 281-196.
17. BPEL4WS, Web Services. http://www-106.ibm.com/developerworks/webservices/, 2002, IBM.
18. BPML, Business Process Modeling Language. http://www.bpmi.org/, 2004.
19. WSCI, Web Service Choreography Interface (WSCI) 1.0. http://www.w3.org/TR/wsci/, 2002, World Wide Web Consortium (W3C).
20. Aalst, W.M.P.v.d., et al. Advanced Workflow Patterns. in Seventh IFCIS International Conference on Cooperative Information Systems. 2000.
21. Fensel, D. and C. Bussler, The Web Service Modeling Framework. http://www.cs.vu.nl/~dieter/ftp/paper/wsmf.pdf, 2002, Vrije Universiteit Amsterdam (VU) and Oracle Corporation.
22. Bergstra, J.A., A. Ponse, and S.A. Smolka, Handbook of Process Algebra. 2001: Elsevier.
23. Aalst, W.M.P.v.d., The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers, 1998. 8(1): p. 21-66.
24. Hopcroft, J.E., R. Motwani, and J.D. Ullman, Introduction to Automata Theory, Languages, and Computation. 2000, Mass.: Addison-Wesley Publishing Company.
25. Bosilj, V., M. Stemberger, and J. Jaklic, Simulation Modelling Toward E-Business Models Development. International Journal of Simulation Systems, Science & Technology, Special Issue on: Business Process Modelling, 2001. 2(2): p. 16-29.

26. Attie, P., et al. Specifying and Enforcing Intertask Dependencies. in Proceedings 19th Intlernational Conference on Very Large Data Bases. 1993. Dublin, Ireland: Morgan Kaufman.
27. Cardoso, J., A. Sheth, and J. Miller. Workflow Quality of Service. in International Conference on Enterprise Integration and Modeling Technology and International Enterprise Modeling Conference (ICEIMT/IEMC'02). 2002. Valencia, Spain: Kluwer Publishers.
28. Cardoso, J., et al., Quality of service for workflows and web service processes. Web Semantics: Science, Services and Agents on the World Wide Web Journal, 2004. 1(3): p. 281-308.
29. SOAP, Simple Object Access Protocol 1.1. http://www.w3.org/TR/SOAP/, 2002.
30. UDDI, Universal Description, Discovery, and Integration. http://www.uddi.org/, 2002.
31. SWSI, Semantic Web Services Initiative (SWSI). http://www.swsi.org/, 2004.
32. OWL-S, OWL-based Web Service Ontology. http://www.daml.org/services/owl-s/, 2004.
33. WSMO, Web Services Modeling Ontology (WSMO). http://www.wsmo.org/, 2004.
34. WSML, Web Service Modeling Language (WSML). http://www.wsmo.org/wsml/index.html, 2004.
35. WSMX, Web Services Execution Environment (WSMX). http://www.wsmx.org/, 2004.
36. LSDIS, METEOR-S: Semantic Web Services and Processes. http://lsdis.cs.uga.edu/Projects/METEOR-S/index.php, 2004.
37. SWWS, Semantic Web Enabled Web Service. http://swws.semanticweb.org/, 2004, Digital Enterprise Research Institute (DERI).
38. DERI, Digital Enterprise Research Institute (DERI). http://www.deri.ie/, 2004.

Academic and Industrial Research: Do Their Approaches Differ in Adding Semantics to Web Services?

Jorge Cardoso[1], John Miller[2], Jianwen Su[3], and Jeff Pollock[4]

[1] Department of Mathematics and Engineering, University of Madeira, 9100-390, Funchal, Portugal
jcardoso@uma.pt
[2] Department of Computer Science, University of Georgia, 415 Graduate Studies Research Center, Athens GA 30602-7404, USA
jam@cs.uga.edu
[3] Department of Computer Science, University of California, Santa Barbara, CA 93106-5110, USA
su@cs.ucsb.edu
[4] Network Inference, 5900 La Place Ct., Suite 250 Carlsbad, CA 92008, USA
jeff.pollock@networkinference.com

Abstract. Since the new terms, “Semantic Web” and “Web services”, have been introduced, researchers have followed two different roads. Following one road, academia has focused on developing a new set of languages to enable the automation of Web services execution and integration based on the Semantic Web. On the other road, industry has taken the lead to propose and develop technologies and infrastructures to support Web services and Web processes without, until recently, paying much attention to semantics. It is fundamental to analyze the trend that is being followed with regard to the “Semantic Web” and “Web services”. Therefore, two important questions need to be answered: “do the approaches taken by academia and industry differ in how they add semantics to Web services?” and “are their efforts converging or diverging?” This paper, based on a panel discussion at an international conference on Web services, which consisted of members of both academia and industry, addresses precisely these two questions.

1 Introduction

In July of 2004, a panel was convened to consider a convergence or divergence between academic and industrial approaches to adding semantics to Web service and/or Web process descriptions. Everyone agrees that more semantics (or meaning) should be added to Web service descriptions. Differences results when the communities address the questions of how and how much. How much semantics? Should a Web service operation be given a full semantic specification, say using operational semantics [15] or would a functional classification or categorization suffice? How machine processable or understandable should the semantics be on the formality vs. informality scale? For example, a complete and formal semantic specification is difficult for humans to create or understand. A simpler agreement based approach predicated on standard interfaces (e.g., port types) may be a better short-term solution. It is also possible to move the standard back to an ontological level and then require the parts of a port type to map to an ontology. This approach to interoperability has proved successful in database integration (where each schema is mapped to a common ontology). One could expect similar success for Web services, yet the problem is more complicated since the description of operations is more complicated than description of data objects.

Given the importance and complexity of the issue (adding semantics), it makes sense that the academic and industrial approaches do differ. The industrial approach should be near-term, practical and with a high probability of success, while academia can afford to be long-term, ambitious and speculative. However, too much divergence may cause a fracture in which industry settles for too little and academia will design great things that will never be used.

In this paper, we briefly survey the current research and development occurring in academia and industry on Semantic Web Services (SWS). The panel consisted of researchers from both sectors and the paper strives for a balanced treatment highlighting the strengths of both approaches, analyzing their differences and seeking common ground for future work.

The paper is organized as follows: Section 2 reviews the brief history of attempting to provide semantics for Web services and relates this to the long history of attempting this for programs. Issues and directions are discussed as well as some aspects of current active research projects are highlighted. Section 3 parallel section 2, but from an industrial perspective. Because of the complexity of semantics, there are likely to be diminishing returns if too much is added (e.g., problems with intractability and undecidability as well as too hard to use). This section will start with the currently used standards for describing (WSDL 1.1) [22], publishing (UDDI 2.0) [19] and orchestrating (BPEL 1.1) [1] Web services and will consider how semantics are and will impact new (e.g., WSDL 2.0 [23] and UDDI 3.0 [20]) as well as future standards. Section 4 attempts to resolve the differing approaches into a recipe for long-term cooperation and success of this most vital new technological area. Finally, section 5 gives a brief summary of the most important aspects discussed in this paper.

2 Academic Research on SWS

Academic research into Semantic Web Services began with the work of DAML-S group [4]. The idea was to use a formal language to precisely define what a Web service does. A basic description along these line is provided by the Web Service Description Language (WSDL). WSDL descriptions are rather shallow and focus on operational aspects. As a consequence, these descriptions are inadequate for automated discovery or composition of Web services. Much richer and deeper machine-processable descriptions are therefore required. The DAML-S (now the OWL-S [13]) group added profile, process and grounding descriptions. A profile describes what the Web service does functionally in terms of input (I), output (O), precondition (P), and result (R), the process describes how it is built out of components and the grounding maps these to WSDL files. Much of the semantics is captured in the IOPR specifications.

A Web service, as a software component, has one or more operations that can be invoked as well as its own state. An operation may be described by indicating the types of its inputs and outputs, any preconditions required of the input as well as the results of the operation (either on the state or the outputs produced). Actually, this goal of specifying what an operation does or, in general, what a process does has a long tradition in Computer Science and includes work in the fields of program methodology, formal programming language semantics, software engineering and software agents. The problems are complex, but the potential payoff is great.

Besides the major OWL-S project, there are two ongoing projects being developed in the US, the LSDIS METEOR-S project, and in Europe, the DERI SWWS project.

The METEOR-S [14] (METEOR for Semantic Web services) project is focused on the usage of semantics for the complete lifecycle of semantic Web processes, which represent complex interactions between semantic Web services. The METEOR-S project targets research on four important areas of the lifecycle of semantic Web processes, namely, annotation, discovery, composition, and execution. For each of the research stages in the lifecycle a framework, infrastructure or environment has been developed and implemented. The METEOR-S semantic Web Service Annotation Framework (MWSAF) semi-automatically marks up Web service descriptions with ontologies. The algorithms developed match and annotate WSDL files with relevant ontologies. The METEOR-S Web Service Discovery Infrastructure (MWSDI) uses an ontology-based approach to organize registries, enabling semantic classification of all Web services based on domains. Each of these registries supports semantic publication of the Web services, which is used during the discovery process. The METEOR-S Web Service Composition Framework (MWSCF) enhances current Web process composition techniques by using Semantic Templates to capture the semantic requirements of the process [3]. The METEOR-S Web Service Dynamic Process Manager (MWSDPM) allows deployment-time and run-time binding of Web services to an abstract process, based on business and process constraints.

DERI [5] is currently working on a project titled Semantic Web enabled Web Services (SWWS). DERI researchers recognize that to use the full potential of Web services and the technology around UDDI, WSDL and SOAP, it is indispensable to use semantics, since current technologies provide limited support for automating Web service discovery, composition and execution. Important objectives of the SWWS initiative include providing a richer framework for Web service description and discovery, as well as, providing scalable Web service mediation middleware. Any necessary mediation would be applied based on semantic data and process ontologies and semantic interoperation.

Aside from investigations on functional descriptions of Web services, there are also work on behavioral descriptions (see [11]). The behavior signature [11] of a service describes how the service can interact with other services. Providing behavior signatures is critical in service composition. For example, the two interacting services may both wait for messages from each other and none of them can thus proceed [6, 7]. It has been argued that Web service composition, automated or semi-autmated, critically relies on the interaction patterns in the behavior specification [9, 10, 21, 2]. A tool WSAT was recently developed for analyzing conversations and Web service bahaviors [7].

3 Industrial Research and Development on SWS

The industrial research related to semantic Web services depends on the ongoing development of open standards that ensure interoperability between different implementations. Several initiatives have been conducted with the intention to provide platforms and languages that will allow easy integration of heterogeneous systems. The standardization efforts for the technologies that underlie Web services include Simple Object Access Protocol (SOAP) [17], Web Services Description Language (WSDL), Universal Description, Discovery and Integration (UDDI), and process description languages. Several process description languages have been proposed and studied by the industry.

These languages include W3C WS Choreography Group, Business Process Execution Language for Web Services (BPEL4WS, or simply BPEL) (from Microsoft, IBM, BEA), WSCL (from HP), BPML (from Microsoft), WSCI (from SUN, BEA, Yahoo, and other), XLANG (from Microsoft), and WSFL (from IBM).

The WSDL is already well established as an essential building block in the evolving stack of Web service technologies, and is being developed and standardized in the W3C's Web Services Description Working Group. WSDL is a specification to describe networked XML-based services. It provides a simple way for service providers to describe the basic format of requests to their systems regardless of the underlying protocol. WSDL is a key part of the effort of the UDDI initiative to provide directories and descriptions of such on-line services for electronic commerce and electronic business. WSDL does not, however, support the specification of processes composed of basic Web services nor it envision the use of semantics.

In this area, the BPEL4WS, currently has the most prominent status and enables defining business processes as coordinated sets of Web service interactions. The W3C's Web Services Choreography Working Group also has been chartered to explore this technical area.

All in all, there are few commercial products available that have successfully implemented a semantic layer alongside robust a Web services infrastructure, this despite significant industrial support which exists for standards such as

WSDL, BPEL, and UDDI. As has been mentioned, there are two primary considerations for semantics with Web services - the process layer and the data layer. Most enterprise vendors have indeed recognized the importance and value of semantic metadata for each area, but tend to implement solutions in proprietary and brittle ways; using their own metadata formats for internal semantic reconciliation.

With regard to the process and orchestration semantics, many vendors seem to be taking a “wait-and-see” approach while the emerging standards converge. OWL-S, SWWS/WSML, and BPEL each have important strengths to add to an overarching semantic Web services capability. Leadership from DERI and the W3C have each expressed a strong interest in converging the best of each specification - vendors will no doubt wait for this alignment prior to implementing either on their own.

The hesitation shared by most commercial vendors will not be shared by many industrial research groups - IBM, HP, France Telecom, and Fujitsu have all applied semantics to Web services for innovative, discovery-driven use cases.

In contrast to “negotiation-style” semantic Web services, there are others who take a “query-driven” approach. In fact, some commercial vendors have begun implementing semantic layers on top of Web services as a way to issue queries to them instead of writing more brittle contracts. Annotating Web services using the W3C Web Ontology Language (OWL) can make it simpler evolve services in dynamic businesses. To do this, modeling tools map ontologies to Web service WSDL interfaces and a runtime inference engine issues query plans to the underlying services. This style of semantic query is clearly distinct from process-centric approaches, but both approaches help automate meaningful access to overly abundant corporate information.

4 Common Ground for Future Work

For Web services to become a platform for semantic service oriented computing, academic and industrial researchers will need to create terminologies, technologies, and products that enable sophisticated solution for the advertisement, discovery, selection, composition, and execution of Web services.

Recently, the Semantic Web Services Initiative (SWSI), an initiative of academic and industrial researchers has been composed to create infrastructure that combines Semantic Web and Web services to enable the automation in all aspects of Web services. In addition to providing further evolution of OWL-S, SWSI will also be a forum for working towards convergence of OWL-S with the products of the SWWS/WSMO [25]/WSML [24]/WSMX [26] research effort, which supplies Web service providers with a core set of constructs for describing the properties of their Web services in computer-interpretable form. OWL-S will facilitate the automation of Web service tasks, including automated Web service discovery, composition, and execution. The current version of OWL-S builds on the Ontology Web Language (OWL) recommendation produced by the Web-Ontology Working Group at the World Wide Web Consortium. OWL-S is the first well-researched Web Services Ontology, and has numerous users from the academia.

WSMO is a complete ontology for the definition of Semantic Web Services. It follows the WSMF as a vision of Semantic Web Services. WSMO itself is defined using an ontology language based on F-Logic [12]. It contains all concepts required for Semantic Web Services: Ontology, Mediator, Goal, Web Service Interface. The WSML is a family of languages that allow Semantic Web service designers to define Semantic Web services in a formal language. The WSMX provides a standard architecture for the execution of Semantic Web services. Its architecture is component-based and one possible implementation of Service-oriented Architectures. WSMX itself has execution semantics.

The largest patch of common research ground that industry and academia have to share is simple, or rather, making semantic Web services simpler. As with all semantic technologies, the rigor of expressing semantic Web services metadata (OWL, OWL-S, F-Logic, XML, etc.) with required precision is daunting without good tools. One day analysts will be dragging-and-dropping process diagrams and point-and-clicking ontology mappings. Until then, researchers in industry and academia would be well served to examine modeling heuristics to lower barriers for widespread adoption.

The more likely path of common ground will likely be to reach agreement on ontologies for service descriptions, processes, and security. At an even more fundamental level, researchers will have to measure the strengths and limitations of different representations such as description logics, horn-logic, and F-Logic for the erent layers of the semantic Web services architecture. In significant ways, the infusion of semantics will alter today's conceptions of the service-oriented architecture paradigm.

5 Summary

Many believe that a new Web will emerge in the next few years, based on the large-scale research and development ongoing on the Semantic Web and Web services. The intersection of these two, Semantic Web services, may prove to be even more significant. Academia has mainly approached this area from the Semantic Web side, while industry is beginning to consider its importance from the Web services side. Academia started developing semantic-based Web services languages, such as DAML-S (now OWL-S), to enrich the description of Web services to facilitate greater automation. The idea was to make explicit the representation of the semantics underlying data, services, and other resources, providing a qualitatively new level of service. Industry was interested in developing an infrastructure that could allow software applications to be accessed and executed via the Web based on the idea of Web services. Their efforts resulted in important, practical, and functional standards such as UDDI, WSDL, SOAP, XLANG, WSFL, WSCI, BPML, BPEL4WS, etc. While the two approaches can be seen as being parallel, recently their is some area of convergence. Both academia and industry have realized that for the sake of automation and dynamism in all aspects

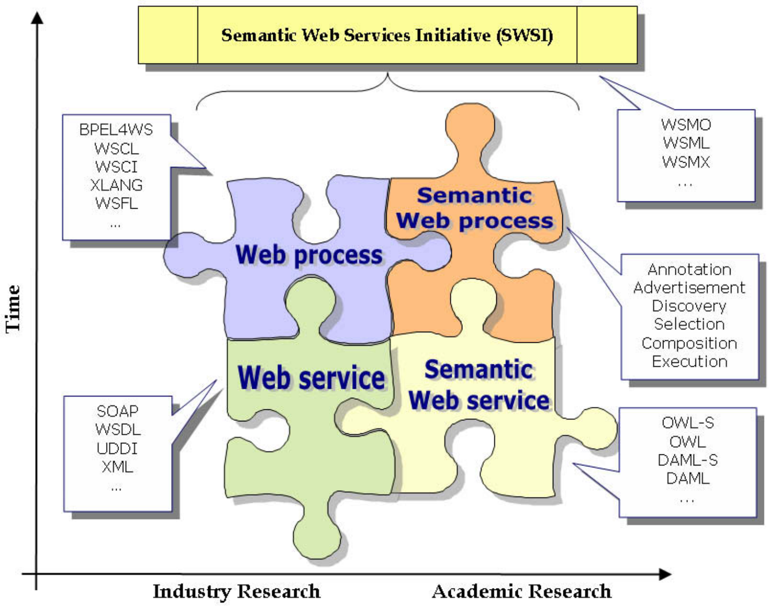


Fig. 1. Industry and academic research

of Web services provision, it was indispensable to create an infrastructure that combines, at least to some extent, Semantic Web and Web services technologies (Fig. 1). This paper has highlighted some of the contributions of both industry and academia and discussed recent cooperative efforts such as SWSI. Semantic Web Service technology's potential impact makes it essential for further and expanding cooperative efforts to be pursued in the future.

References

1. Business Process Execution Language for Web Services Version 1.1, 05 May 2003, http://www-128.ibm.com/developerworks/library/ws-bpel/.
2. T. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification: A new approach to design and analysis of e-service composition. In *Proc. 12th Int. World Wide Web Conference (WWW)*, May 2003.
3. Cardoso, J. and A. Sheth, Semantic e-Workflow Composition. Journal of Intelligent Information Systems (JIIS), Vol. 12, No. 3 (November 2003) pp. 191-225.
4. DAML Services Coalition (alphabetically A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, H. Zeng), "DAML-S: Semantic Markup for Web Services," in Proceedings of the International Semantic Web Working Symposium (SWWS), July 30-August 1, 2001.
5. Digital Enterprise Research Institute, http://www.deri.ie/.
6. H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based verification of Web service compositions. In *Proc. 18th IEEE Int. Conf. on Automated Software Engineering Conference*, 2003.

7. X. Fu, T. Bultan, and J. Su. WSAT: A tool for formal analysis of Web service compositions. In *Proc. of 16th Int. Conf. on Computer Aided Verification (CAV)*, 2004.
8. Gruber, T.R., Toward principles for the design of ontologies used for knowledge sharing. International Journal of Human-Computer Studies, 1995. 43(5-6): pp. 907-928.
9. J. E. Hanson, P. Nandi, and S. Kumaran. Conversation support for business process integration. In *Proc. 6th IEEE Int. Enterprise Distributed Object Computing Conference (EDOC)*, 2002.
10. J. E. Hanson, P. Nandi, and D. W. Levine. Conversation-enabled Web services for agents and e-business. In *Proc. Int. Conf. on Internet Computing (IC-02)*, CSREA Press, 2002.
11. R. Hull, M. Benedikt, V. Christophides, and J. Su. E-services: A look behind the curtain. In *Proc. ACM Symp. on Principles of Database Systems*, 2003.
12. M. Kifer, G. Lausen, and James Wu: Logical foundations of object oriented and frame-based languages. Journal of the ACM, 42(4):741-843, 1995.
13. OWL-S: David Martin, Massimo Paolucci, Sheila McIlraith, Mark Burstein, Drew McDermott, Deborah McGuinness, Bijan Parsia, Terry Payne, Marta Sabou, Monika Solanki, Naveen Srinivasan, Katia Sycara, "Bringing Semantics to Web Services: The OWL-S Approach," Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), July 6-9, 2004, San Diego, California, USA.
14. METEOR-S: Semantic Web Services and Processes, LSDIS Lab, University of Georgia, http://lsdis.cs.uga.edu/ and http://swp.semanticweb.org/.
15. G. D. Plotkin, "A Structural Approach to Operational Semantics," University of Aarhus, Denmark (1981)
16. OWL Web Ontology Language Overview, W3C Candidate Recommendation, 18 August 2003, http://www.w3.org/TR/owl-features/
17. Simple Object Access Protocol (SOAP) 1.1, http://www.w3.org/TR/SOAP/
18. Semantic Web Services Initiative (SWSI), http://www.swsi.org/
19. UDDI Spec Technical Committee Specification, 19 July 2002, http://www.uddi.org/specification.html
20. UDDI Spec Technical Committee Specification, 14 October 2003, http://uddi.org/pubs/uddi_v3.htm
21. Web Services Conversation Language (WSCL) 1.0, W3C Note, 14 March 2002, http://www.wscl.org/
22. Web Services Description Language (WSDL) 1.1, W3C Note, 15 March 2001, http://www.w3.org/TR/wsdl
23. Web Services Description Language (WSDL) 2.0, W3C Working Draft, 3 August 2004, http://www.w3.org/TR/2004/WD-wsdl20-20040803/
24. Web Service Modeling Language (WSML), http://www.wsmo.org/wsml/index.html
25. Web Services Modeling Ontology (WSMO), http://www.wsmo.org/
26. Web Services Execution Environment (WSMX), http://www.wsmx.org/

Interoperability in Semantic Web Services

Boualem Benatallah and H.R. Motahari Nezhad

School of Computer Science and Engineering,
The University of New South Wales,
Sydney, NSW 2052, Australia
{boualem, hamidm}@cse.unsw.edu.au

Abstract. Semantic Web services approach is emerging as a promising technology for the effective automation of services development and interoperability by providing richer descriptions of service properties, capabilities and behavior in form of metadata. In this short paper, we discuss interoperability issues in semantic Web services.

1 Introduction

The vision of Web services is to allow autonomous partners to advertise their terms and capabilities, and engage in peer-to-peer interactions with any other partner and enable on demand computing through composition and outsourcing [4]. While Web services technologies have clearly influenced positively the potential of the Web infrastructure by providing programmatic access to information and services, they are hindered by lack of rich and machine understandible abstractions to describe service properties, capabilities, and behavior.

Semantic Web aims at improving the technology to organise, search, integrate, and evolve Web-accessible resources by using rich and machine understandable abstractions for the representation of resources semantics. Ontologies are proposed as means to address semantic heterogeneity among Web-accessible information sources and services. Efforts in this area include the development of ontology languages such as RDF and OWL. In the context of Web services, ontologies promise to take interoperability a step further by providing rich description and modelling of services properties, capabilities, and behaviour [3]. OWL-S (formerly called DAML-S) [2] is an ontology based on OWL ontology language for describing Web services by adding metadata that support service descriptions to be understood by machines.

OWL-S consists of three interrelated subontologies, known as *ServiceProfile*, *ServiceModel*, and *ServiceGrounding*. The profile describes the capabilities and parameters of the service including both functional and non-functional properties. The service model details both the control structure and dataflow structure of the service required to execute a service. The service grounding specifies the details of how to access the service via messages (e.g., communication protocol, message formats, addressing, etc).

In this paper, we discuss interoperability issues in the context of semantic Web services at three layers: *content*, *conversation*, and *policy*.

2 The Content Layer

This layer provides protocols, languages and mediators for interoperable and consistent interpretation of the content of interfaces of services and exchanged messages by hiding encoding, structure and semantic heterogeneities. Encoding differences arise when two services provide the same functionality using different operation signatures, i.e., different operation names and input/output schemas. Structure heterogeneity happens due to presence of structure differences between the interfaces of two or more partner services, e.g., missing/extra operations or input/output messages in operations of one of the services. Semantic heterogeneity means that services provide overlapping but not the same functionality or when they have different interpretations of the same concept in exchanged business documents. For instance, the data item "Price" in an invoice document may mean inclusion or exclusion of tax.

At the content layer the contribution of semantic Web services is considerable, since adding meta-data to the description of Web services and support of ontology provide services with a rich description of interfaces and messages. OWL-S uses the profile ontology for this purpose. Rich descriptions would hide the differences in the service interfaces and business documents mainly of type of encoding and semantic heterogeneities. In particular, when two or more services interoperate their ontologies can provide the basis for understanding the terms and interfaces of participant services, the differences among them and reasoning about how to resolve the differences. However, scalable integration and management of ontology-based descryptions is still an open issue.

3 The Conversation Layer

This layer deals with the semantic of interactions between partners. The semantic of interactions must be well defined such that there is no ambiguity as to what a message may mean, what actions are allowed, what responses are expected and in what order messages should be sent. For instance, if the protocol of a client requires explicit acknowledgement when sending a purchase order message, the protocol of the provider should support that. Interoperability at this layer is a challenging task since it requires understanding the semantics of external business protocols of partner services. Automation requires rich description models but a balance between expression power and simplicity is important for the success of the technology.

At the conversation layer, OWL-S uses the service model ontology. The focus of the service model ontology is to define and model atomic, abstract and composite processes. Indeed, it considers a process as an implementation of a composite service and does not provide high level frameworks, notations and methodologies for modeling choreographies of services and supporting automated interoperability (e.g., protocol compatibility and conformance). In addition, it does not cater for important abstraction such as temporal constraints (e.g., when an operation should occur) and operation invocations implications and effects from the requested perspective (e.g., whether requesters can cancel an operation and what is the cancellation fee). Nevertheless, the ontologies provide the basis for development of richer conversation models, which enables more effective static and dynamic binding, as client can be more selective on the behavioral properties of the services they bind to.

4 The Policy Layer

This layer is concerned with the matching and compliance checking of service policies (e.g, QoS, privacy policies). Policies play a vital role in business to business integration using Web services by making the implicit information, as in closed environments, explicit, which is essential in autonomous environments. Policy matching means checking whether capabilities and requirements of partner services are compatible and tries to find a composition of policy assertions, which allow autonomous services to interoperate.

At the policy layer, OWL-S does not explicitly formalize and specify policies. However, the profile of OWL-S can be used to express policies such as security and privacy as a part of unbounded list of service parameters of the profile. In addition to the fact that OWL-S does not provide for the fragmentation of different policies, the lacks of high level modeling and reasoning about policies hinders the specification of relevant properties in a way that is useful for activities such as formal analysis, consistency checking of system functionalities, refinement, and code generation. At the same times, there are several emerging research efforts to use ontologies as the basis for defining vocabularies to represent policies (e.g, [5] uses an ontology-based approach for representing security policies).

5 Conclusion

By leveraging efforts in both Web services and semantic Web, semantic Web services paradigm promises to take Web technologies a step further by providing foundations to enable automated discovery, access, combination, and management of Web services. Efforts in this area focus on providing rich and machine understandable representation of services properties, capabilities, and behavior as well as reasoning mechanisms to support automation activities.

Advances and standardization efforts in area of semantic Web services provide the basis to address the interoperability issues in the content, conversation and policy layers. However, much more efforts are required in semantic Web services area to support richer metadata descriptions of service interactions, choreography and automated convesation protocols interoperability. In addition, the support for specifying, rich description, and fragmentation of policies is important for automated development (e.g., code generation and compliance checking) and service clients to know how to interact with services.

To coclude, effective abstracting and supporting service protocols and policies with rich descriptions can form the basis of the building blocks of a scalable and agile service oriented infrastructure. Richer conversation models enable a more effective static and dynamic binding as clients for instance may require that the selected service allow the cancellation of a given operation within a certain time interval from its completion but currently there is no consideration for such descriptions in conversation models. Other automation that will benefit from richer descriptions are compatibility checking of protocols, validation of service composition models, generation of service composition skeletons, and joint analysis of compositions and protocol specifications [1].

Acknowledgement. Authors would like to thank Fabio Casati, Farouk Toumani and Halvard Skogsrud for their valuable contributions to this work.

References

1. Benatallah, B., Casati, F., Skogsrud, H., and Toumani, F.: Abstracting and Enforcing Web Service Protocols. Int'l Journal of Cooperative Information Systems (IJCIS). World Scientific, December (2004) (To appear).
2. OWL-S: Semantic Markup for Web Services. www.daml.org/services/owl-s/.
3. Fensel, D., and Musen, M.A. (ed.): Special Issue on the Semantic Web. IEEE Intelligent Systems, vol. 16 no. 2, March/April (2001), 24-79.
4. Chung, J.Y., Lin, K.J., Mathieu, R.G. (ed.): Special Section on Web Services Computing. IEEE Computer, vol. 36, no. 10, (2003) 35-71.
5. Kagal, L., Paolucci, M., Srinivasan, N., Denker, G., Finin, T., Sycara, K.: Authorization and Privacy in Semantic Web Services. IEEE Intelligent Systems, vol. 19, no. 4, July/August (2004) 50-56.

Bringing Semantics to Web Services: The OWL-S Approach

David Martin[1], Massimo Paolucci[2], Sheila McIlraith[3], Mark Burstein, Drew McDermott, Deborah McGuinness, Bijan Parsia, Terry Payne, Marta Sabou, Monika Solanki, Naveen Srinivasan, and Katia Sycara

[1] Artificial Intelligence Center, SRI International, Menlo Park, CA, USA
martin@ai.sri.com
[2] Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA
paolucci+@ri.cmu.edu
[3] Department of Computer Science, University of Toronto, Toronto, Ontario, Canada
sheila@cs.toronto.edu

Abstract. Service interface description languages such as WSDL, and related standards, are evolving rapidly to provide a foundation for interoperation between Web services. At the same time, Semantic Web service technologies, such as the Ontology Web Language for Services (OWL-S), are developing the means by which services can be given richer semantic specifications. Richer semantics can enable fuller, more flexible automation of service provision and use, and support the construction of more powerful tools and methodologies. Both sets of technologies can benefit from complementary uses and cross-fertilization of ideas. This paper shows how to use OWL-S in conjunction with Web service standards, and explains and illustrates the value added by the semantics expressed in OWL-S.

1 Introduction

The promise of Web services and the need for widely accepted standards enabling them are by now well recognized, and considerable efforts are underway to define and evolve such standards in the commercial realm. In particular, the Web Services Description Language (WSDL) [5] is already well established as an essential building block in the evolving stack of Web service technologies, and is being developed and standardized in the W3C's Web Services Description Working Group [34]. WSDL, in essence, allows for the specification of the syntax of the input and output messages of a basic service, as well as other details needed for the invocation of the service. WSDL does not, however, support the specification of workflows composed of basic services. In this area, the Business Process Execution Language for Web Services (BPEL4WS) [1], under development at OASIS, has the most prominent status. The W3C's Web Services Choreography Working Group [33] also has been chartered to explore this technical area. With respect to registering Web services, for purposes of advertising and discovery, Universal Description, Discovery and Integration (UDDI) [32] has received the most attention to date.

At the same time, recognition is growing of the need for richer semantic specifications of Web services, so as to enable fuller, more flexible automation of service provision and use, support the construction of more powerful tools and methodologies, and promote the use of semantically well-founded reasoning about services. Because a rich representation language permits a more comprehensive specification of so many different aspects of services, they can provide a better foundation for a broad range of activities, across the Web service lifecycle. For example, richer semantics can support greater automation of service selection and invocation, automated translation of message content between heterogeneous interoperating services, automated or semi-automated approaches to service composition, and more comprehensive approaches to service monitoring and recovery from failure. Further down the road, richer semantics can help to provide fuller automation of such activities as verification, simulation, configuration, supply chain management, contracting, and negotiation of services.

To meet this need, researchers have been developing languages, architectures and related approaches; the resulting body of work goes under the heading of *Semantic Web services* [21]. In particular, the authors of this paper, members of the OWL-S Coalition, are developing the Ontology Web Language for Services (OWL-S) [25], which seeks to provide the building blocks for encoding rich semantic service descriptions, in a way that builds naturally upon OWL [19], the Semantic Web language undergoing standardization at the W3C.

OWL-S (formerly DAML-S) and other related work may be viewed as efforts to lay the foundations for the most effective evolution of Web service-related capabilities that can be supported with current and maturing technologies. But at the same time, our goal is to promote the rapid adoption of semantically expressive technologies that are already well-understood, and there is much that can be done in the near term. Therefore, we have taken pains to construct mechanisms by which OWL-S can be used along with the dominant Web services standards, such as WSDL. The purpose of this paper is to provide an initial roadmap towards deployment of Semantic Web services, using OWL-S in conjunction with WSDL and related standards, and to begin to provide a clear delineation of the potential benefits of richer semantics in specifying Web services.

In this paper, we show how to use OWL-S in conjunction with Web service standards — focusing particularly on its use with WSDL — and explain and illustrate the value added by the semantics expressed in OWL-S. We illustrate these points using a simple running example, which is presented in Section 2. Section 3 explains how OWL-S can be used to describe the example service, and can be *grounded* in the WSDL description. In Sections 4 – 6, we show how the combined OWL-S specifications can be used to support service enactment, service discovery, and service composition, respectively. Sections 7 and 8 present related work and a summary of our approach and its potential importance for the future of Web services.

2 A Motivating Example

Amazon.com provides an openly available Web service which allows client programs to browse Amazon's databases, locate books and other products and put them in a Web ‘shopping cart’ that can be accessed from the main Amazon Web site using a browser to finalize purchases. Web service client programs, written to avail themselves of the provided WSDL specification of the service, can request a wide range of semi-structured keyword searches on the Amazon Web site data base. Clients can search for books with a given author, products from a particular manufacturer, or DVDs of movies by a given director. Customer reviews to seller profiles are also accessible. (For more information visit http://www.amazon.com/webservices.)

Amazon provides a WSDL specification of its Web service describing the operations that can be performed, along with tutorials and code for sample clients. The tutorials and code samples are needed so that *programmers* can properly utilize the WSDL interface. No software system (agent) could read and utilize the WSDL interface without human assistance, because the WSDL specification language provide no means of including representations of the semantics of the defined operations and associated messages elements. For example, all of the inputs and outputs (parts of corresponding WSDL messages) in Amazon's WSDL operations are typed as strings. We take it as a key objective of Semantic Web services and OWL-S to bridge that gap. OWL-S provides a language for specifying the function (preconditions and effects) of an operation and semantic types for each of the inputs and outputs of the service. OWL-S is based on the assumption that the *definitions* of these semantic concepts are available at referenced URIs on the Semantic Web, so that the service and client programs have a means of sharing terms and clients can find the definitions of all referenced concepts, represented in the OWL semantic description language.

The result is that, by taking an OWL-S description of the services together with the WSDL description, a client program can distinguish the operation taking a model number of a camcorder from one requiring a book author's name in what would otherwise look to be similar request operations to search the database. The client can also properly interpret the result of those queries, without programming specific to that interface. By using OWL for semantic typing of the elements of communication, our Amazon client [26] can automatically identify which inputs (elements of its own internal goals) are required for the kind of search desired, *transform* those elements, if necessary, to the appropriate (string) form, and interpret the elements of a returned message.

The WSDL specification of the *outputs* of each call to the service similarly lacks semantic definition. All Amazon's defined search operations return results using the same data structure, named *Details*, regardless of what product information is requested. Product types can be inferred from the data structure by analyzing the elements that are filled in. For example, *Details* contains a field for *Authors* which is used to describe books, and a field *Directors* which is used to describe movies. It is up to the client to recognize that values in these fields indicate whether it is a book or a movie. Even if the type of item specified in a *Details* record were clearly identified in a Type field by the interface designer, WSDL provides no way uniform way of enabling such interpretations.

WSDL's lack of semantic descriptions of the meaning of inputs and outputs makes it impossible to develop software clients that can, without human assistance, dynamically find and successfully invoke a service. WSDL specifications of services must be interpreted by programmers, who interpret the names of keywords given for message elements using other supporting documentation to integrate *specific* services with their client applications. The objective of Semantic Web services is to support clients that can find and correctly utilize newly discovered services without additional programming. Such clients will, for example, be capable of finding sites selling books or CDs and comparing the prices of particular items from those sites even when those

CDs and comparing the prices of particular items from those sites even when those services' WSDL interfaces were not known, in advance, to the developer of that client. Semantic Web service clients will be able to interact with any such service as long as they describe their WSDL operations in terms of compatible, shared, Semantic Web representations for books, CDs, information requests, purchase/sale requests and monetary units. For the same reasons, these Web services can be discovered in a service repository using semantic descriptions characterizing the services proviced with no (or minimal) human intervention. Furthermore, both the discovery and use of such services is robust in the face of service design changes over time, because the service protocols would be republished and re-interpreted by the client software at the time of use.

3 Introducing Semantics

OWL-S (formerly DAML-S) is an OWL ontology with three interrelated subontologies, known as the profile, process model, and grounding. In brief, the profile is used to express "what a service does", for purposes of advertising, constructing service requests, and matchmaking; the process model describes "how it works", to enable invocation, enactment, composition, monitoring and recovery; and the grounding maps the constructs of the process model onto detailed specifications of message formats, protocols, and so forth (normally expressed in WSDL). This paper is primarily concerned with some of the fundamental constructs of the process model, and their groundings.

WSDL 1.1 allows for the specification of *operations* as the basic building blocks of Web services. (Although the development of WSDL 2.0 is well underway, it is not yet stable enough at time of writing to allow for OWL-S groundings based on it.) Operations provide the organizational structure around which input/output message syntax and patterns are specified. OWL-S provides an analogous but somewhat more abstract construct known as the *atomic process*, which is characterized primarily in terms of its inputs, outputs, preconditions, and effects (IOPEs).

The inputs and outputs of an atomic process are given types from the (class-hierarchical, description logic-based) typing system of OWL, which allows for the use of concepts defined and shared as part of the Semantic Web. For example, the accompanying code sample (Fig. 1) gives a simplified OWL-S declaration of an atomic process with its IO specifications. (Due to space constraints, we omit namespace qualifiers in this example). In this case, AtomicProcess, input, output, and parameterType belong to the OWL-S process model namespace. We assume that Human, BookTitle, and ISBN are classes defined in appropriate domain ontologies having other namespaces.

The grounding for this atomic process would establish its correspondence to a particular WSDL operation, and the correspondence of each IO element to a particular WSDL message part element. Also, if needed, the grounding could specify an XSLT script to transform each OWL-expressed input (an instance of the relevant class) to the precise syntactic form specified by WSDL, and vice versa for outputs. Additional details and examples of OWL-S groundings may be found in [14].

```
<AtomicProcess ID="AuthorSearch">
  <hasInput>
    <Input ID="Author">
      <parameterType resource="#Human">
    </Input>
  </hasInput>
  <hasInput>
    <Input ID="Title">
      <parameterType resource="#BookTitle">
    </Input>
  </hasInput>
  <hasOutput>
    <Output ID="BookID">
      <parameterType resource="#ISBN">
    </Output>
  </hasOutput>
```

Fig. 1. OWL-S declaration of an atomic process with its IO specifications

An important part of Semantic Web service description is the specification of conditions and constraints, including the preconditions and effects of a process or service. Preconditions are logical formulae that need to be satisfied (ensured to be true) by a service requestor prior to the execution of the service. Effects are logical formulae that state what will be true upon the successful execution of the service. OWL-S Effects are the side-effects of the execution of the service. Many information-providing services have no side effects. Nevertheless, other, often transaction-oriented, services do have side effects in the world, such as debiting the user's credit card, sending goods, etc. Description of these side effects is critical to certain aspects of Web service automation, as we discuss in subsequent sections.

For the specification of a process' preconditions and effects, OWL-S allows for the use of a more expressive language than OWL, such as RuleML [31], DRS [18], or the recently proposed OWL Rules Language [10]. For example, one of these languages could be used to express a precondition for a bookselling service, stating that one must have a valid account and a valid credit card in order to make a purchase.

A more complete exposition of OWL-S may be found at [25], and in the various papers listed there. In the following three sections, we discuss several case studies of OWL-S' contributions in the areas of service enactment, discovery, and composition.

4 Enactment

Enactment is the process by which a client applies a declarative description of a service to request something of the service and interpret the response. Here, the description being interpreted is the OWL-S process model published by the service along with the WSDL specs to which it is grounded. Enactment begins by reasoning backwards from the inputs required by the selected service to find the information available to the client that is required to successfully invoke the service. These input values are then mapped via the service grounding onto the corresponding elements of a WSDL message pattern, resulting finally in a message being communicated to the service. The output message (if any) is handled by essentially reversing the process.

A WSDL output message that is received by the client is transformed (again, via the grounding) into an OWL-S representation of the content of that message which can be interpreted by the client's reasoning engine.

To implement this process with Amazon's Web service, we first require an OWL-S description of that service that more fully represents the inputs and outputs of the service. We can partially automate the creation of this description by generating an initial OWL-S description of Amazon's Web service using tools that transform WSDL into a partial OWL-S specification [27]. Since the WSDL description does not contain sufficient information to form a complete OWL-S process model, we manually supplement the generated description in two steps:

(1) adding semantic descriptions of each input parameter to the generated process model, and supplying any (XSLT-based) data transformations needed to produce the corresponding grounded message parameter strings

(2) constructing a composite process model that links the various operations provided by the Web service into semantically meaningful message patterns (e.g., login before search before add-to-shopping-cart).

The resulting process model is given in Fig. 2, which shows the relationships between the various service operations represented in the resulting OWL-S process model.

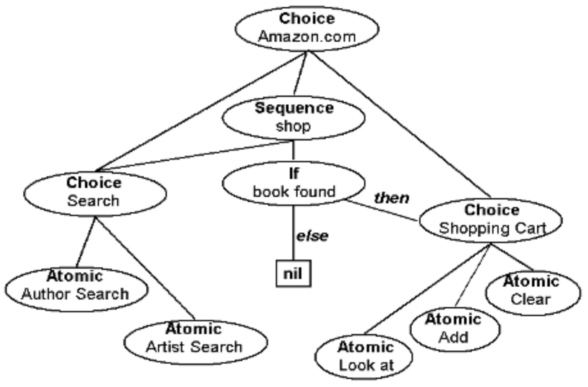


Fig. 2. Simplified process Model for Amazon Web Service

The client can perform three types of tasks: *search* the Amazon's data bases using author search, artist search or other types of searches; *view or modify the shopping cart* by adding new items, clearing it, or looking at its contents; or performing the composite *shopping* process that combines the other two by first searching and then adding the product found to the shopping cart. The WSDL description of the Amazon Web service only described the operations corresponding to the leaves of this graph.

Each of the OWL-S process descriptions specifies the semantic types of the data required as input, and returned as output. For example, the input of the author search should be an instance of class *Human* that stands in a particular relationship to the book being sought (*written-by).* The use of OWL classes and properties as constraints on the instances that must be identified for particular input values is critical to the inference process; it allows for the formulation of service requests without requiring a programmer to write special purpose code specific to each possible type of service request.

If the client has a goal to find the price of a particular book by searching, it can construct the appropriate elements of the search request by identifying the items (such as author) that are relevant, based on their relationship to the information about the book sought. Since the service to be invoked is selected because the right kind of information is described as part of the output of the service, and it describes this information as associated with database elements about books, the client can reason from that output description (ISBN of a book) back to the necessary input elements (author, title of the book whose ISBN is sought). As a consequence the client also knows what data will be returned with no need to guess from the instantiation of the *Details* data structure.

The OWL-S grounding takes care of the mapping from the concepts that describe the inputs and outputs of the processes to the inputs and outputs of the corresponding operations in the Amazon WSDL specification. As a result, while reasoning about the Web service can take advantage of the OWL logics and ontologies, the actual invocation is consistent with Amazon's requirements. Indeed, we are able to interact successfully with the Amazon Web site using the DAML-S Virtual Machine [26].

5 Discovery

Discovery is the process of finding Web services with a given capability. In general, discovery requires that Web services advertise their capabilities with a registry, and that requesting services query the registry for Web services with particular capabilities. The role of the registry is both to store the advertisements of capabilities, and to perform a match between the request and the advertisements. (Here we assume an infrastructure based on a centralized registry, because this is the type of infrastructure that is emerging for Web services. Nevertheless, our discussion generalizes to other architectures.)

The discovery process requires a language that can be used to encode Web service capabilities for advertisement and for requests. Furthermore, discovery requires a matching process that compares the advertisements with the requests to verify whether they describe matching capabilities.

In this section, we will describe how OWL-S may be used to express and match capabilities. Finally, we will show how OWL-S can be used to add capability matching to UDDI, the de-facto standard discovery registry for Web Services.

5.1 Representing Capabilities

Capabilities of Web services correspond to the functionalities provided by Web services. Broadly speaking, there are two ways to represent functionalities. The first approach provides an extensive ontology of functions where each class in the ontology corresponds to a class of homogeneous functionalities. A simple example of an ontology which specifies a taxonomy of e-services is shown below (Fig. 3). Using such an ontology, Web services such as Amazon may be defined as instances of classes that represent their capabilities. Amazon, for example, may advertise itself as a Bookselling service.

```
<owl:Class rdf:ID="e_Service">

<owl:Class rdf:ID="Information_Service">
 <rdfs:subClassOf rdf:resource="e_Service"/>
</owl:Class>

<owl:Class rdf:ID="SellingService">
   <rdfs:subClassOf rdf:resource="e_Service"/>
</owl:Class>

<owl:Class rdf:ID="BookSelling">
  <rdfs:subClassOf
      rdf:resource="SellingService"/>
</owl:Class>

<owl:Class rdf:ID="AirlineTicketing">
  <rdfs:subClassOf
       rdf:resource="SellingService"/>
</owl:Class>
```

Fig. 3. Example of an ontology which specifies a taxonomy of e-services

The second way to represent capabilities is to provide a generic description of function in terms of the state transformation that it produces. The latter is typically used by AI planning languages such as PDDL [17]. For example, Amazon may specify that it provides a service that requests a book title, author, address and a valid credit card number, and produces a state transition where the book is delivered to address, the credit card will be charged, and the book will change ownership. Despite their differences, both ways to represent capabilities use ontologies to provide the connection between what the Web service does and the general description of the environment in which the Web service operates.

There are trade-offs between the two representations of functionalities that help choose the representation by analyzing the task needs. The use of an explicit ontology of capabilities facilitates the discovery process since the matching process is reduced to subsumption between the capabilities in the ontology. On the other hand, enumerating all possible capabilities even in restricted domains for ontology encoding may be difficult. For example, consider the problem of representing translation services from a source language L_S to a target language L_T. Assuming n possible languages, there are n^2 possible types of translation services. A services taxonomy might have different classes of service for each pair of languages that could be translated, or it might just represent translation services as one general category, with explicit properties that allow particular services to describe the languages that they can translate from and translate to. This latter approach is consistent with describing the capability in terms of a state transformation. It distinguishes the translators by describing how they produce different kinds of results.

Note that describing the types of the inputs and outputs of such a service is not sufficient to distinguish capabilities. Consider, for example, a service that takes a geographic region as input and produces the names of different wines as output. This input/output couple can be used by two very different services: one that reports which wines are produced in a region, the other that reports the wines that are sold in a region.

OWL-S supports both views of the capabilities of Web services. The *Service Profile* module of OWL-S provides a high level descriptions of services as a transformation from one state to another. To this extent, at its core, a Service Profile provides a view of the Web service as a process which requires inputs, and some precondition to be valid, and it results in outputs and some effects to become true. Furthermore, OWL-S provides a schema by which Service Profiles can be subclassed to describe a specific class of capabilities such as translation services, or wine selling services. More precisely, a Service Profile provides two types of information: the first one is a *functional description* of the Web service in terms of the transformation that the Web service produces, the second one is a set of *non-functional properties* that specify constraints on the service provided. The functional description describes both the information transformation which results in the production of outputs from a set of inputs; and the state transformation that results in the generation of the effects starting from a state where the preconditions are satisfied. Non-functional properties specify the quality of service provided by the Web service, or its security requirements [7], such as the type of encryption and policies that apply.

Since OWL-S synthesizes both an extensional and functional view of Web services, it provides a complete description of the services that it describes. It can take advantage of ontologies of services and products wherever they exist to the extent that they are able to represent the capabilities of a Web service. Furthermore, it can make use of transformation produced by the Web service to provide a finer grain description of the Web service or to be able to describe the effects of using a Web service even when its capability does not correspond to any functional description.

5.2 Matching Capabilities

Capability matching compares the capabilities provided by any of the advertised services with the capabilities needed by the requester. The goal is to find the advertiser that produces the results required for the requester. In general it is unrealistic to expect that the capability offered will exactly match the query. For example, the requested service may be for stock quote information, and the task of the matching engine is to decide whether it can be accomplished by a service that provides financial news. The matchmaker should determine how likely it is that each capability advertisement indicates that the service will accomplish the particular function specified in the query.

A number of capability matching algorithms have been proposed for OWL-S. They use the service descriptions in the Service Profiles and the ontologies that are available to decide whether there is a match between service requests and the advertisements of the services provided. In general, they exploit one of the two views of the capabilities described above.

Matching algorithms, such as described in [11] and [12], assume the availability of ontologies of functionalities to express capabilities. Matching between the request and the available advertisements is reduced to their subsumption relation. Different degrees of match are detected depending on whether the advertisement and the request describe the same capability or whether one subsumes the other.

Other matching algorithms, such as in [28], [8], [2], and also again [13], assume that capabilities are described by the state transformation produced by the Web service. These matchmakers compare the state transformation described in each advertisement to the one described in the request. They perform two matches, one comparing outputs and one comparing inputs. If the output required by the requester is of a kind covered (subsumed) by the advertisement, then the inputs are checked. If the inputs specified in the request are subsumed by the input types acceptable to the service, then the service is a candidate to accomplish the requester's requirement.

In reality, there is an asymmetry between the matching of the inputs and the outputs of a Web service. Ultimately, the requester needs a Web service that produces the desired outputs. Once the Web service that provides the desired outputs has been found, the requester can either attempt to satisfy all the inputs, or use its own composition capabilities to find other Web services that can provide the desired inputs.

5.3 Relation with UDDI

UDDI (Universal Description Discovery and Integration) [32] is an industrial initiative whose goal is to create an Internet wide network of registries of Web services. UDDI allows businesses to register their presence on the Web by specifying their points of contact both in terms of the ports used by the service to process requests and in terms of the physical contacts with people that can answer questions about the service. In addition, UDDI provides a language to specify an unbounded set of features of services that can help the process of service location and selection as well as service invocation.

UDDI enjoys the support of many prominent software and hardware companies that invested heavily in Web services. Because of this support, UDDI is becoming the de facto standard repository of Web services. Despite its role, UDDI provides a very weak discovery mechanism which does not allow the discovery of any Web service only on the bases of what problems it provides.

The main problem with UDDI is that it does not provide a capability representation language such as the OWL-S Service Profile. As a consequence, UDDI does not provide capability based search. The result is that UDDI supports the location of information about the Web service, once it is known which Web service to use, but it is impossible to locate a Web service only on the basis of what problems it solves.

OWL-S and UDDI complement each other. UDDI provides a World Wide distributed registry that is virtually an industry standard. On the other side, OWL-S provides the information required for capability matching. The OWL-S/UDDI matchmaker [28] integrates OWL-S capability matching in the UDDI registry. This integration is based on the mapping of OWL-S Service Profiles into UDDI Web service representations [29] shown in Fig. 4. The mapping function defines a set of specialized UDDI TModels that store OWL-S information that cannot be represented in the standard UDDI Web Service representation. (TModels are an unbounded set of properties that can be associated with a Web service specification.)

The integrated OWL-S/UDDI provides all the functionalities provided by UDDI using exactly the same API, so that any UDDI can interact with it to retrieve information about available Web services. In addition, OWL-S/UDDI supports capability matching by taking advantage of OWL-S capability representation and the matching process proposed in [28]. The result is a UDDI in which it is possible to search, and find, Web services by their capabilities.

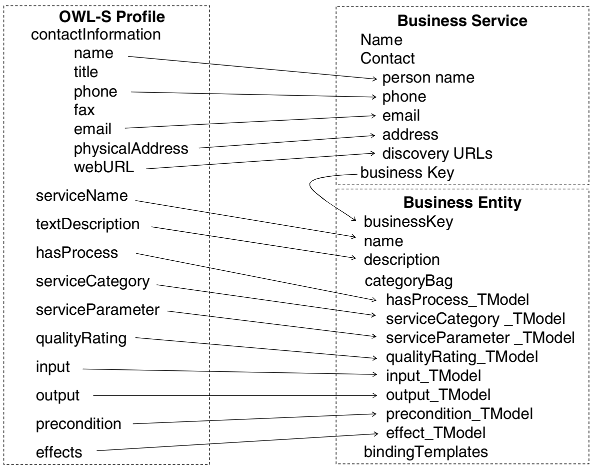


Fig. 4. OWL-S to UDDI mapping

6 Composition

Composition is the process of selecting, combining and executing Web services (WS) to achieve a user's objective. "Make the travel arrangements for my WWW2004 conference trip" or "Buy me an Apple iPod at the best available price" are examples of possible user objectives addressed by composition. Human beings perform manual WS composition by exploiting their cultural knowledge of what a Web service does (e.g, that www.apple.com will debit your credit card and send you an iPod), as well as information provided on the service's Web pages, in order to execute a collection of services to achieve some objective. To automate WS Composition, all this information must be encoded explicitly in an unambiguous computer interpretable form. None of the existing industrial standards for WS description encode this level of detail. Further, the descriptions they provide are not unambiguously computer interpretable and as a consequence not reliably manipulated by an automated reasoning system; hence the need for OWL-S.

Automated WS Composition is akin to both an AI planning problem and a software synthesis problem, and draws heavily on both of these areas of research [20]. In order to perform automated WS composition, a reasoning system must order, combine and execute Web services that collectively achieve the user's objective. This involves resolving constraints between Web service inputs, outputs, preconditions and effects (IOPEs) and (typically) the outputs and effects (OEs) the user desires. For example, if one starts with an agent's goal (some desired outputs and effects), and matches it to the outputs and effects of a Web service (modeled as a process), the result is an instantiation of the process, plus descriptions of new goals to be satisfied based on the inputs and preconditions of that process. The new goals (inputs and preconditions) then naturally match other processes (outputs and effects), so that composition arises naturally. The constraints between these inputs, outputs, preconditions and effects dictate the composition of Web services. Two types of composition problems can be distinguished: i) those that involve only information-providing services, and ii) those that involve both information-providing and world-altering services. The former requires a rich semantic representation of inputs and outputs (IO). The latter requires a like representation of IOPEs. Recall that the effects (E) are the side effects of the program (e.g., that www.apple.com will debit your credit card and send you an iPod). WS preconditions and (conditional) effects are not encoded in any existing industrial standard. They are encoded, in unambiguous computer-interpretable form in OWL-S. Since they supplement the information contained in WSDL, there is no grounding for these features at the WSDL level.

In addition to matching IOPEs, the automated WS Composition problem also can involve selecting from among alternative Web services that match the IOPE constraints of the composition problem. For example, there are many Web services from which a user can buy an iPod. In order to select from among alternative services, a composition engine also requires some form of service selection. This is akin to the discovery problem described in the previous section, and as argued there, requires a representation of the properties, capabilities and functioning of a Web service.

There are several different approaches to WS Composition. All characterize OWL-S processes as actions with inputs, outputs, preconditions and effects, and use planning technology to achieve WS composition. For example, the work of [16] models processes in the same format as a STRIPS operator [9] and plans from a sequence of Web services to achieve the user's goal. In principle the system can string together a series of actions to arrive at a novel plan for dealing with a Web service. However, the system as described is at a very early stage of development, and fails to address such basic problems as how to deal with unpredictable results of actions. [22] also investigates the use of plan synthesis for WS Composition, though their focus is on the specific problem of planning with existing composite Web services and the work reported is preliminary.

In contrast to this approach to WS Composition, several other researchers have taken the approach of using some sort of plan script or task model that describes approximately *how to* achieve some objective. This high-level plan is expanded and refined using automated reasoning machinery. The first such system to be built was the Golog system (e.g., [20], [21]). It models both world-altering and information-providing services as actions with IOPEs, uses Golog procedures (modeled as OWL-S composite processes) to represent generic procedures of approximately *how-to* perform tasks, and uses interleaving online deductive synthesis and execution to generate a sequence of Web services customized to user's preferences and constraints. Information gathering actions are executed as necessary, while world-altering actions are projected or simulated in order to enable the system to deliberate before committing to the execution of world-altering services.

In a similar spirit, several other researchers (e.g., [35],[30]) have used the paradigm of Hierarchical Task Network (HTN) planning (e.g., [23]) to perform automated WS composition. In this paradigm, a planner is supplied with a library of standard plans, each characterized by what it is supposed to accomplish (that is, effects given preconditions)[35] uses the SHOP2 system (e.g., [23] [24]), which is a state-of-the-art HTN planner. To solve a composition problem, SHOP2 must be given a top-level sketch of the composed plan (encoded in OWL-S as a CompositeProcess). However, many of the steps in the plan are described in a high-level vocabulary (analogous to the OWL-S control constructs) that allows multiple alternative subplans to carry out those steps.

The system searches through ways of combining those subplans in order to arrive at an overall plan. Central to the SHOP2 approach to planning with Web Services is the exploitation of the sharp distinction between information-providing and world-altering services in the planning process, given that the information provided by services is often critical to finding a plan. When mapping from a set of OWL-S service descriptions to a SHOP2 domain, information-gathering services are detected and encoded so as to be executed at planning time, rather than at run time (as so-called "book-keeping" operators, or, in current work, as SHOP2 evaluated preconditions). [20],[21],[30] also execute information-gathering services at plan time to reduce the search space for plans and to reduce non-determinism.

HTN planning has also been used in [30] to compose Web services in the travel domain and in the organization of a B2B supply chain. The basic idea explored in this work is that Web services expand their own capabilities through collaboration. Consistently with the work presented above, especially [16] and [35], during the planning process, outputs and preconditions are satisfied either directly using an action that the Web service can perform or by asking other Web services to do something that satisfies that output or precondition. Locating appropriate Web services can be done using the OWL-S/UDDI matchmaker as discussed in Section 5.

There are many systems that deal with the restricted problem of composing services without consideration of preconditions and effects (PEs). Included in these is the work of [12] that augments BPEL4WS, a popular business-process language [1], with a composition module. When the BPEL4WS process requires a certain input, described as an XML data type, their system searches for a WS that can translate from available formats to the desired format. For example, if the process declares a need for a complex type containing a date in US format, and a known service supplies a data type identical except that the date is in UK format, the system searches for a translation service that can perform the desired data transformation. If necessary, it breaks the transformation process into substeps and recursively searches for methods to accomplish the substeps. A similar approach is integrated with an end-user interactive composition system, STEER described in [15]. These approaches represent prototype solutions to an important subtask of service composition, namely, *data-transfer interoperation.* For it to work, it is necessary for process descriptions to include rich, computer-interpretable descriptions of the inputs and outputs of a process — the IO half of IOPEs.

While this early work is promising, we are still some distance from the goal of automated WS composition. We have argued that we need rich, representations of Web services in a language with a well-defined semantics, to enable automated WS composition. Specifically, we require rich, declarative descriptions of Web service IOPEs to determine a composition, and we require rich representations of the properties, capabilities and functioning of services to enable WS selection during the composition process. We have achieved both these requirements in great measure with OWL-S. In contrast, current industrial standards for WS description only describe WS inputs and outputs and they do so in a language that is not richly expressive and is without a well-defined semantics.

We also require rich declarative representations of composite processes (existing compositions of Web services, such as Amazon's workflow) so that we can exploit them in our WS composition tasks. (Many of the existing WS composition technologies only compose atomic processes.) We have addressed the problem of describing composite processes in OWL-S, but we believe the solution can be improved upon by appealing to a language that is more expressive than OWL, leveraging emerging industrial process modeling standards. To realize the goal of automated WS composition, we also require further advances in automated reasoning/planning technology for WS. A final barrier to the goal of automated WS composition is the need for widespread adoption of OWL-S WS descriptions.

Despite the need for further work, the accomplishments of OWL-S and associated composition technologies provide immediate value-added. With existing technology we can perform automated composition of information-gathering services. It has also been demonstrated [12] that we can augment existing industrial WS choreography and orchestration tools with composition technology for data-transfer interoperation and for run-time binding of Web services. These systems enable manual composition of WSs. We can augment this with some semantic integration of the data sources. Finally, as demonstrated, we can currently perform automated WS composition of both information-gathering and atomic world-altering services under controlled conditions. Automated WS Composition is at the heart of seamless interoperation among Web services. With adoption of approaches to WS description such as OWL-S and advances in planning-related technologies, we believe that broad-scale automated WS composition is well within reach.

7 Related Work

Throughout this paper we have identified related work that exploits OWL-S (or DAML-S, the name by which earlier versions of OWL-S were known). Here we briefly note other work on Semantic Web Services that does not use DAML-S or OWL-S to describe Web services.

Most of the work on discovery of Web services using the Semantic Web has been based on OWL-S. Nevertheless, other work on discovery does not assume OWL-S, most notably [2] which bases Web service descriptions on the MIT Process Handbook [12]. In this work, the matching process is based on the workflow description of the process model of a Web service rather than an abstract representation such as the OWL-S Profile. The retrieval mechanism maps the request against all the process models advertised by available services until only the process models that match the request are retrieved.

The matching process allows the requester to ask for Web services that “do X before Y”. In other words, the requester can constrain not only the type of process it performs and the results that it achieves, but also the way in which a service is achieved. Implicitly, it also assumes that the requester and the provider have a shared and intimate knowledge on how processes are performed. In turn, this assumes that the provider and the requester should share ontologies such as the MIT Process Handbook. While OWL-S does not make such strong assumptions on the ontologies needed for discovery, when those assumptions are known to hold, results similar to those obtained in [2] can be obtained by using the matching processes suggested for OWL-S, by first selecting Web services with a given capability and then selecting those services whose process model satisfies the temporal constraint.

In the area of WS Composition, most of the early work has exploited OWL-S. More recently, researchers from the planning community (e.g., [1]) have begun to examine the WS Composition problem; however, most have not explicitly addressed the problem of how to describe Web services, beyond modeling service IOPEs as actions in first-order logic, propositional logic or PDDL [17].

8 Summary

Our objective in this paper has been to show how OWL-S can be put to use in the near-term, in the context of emerging Web service standards such as WSDL, UDDI and BPEL. We have explained some of the basics of OWL-S, and the techniques by which it can be used in conjunction with these standards; and we have given an overview of projects that have employed OWL-S in combination with one or more of them.

We have discussed the benefits of the richer service descriptions supported by OWL-S, focusing primarily on the descriptions of inputs, outputs, preconditions, and effects of services. In the area of enactment, OWL-S supports the specification of composite processes, and allows for flexible, robust invocation and interoperation between service clients and providers. In addition, OWL-S grounding mechanisms allow process descriptions and enactment procedures to be used in conjunction with WSDL. In the area of discovery, OWL-S allows service registries and matchmaking algorithms to take advantage of two distinct styles of ontology-based characterization of services, whose use may be integrated with UDDI. In the area of service composition, a variety of approaches exist to reason about OWL-S IOPEs, in support of manual, semi-automated, and, under controlled conditions, automated composition of both information-gathering and world-altering services.

In conclusion, OWL-S can help to enable fuller automation and dynamism in many aspects of Web service provision and use, support the construction of powerful tools and methodologies, and promote the use of semantically well-founded reasoning about services.

References

[1] J. L. Ambite (Ed.). *Proceedings of the ICAPS2003 Workshop on Planning for Web Services*, 2003.

[2] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte (Editor), Ivana Trickovic, Sanjiva Weerawarana. Business Process Execution Language for Web Services, Version 1.1, 2003. At http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.

[3] A. Bernstein and M. Klein. High Precision Service Retrieval. In *Proceedings of the First International Semantic Web Conference* (ISWC 2002), Sardegna, 2002.

[4] Boualem Benatallah, Mohand-Said Hacid, Christophe Rey and Farouk Toumani. Request Rewriting-Based Web Service Discovery. In *Proceedings of the Second International Semantic Web Conference* (ISWC 2003), pp 335-350, October 2003.

[5] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana. Web Services Description Language (WSDL) 1.1, 2001. At http://www.w3.org/TR/2001/NOTE-wsdl-20010315

[6] M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P.F. Patel-Schneider, L. A. Stein. Web Ontology Language (OWL) W3C Reference version 1.0, 18 August 2003. At http://www.w3.org/TR/2002/WD-owl-ref-20021112.

[7] Grit Denker, Lalana Kagal, Tim Finin, Massimo Paolucci, Naveen Srinivasan and Katia Sycara. Security For DAML Web Services: Annotation and Matchmaking. In *Proceedings of the Second International Semantic Web Conference* (ISWC 2003), pp. 335-350, October 2003.

[8] Tommaso Di Noia, Eugenio Di Sciacio, Francesco M. Donini and Marina Mongiello. Semantic Matchmaking in a P-2-P Electronic Marketplace. SAC 2003, pp. 582-586, 2003.

[9] R. Fikes and N. J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence 2*, pp. 189-208, 1971.

[10] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet. OWL Rules Lan guage, Draft version . Technical report, 29 October 2003

[11] Lei Li and Ian Horrocks. A Software Framework for Matchmaking Based on Semantic Web Technology. In *Proc. of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 331-339, ACM, 2003.

[12] T. W. Malone, K. Crowston, B. P. Jintae Lee, C. Dellarocas, G. Wyner, J. Quimby, C. S. Osborn, A. Bernstein, G. Herman, M. Klein, and E. O'Donnell. Tools for Inventing Organizations: Toward a Handbook of Organizational Processes. *Management Science*, 45(3):425--443, March, 1997.

[13] Daniel J. Mandell and Sheila A. McIlraith. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. . In *Proceedings of the Second International Semantic Web Conference* (ISWC2003), pp. 227--241, 2003

[14] David Martin, Mark Burstein, Ora Lassila, Massimo Paolucci, Terry Payne, Sheila McIlraith. Describing Web Services using OWL-S and WSDL. October 2003. At http://www.daml.org/services/owl-s/1.0/owl-s-wsdl.html

[15] Ryusuke Masuoka, Yannis Labrou, Bijan Parsia, and Evren Sirin. Ontology-Enabled Pervasive Computing Applications. In *IEEE Intelligent Systems*, 18(10):68-72, 2003.

[16] D. McDermott. Estimated-Regression Planning for Interaction with Web Services. In *Proceedings of the Sixth International Conference on AI Planning and Scheduling,* pp. 204—211, 2002.

[17] D McDermott. The Planning Domain Definition Language Manual. *Yale Computer Science Report 1165* (CVC Report 980003), 1998.

[18] D. McDermott and D. Dou . Representing Disjunction and Quantifiers in RDF. *Proceedings of the First International Semantic Web Conference* (ISWC2002), 2002.

[19] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. World Wide Web Consortium (W3C) Candidate Recommendation. August 18, 2003. At http://www.w3.org/TR/owl-features/

[20] S. McIlraith and T. Son. Adapting Golog for Composition of Semantic Web Services. In *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning* (KR2002), pp. 482-493, 2002.

[21] S. McIlraith., T.C. Son and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems, Special Issue on the Semantic Web*, 16(2):46--53, March/April, 2001.

[22] S. McIlraith and R. Fadel. Planning with Complex Actions. In *Proceedings of the Ninth International Workshop on Non-Monotonic Reasoning (NMR2002)*, pages 356-364, April, 2002.

[23] D. S. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila. SHOP: Simple Hierarchical Ordered Planner. In *Proceedings of the International Joint Conference on Artificial Intelligence* (IJCAI-99), pp.968—973, 1999.

[24] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F. Yaman 2003 SHOP2: An HTN Planning System. To appear, *Journal Artificial Intelligence Research.*

[25] OWL-S Coalition. OWL-S 1.0 Release. At http://www.daml.org/services/owl-s/1.0/
[26] M. Paolucci, A. Ankolekar, N. Srinivasan, and K. Sycara. The DAML-S Virtual Machine. In *Proceedings of the Second International Semantic Web Conference* (ISWC 2003), pp 335-350, October 2003.
[27] M. Paolucci, N. Srinivasan, K. Sycara, and T. Nishimura. Toward a Semantic Choreography of Web services: from WSDL to DAML-S. In *Proceedings of ICWS03*, 2003.
[28] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Semantic Matching of Web Services Capabilities. In *Proceedings of the First International Semantic Web Conference* (ISWC2002), 2002.
[29] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Importing the Semantic Web in UDDI. In *Proceedings of E-Services and the Semantic Web* (ESSW02), 2002.
[30] Massimo Paolucci, Katia Sycara, and Takahiro Kawamura. Delivering Semantic Web Services. In *Proceedings of the Twelfth World Wide Web Conference (WWW2003)*, Budapest, Hungary, May 2003, pp 111- 118.
[31] The Rule Markup Initiative. At http://www.dfki.uni-kl.de/ruleml/.
[32] The Universal Description, Discovery and Integration (UDDI) protocol. Version 3, 2003. At http://www.uddi.org/
[33] Web Services Choreography Working Group. At http://www.w3.org/2002/ws/chor/
[34] Web Services Description Working Group. At http://www.w3.org/2002/ws/desc/
[35] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S Web Services Composition Using SHOP2. In *Proceedings of the Second International Semantic Web Conference* (ISWC2003), 2003.

A Survey of Automated Web Service Composition Methods

Jinghai Rao and Xiaomeng Su

Norwegian University of Science and Technology,
Department of Computer and Information Science,
N-7491, Trondheim, Norway
{jinghai, xiaomeng}@idi.ntnu.no

Abstract. In today's Web, Web services are created and updated on the fly. It's already beyond the human ability to analysis them and generate the composition plan manually. A number of approaches have been proposed to tackle that problem. Most of them are inspired by the researches in cross-enterprise workflow and AI planning. This paper gives an overview of recent research efforts of automatic Web service composition both from the workflow and AI planning research community.

1 Introduction

Web services are considered as self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Nowadays, an increasing amount of companies and organizations only implement their core business and outsource other application services over Internet. Thus, the ability to efficiently and effectively select and integrate inter-organizational and heterogeneous services on the Web at runtime is an important step towards the development of the Web service applications. In particular, if no single Web service can satisfy the functionality required by the user, there should be a possibility to combine existing services together in order to fulfill the request. This trend has triggered a considerable number of research efforts on the composition of Web services both in academia and in industry.

In the research related to Web services, several initiatives have been conducted with the intention to provide platforms and languages that will allow easy integration of heterogeneous systems. In particular, such languages as Universal Description, Discovery, and Integration (UDDI) [4], Web Services Description Language (WSDL) [9], Simple Object Access Protocol (SOAP) [6] and part of DAML-S [14] ontology (ServiceProfile and ServiceGrounding), define standard ways for service discovery, description and invocation (message passing). Some other initiatives such as Business Process Execution Language for Web Service (BPEL4WS) [2] and DAML-S ServiceModel, are focused on representing service compositions where flow of a process and bindings between services are known a priori.

Despite all these efforts, the Web service composition still is a highly complex task, and it is already beyond the human capability to deal with the whole process manually. The complexity, in general, comes from the following sources. First, the number of services available over the Web increases dramatically during the recent years, and one can expect to have a huge Web service repository to be searched. Second, Web services can be created and updated on the fly, thus the composition system needs to detect the updating at runtime and the decision should be made based on the up to date information. Third, Web services can be developed by different organizations, which use different concept models to describe the services, however, there does not exist a unique language to define and evaluate the Web services in an identical means.

Therefore, building composite Web services with an automated or semi-automated tool is critical. To that end, several methods for this purpose have been proposed. In particular, most researches conducted fall in the realm of workflow composition or AI planning.

For the former, one can argue that, in many ways, a composite service is similar to a workflow [8]. The definition of a composite service includes a set of atomic services together with the control and data flow among the services. Similarly, a workflow has to specify the flow of work items. The current achievements on flexible workflow, automatic process adaption and cross-enterprise integration provide the means for automated Web services composition as well. In addition, the dynamic workflow methods provide the means to bind the abstract nodes with the concrete resources or services automatically.

On the other hand, dynamic composition methods are required to generate the plan automatically. Most methods in such category are related to AI planning and deductive theorem proving. The general assumption of such kind of methods is that each Web service can be specified by its preconditions and effects in the planning context. Firstly, a Web service is a software component that takes the input data and produces the output data. Thus the preconditions and effects are the input and the output parameters of the service respectively. Secondly, the Web service also alters the states of the world after its execution. So the world state pre-required for the service execution is the precondition, and new states generated after the execution is the effect. A typical example is a service for logging into a Web site. The input information is the username and password, and the output is a confirmation message. After the execution, the world state changes from "not_logged_in" to "logged_in". The "logged_in" state will be keeping until the "log_out" service is invoked. If the user can specify the preconditions and effects required by the composite service, a plan or process is generated automatically by logical theorem prover or AI planners without knowledge of predefined workflow. During the planning, the business logic can provide constraints in the planning setting.

In this paper we will present an overview of recent methods that provide automation to Web service composition. The automation means that either the method can generate the process model automatically, or the method can locate the correct services if an abstract process model is given. Some methods based on workflow have been reported in the work by Benatallah [5], but to our knowledge, no overview on service composition methods related to AI planning has been published yet. As a result, in the paper, we will have more focus on the AI planning methods than the workflow based methods.

This paper is organized as follows. Section 2 presents an abstract framework for Web service composition. Section 3 is the introduction of automatic Web service composition based on workflow methods. Section 4 provides an overview and comparison for the selected composition methods based on AI planning. The last section concludes the paper.

2 Web Services Composition Framework

Here, we propose a general framework for automatic Web services composition. This framework is in high-level abstraction, without considering a particular language, platform or algorithm used in composition process. The aim of the framework is to give the basis to discuss similarities and differences of the available service composition methods. In addition, we also use the framework to unify the terms used in the paper.

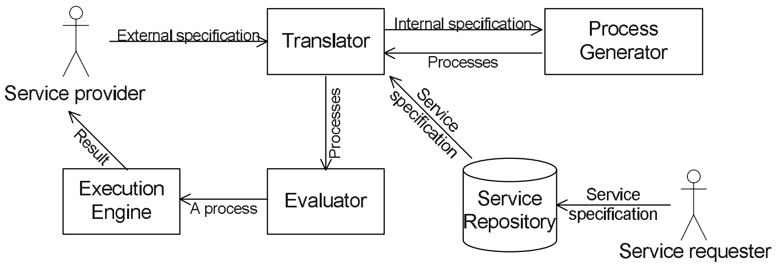


Fig. 1. The framework of the service composition system

A general framework of the service composition system is illustrated in Fig. 1. The composition system has two kinds of participants, service provider and service requester. The service providers propose Web services for use. The service requesters consume information or services offered by service providers. The system also contains the following components: translator, process generator, evaluator, execution engine and service repository. The translator translates between the external languages used by the participants and the internal languages used by the process generator. For each request, the process generator tries to generate a plan that composes the available services in the service repository to fulfill the request. If more than one plan is found, the evaluator evaluates all plans and proposes the best one for execution. The execution engine executes the plan and returns the result to the service provider.

Most precisely, the process of automatic service composition includes the following phases:

Presentation of Single Service: firstly, the service providers will advertise their atomic services at a global market place. There are several languages available for advertising, for example, UDDI [4] or DAML-S ServiceProfile [14]. The essential attributes to describe a Web service include the signature, states and the non-functional values. The signature is represented by the service's inputs, outputs and exceptions. It provides information about the data transformation during the execution of a Web service. The states are specified by precondition and postcondition. We model it as the transformation from one set of states to another in the world. Non-functionality values are those attributes that are used for evaluating the services, such as the cost, service quality and security issues.

Translation of the Languages: most service composition systems distinguish between the external and internal service specification languages. The external languages are used by the service users to enhance accessibility of the users in the sense that the users can express what they can offer or what they want in a relatively easy manner. They are usually different from the internal ones that are used by the composition process generator, because the process generator requires more formal and precise languages, for example, the logical programming languages. So far, the users have already get used to the standard Web service languages, such as WSDL and DAML-S. Thus the translation components between the standard Web service languages and the internal languages have to be developed.

Generation of Composition Process Model: in the meantime, the service requester can also express the requirement in a service specification language. A process generator then tries to solve the requirement by composing the atomic services advertised by the service providers. The process generator usually takes the functionalities of services as input, and outputs process model that describes the composite service. The process model contains a set of selected atomic services and the control flow and data flow among these services.

Evaluation of Composite Service: it is quite common that many services have the same or similar functionalities. So it is possible that the planer generates more than one composite service fulfilling the requirement. In that case, the composite services are evaluated by their overall utilities using the information provided from the non-functional attributes. The most commonly used method is utility functions. The requester should specify weights to each non-functionality attributes and the best composite service is the one who is ranked on top.

Execution of Composite Service: after a unique composite process is selected, the composite service is ready to be executed. Execution of a composite Web service can be thought as a sequence of message passing according to the process model. The dataflow of the composite service is defined as the actions that the output data of a former executed service transfers to the input of a later executed atomic service.

In the following we will give a survey on the methods used for the process generator to generate the process. The methods can be either fully automated or semi-automated.

3 Web Service Composition Using Workflow Technique

In the workflow-based composition methods, we should distinguish the static and dynamic workflow generation. The static one means that the requester should build an abstract process model before the composition planning starts. The abstract process model includes a set of tasks and their data dependency. Each task contains a query clause that is used to search the real atomic Web service to fulfill the task. In that case, only the selection and binding of atomic Web service is done automatically by program. The most commonly used static method is to specify the process model in graph. On the other hand, the dynamic composition both creates process model and selects atomic services automatically. This requires the requester to specify several constraints, including the dependency of atomic, the user's preference and so on.

EFlow[7] is a platform for the specification, enactment and management of composite services. EFlow uses a static workflow generation method. A composite service is modeled by a graph that defines the order of execution among the nodes in the process. The graph is created manually but it can be updated dynamically. The graph may include service, decision and event nodes. Service nodes represent the invocation of an atomic or composite service, decision nodes specify the alternatives and rules controlling the execution flow, and event nodes enable service processes to send and receive several types of events. Arcs in the graph denote the execution dependency among the nodes. Although the graph should be specified manually, EFlow provides the automation to bind the nodes with concrete services. The definition of a service node contains a search recipe that can be used to query actual service either at process instantiation time or at runtime. As the service node is started, the search recipe is executed, returning a reference to a specific service. In particular, the search recipe is resolved each time when a service node is activated. They do so because the availability of services may change very frequently in a highly dynamic environment. In [8], the authors further refine the service composition platform and propose a prototype of composite service definition language(CSDL). An interesting feature of CSDL is that it distinguishes between invocation of services and operations within a service. It provides the adaptive and dynamic features to cope with the rapidly evolving business and IT environment in which Web services are executed.

Polymorphic Process Model (PPM)[23] uses a method that combines the static and dynamic service composition. The static setting is supported by reference process-based multi-enterprise processes, the processes that consist of abstract subprocesses, i.e., subprocesses that have functionality description but lack implementation. The abstract subprocesses are implemented by service and bined at runtime. This is similar to the service binding in EFlow. The dynamic part of PPM is supported by service-based processes. Here, a service is modeled by a state machine that specifies that possible states of a service and their transitions. Transitions are caused by service operation(also called service activity) invocations or internal service transitions. In the setting, the dynamic service composition is enabled by the reasoning based on state machine.

4 Web Service Composition Using AI Planning

Many research efforts tackling Web service composition problem via AI planning have been reported. In general, a planning problem can be described as a five-tuple $\langle S, S_0, G, A, \Gamma \rangle$, where S is the set of all possible states of the world, $S_0 \subset S$ denotes the initial state of the world, $G \subset S$ denotes the goal state of the world the planning system attempts to reach, A is the set of actions the planner can perform in attempting to change one state to another state in the world, and the translation relation $\Gamma \subseteq S \times A \times S$ defines the precondition and effects for the execution of each action.

In the terms of Web services, S_0 and G are the initial states and the goal states specified in the requirement of Web service requesters. A is a set of available services. Γ further denotes the state change function of each service.

DAML-S (also called OWL-S in the most recent versions) is the only Web service language that announces the directly connection with AI planning. The state change produced by the execution of the service is specified through the precondition and effect properties of the ServiceProfile in DAML-S. Precondition presents logical conditions that should be satisfied prior to the service being requested. Effects are the result of the successful execution of a service. Since DAML+OIL, the language used to build DAML-S, uses Description Logics [10] as its logical foundation, DAML+OIL has the express power allowing for logical expressions. The majority of the methods reported in this survey use DAML-S as the external Web service description language. There are also a couple of methods that use WSDL or their own languages.

In the following we introduces a list of Web service composition methods based on AI planning. This kind of methods have been reported frequently in recent years, so we can not claim that we have an exhaustive list of the methods. We further classify the methods into five categories, namely, the situation calculus, the Planning Domain Definition Language (PDDL), rule-based planning, the theorem proving and others.

4.1 Situation Calculus

McIlraith et. al. [17, 19, 16] adapt and extend the Golog language for automatic construction of Web services. Golog is a logic programming language built on top of the situation calculus. The authors address the Web service composition problem through the provision of high-level generic procedures and customizing constraints. Golog is adopted as a natural formalism for representing and reasoning about this problem.

The general idea of this method is that software agents could reason about Web services to perform automatic Web service discovery, execution, composition and inter-operation. The user's request (generic procedure) and constraints can be presented by the first-order language of the situation calculus(a logical language for reasoning about action and change). The authors conceive each Web service as an action - either a *PrimitiveAction* or a *ComplexAction*. Primitive actions are conceived as either world-altering actions that change the state of the world or information-gathering actions that change the agent's state of knowledge. Complex actions are compositions of individual actions. The agent knowledge base provides a logical encoding of the preconditions and effects of the Web service actions in the language of the situation calculus. The agents use procedural programming language constructs composed with concepts defined for the services and constraints using deductive machinery. A composite service is a set of atomic services which connected by procedural programming language constructs(if-then-else, while and so forth).

The authors also propose a way to customize Golog programs by incorporating the service requester's constraints. For example, the service requester can use the nondeterministic choice to present which action is selected in a given situation, or use the sequence construct to enforce the execution order between two action. The generation of the plan have to obey the predefined constraint.

4.2 PDDL

A strong interest to Web service composition from AI planning community could be explained roughly by similarity between DAML-S and PDDL representations. PDDL is widely recognized as a standardized input for state-of-the-art planners. Moreover, since DAML-S has been strongly influenced by PDDL language, mapping from one representation to another is straightforward (as long as only declarative information is considered). When planning for service composition is needed, DAML-S descriptions could be translated to PDDL format. Then different planners could be exploited for further service synthesis.

In presenting the Web service composition method based on PDDL, McDermott [15] introduces a new type of knowledge, called *value of an action*, which persists and which is not treated as a truth literal. From Web service construction perspective, the feature enables us to distinguish the information transformation and the state change produced by the execution of the service. The information, which is presented by the input/output parameters are thought to be reusable, thus the data values can be duplicated for the execution of multiple services. Contrarily, the states of the world are changed by the service execution. We interpret the change as that the old states disappear and the new states are produced.

To deal with this issue is critical for Web service composition using AI planning because usually in AI planning, closed world assumption is made, meaning that if a literal does not exist in the current world, its truth value is considered to be *false*. In logic programming this approach is called *negation as failure*. The main trouble with the closed world assumption, from Web service construction perspectives, is that merely with truth literals we cannot express that new information has been acquired. For instance, one service requester might want to describe that after sending a message to a Web service, an identity number to the message will be generated. Thus during later communication the ID could be used.

4.3 Rule-Based Planning

Medjahed [18] present a technique to generate composite services from high-level declarative description. The method uses composability rules to determine whether two services are composable. The composition approach consists of four phases. First, the specification phase enables high-level description of the desired compositions using a language called Composite Service Specification Language(CSSL). Second, the matchmaking phase uses composability rules to generate composition plans that conform to service requester's specifications. The third phase is selection phase. If more than one plan is generated, in the selection phase, the service requester selects a plan based on quality of composition (QoC) parameters (e.g. rank, cost, etc.). The final phase is the generation phase. A detailed description of the composite service is automatically generated and presented to the service requester.

Here, we should pay more emphasis on the composability rules because it is the major issue to define how the plan is generated. The composability rules consider the syntactic and semantic properties of Web services. Syntactic rules include the rules for operation modes and the rules for binding protocols of interacting services. Semantic rules include the following subset: (1) message composability defines that two Web services are composable only if the output message of one service is compatible with the input message of another service; (2) operation semantic composability defines the compatibility between the domains, categories and purposes of two services; (3) qualitative composability defines the requester's preferences regarding the quality of operations for the composite service; and (4) composition soundness considers whether a composition of services is reasonable. To this end, the authors introduce the notion of composition templates that define the dependency between the different kinds of services.

The main contribution of this method is the composability rules, because they define the possible Web service's attributes that could be used in service composition. Those rules can be used as a guideline for other Web service methods based on planning.

SWORD [20] is another developer toolkit for building composite Web services using rule-based plan generation. SWORD does not deploy the emerging service-description standards such as WSDL and DAML-S, instead, it uses Entity-Relation (ER) model to specify the Web services. In SWORD, a service is modeled by its preconditions and postconditions. They are specified in a world model that consists of entities and relationships among entities. A Web service is represented in the form of a Horn rule that denotes the postconditions are achieve if the preconditions are true. To create a composite service, the service requester only needs specify the initial and final states for the composite service, then the plan generation can be achieved using a rule-based expert system. Besides the general composition methods, an interesting work done by SWORD is that the authors give a discussion on that the rule-based chaining can sometimes generate "uncertain" results if a precondition can not uniquely determines a postcondition. The authors argue that the uncertain results can avoid only when the preconditions are functionally depending on the postconditions inside a service. In fact, it may happen in most service composition methods described in this survey but not all authors explicitly declare it.

4.4 Other AI-Planning Methods

Some other AI planning techniques are proposed for the automatic composition of Web services. In [26] the SHOP2 planner is applied for automatic composition of Web services, which are provided with DAML-S descriptions. SHOP2 is an Hierarchical Task Network(HTN) planner. The authors believe that the concept of task decomposition in HTN planning is very similar to the concept of composite process decomposition in DAML-S process ontology. The authors also claim that the HTN planner is more efficient than other planning language, such as Golog. In their paper, the authors give a very detail description on the process of translating DAML-S to SHOP2. In particular, most control constructs can be expressed by SHOP2 in an explicit way.

Sirin et al [24] present a semi-automatic method for web service composition. Each time when a user has to select a Web service, all possible services, that match with the selected service, are presented to the user. The choice of the possible services is based both on functionalities and non-functional attributes. The functionalities (parameters) are presented by OWL classes and OWL reasoner is applied to match the services. A match is defined between two services that an output parameter of one service is the same OWL class or subclass of an input parameter of another service. The OWL inference engine can order the matched services so that the priority of the matches are lowered when the distance between the two types in the ontology tree increases. If more than one match is found, the system filters the services based on the non-functional attributes that are specified by the user as constraints. Only those services who pass the non-functional constraints can be presented to the service requester. The idea of semi-automatic service composition is quite interesting because it is very difficult to capture behavior in sufficient detail and compose the services in a fully automatic way, especially for the commercial-grade services. Although the proposed method is simple, it indicates the trend that automatic planner and human being can work together to generate the composite service for the user's request.

4.5 Theorem Proving

Waldinger [25] elaborates an idea for service synthesis by theorem proving. The approach is based on automated deduction and program synthesis and has its roots in his earlier work [13]. Initially available services and user requirements are described in a first-order language, related to classical logic, and then constructive proofs are generated with Snark theorem prover. Finally, service composition descriptions are extracted from particular proofs.

Lämmermann [12] applies Structural Synthesis of Program (SSP) for automated service composition. SSP is a deductive approach to synthesis of programs from specifications. The specifications of services only include the structural properties, i.e. the input/output information. SSP uses propositional variables as identifiers for input/output parameters and uses intuitionistic propositional logic for solving the composition problem. The composition is based on the proofs-as-programs property of intuitionistic logic. It equates the program of service composition to the problem of proof search. The author also takes takes advantage of disjunctions in classical logic to describe exceptions, which could be thrown during service invocation.

Rao et. al. [21, 22] introduces a method for automatic composition of semantic Web services using Linear Logic theorem proving. The method uses semantic Web service language (DAML-S) for external presentation of web services. And, internally, the services are presented by extralogical axioms and proofs in Linear Logic. Linear Logic, as a resource conscious logic, enables people to define attributes of Web services formally (including qualitative and quantitative values of non-functional attributes). In addition, Linear Logic has close relationship with π-calculus, which is the formal foundation of many Web service composition languages. The view of a Linear Logic proof as a π-calculus process was firstly taken up formally by Abramsky [1], and further elaborated by Bellin and Scott [3]. The authors attach the π-calculus to the Linear Logic inference rules in the style of type theory, thus the process model for a composite service presented by π-calculus can be generated directly from the proof. The authors also present the subtyping rules that are used for semantic reasoning with LL inference figures. Thus the Linear Logic theorem prover can deal with both the service specification and the semantic Web information. Unlike other methods that use non-functional attributes only to filter the generated plan, the authors consider the non-functional attributes directly in the theorem proving process. Both service functionalities and non-functional attributes are translated into propositions in the logical axioms, but the distinguish between the functionalities and non-functional attributes is enabled by the Linear Logic inference rules.

5 Conclusion

This paper has aimed to give an overview of recent progress in automatic Web services composition. At first, we propose a five-step model for Web services composition process. The composition model consists of service presentation, translation, process generation, evaluation and execution. Each step requires different languages, platforms and methods.

In these five steps, we concentrate on the methods of composite Web services process generation. We give the introduction and comparition of selected methods to support this step. The methods are enabled either by workflow research or AI planning. The workflow methods are mostly used in the situation where the request has already defined the process model, but automatic program is required to find the atomic services to fulfill the requirement. The AI planning methods is used when the requester has no process model but has a set of constraints and preferences. Hence the process model can be generated automatically by the program.

Although the different methods provide different level of automation in service composition, we can not say the higher automation the better. Because the Web service environment is highly complex and it is not feasible to generate everything in an automatic way. Usually, the highly automated methods is suitable for generating the implementation skeletons that can be refined into formal specification. A discussion on this topic is presented by Hull et. al. [11].

Further work will include a more thorough analysis of the field in addition to practical testing of and experiments with the methods.

References

1. S. Abramsky. Proofs as processes. *Theoretical Computer Science*, 135(1):5–9, 1994.
2. T. Andrews et al. Business Process Execution Language for Web Services (BPEL4WS) 1.1. Online: http://www-106.ibm.com/developerworks/webservices/library/ws-bpel, May 2003.
3. G. Bellin and P. J. Scott. On the pi-calculus and Linear Logic. *Theoretical Computer Science*, 135(1):11–65, 1994.
4. T. Bellwood et al. Universal Description, Discovery and Integration specification (UDDI) 3.0. Online: http://uddi.org/pubs/uddi-v3.00-published-20020719.htm.
5. B. Benatallah, M. Dumas, M.-C. Fauvet, and F. Rabhi. *Patterns and skeletons for parallel and distributed computing*, chapter Towards Patterns of Web Services Composition, pages 265–296. Springer-Verlag, 2003.
6. D. Box et al. Simple Object Access Protocol (SOAP) 1.1. Online: http://www.w3.org/TR/SOAP/, 2001.
7. F. Casati, S. Ilnicki, and L. Jin. Adaptive and dynamic service composition in EFlow. In *Proceedings of 12th International Conference on Advanced Information Systems Engineering(CAiSE)*, Stockholm, Sweden, June 2000. Springer Verlag.
8. F. Casati, M. Sayal, and M.-C. Shan. Developing e-services for composing e-services. In *Proceedings of 13th International Conference on Advanced Information Systems Engineering(CAiSE)*, Interlaken, Switzerland, June 2001. Springer Verlag.
9. R. Chinnici et al. Web Services Description Language (WSDL) 1.2. Online: http://www.w3.org/TR/wsdl/.
10. B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description Logic programs: Combining logic programs with Description Logic. In *Proceedings of the 12th International Conference on the World Wide Web (WWW 2003*, Budapest, Hungary, 2003.
11. R. Hull, M. Benedikt, V. Christophides, and J. Su. E-service: A look behind the curtain. In *Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, San Diego, USA, June 2003.
12. S. Lämmermann. *Runtime Service Composition via Logic-Based Program Synthesis*. PhD thesis, Department of Microelectronics and Information Technology, Royal Institute of Technology, June 2002.
13. Z. Manna and R. J. Waldinger. A deductive approach to program synthesis. *ACM Transactions on Programming Languages and Systems*, 2(1):90–121, 1980.
14. D. Martin et al. DAML-S(and OWL-S) 0.9 draft release. Online: http://www.daml.org/services/daml-s/0.9/, May 2003.

15. D. McDermott. Estimated-regression planning for interactions with Web services. In *Proceedings of the 6th International Conference on AI Planning and Scheduling*, Toulouse, France, 2002. AAAI Press.
16. S. McIlraith and T. C. Son. Adapting Golog for composition of Semantic Web services. In *Proceedings of the 8th International Conference on Knowledge Representation and Reasoning(KR2002)*, Toulouse, France, April 2002.
17. S. McIlraith, T. C. Son, and H. Zeng. Semantic Web services. *IEEE Intelligent Systems*, 16(2):46–53, March/April 2001.
18. B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid. Composing Web services on the Semantic Web. *The VLDB Journal*, 12(4), November 2003.
19. S. Narayanan and S. McIlraith. Simulation, verification and automated composition of Web service. In *Proceedings of the 11th International World Wide Web Conference*, Honolulu, Hawaii, USA, May 2002. ACM. presentation available at http://www2002.org/presentations/narayanan.pdf.
20. S. R. Ponnekanti and A. Fox. SWORD: A developer toolkit for Web service composition. In *Proceedings of the 11th World Wide Web Conference*, Honolulu, HI, USA, 2002.
21. J. Rao, P. Küngas, and M. Matskin. Application of Linear Logic to Web service composition. In *Proceedings of the 1st International Conference on Web Services*, Las Vegas, USA, June 2003.
22. J. Rao, P. Küngas, and M. Matskin. Logic-based Web services composition: from service description to process model. In *Proceedings of the 2004 International Conference on Web Services*, San Diego, USA, July 2004. IEEE.
23. H. Schuster, D. Georgakopoulos, A. Cichocki, and D. Baker. Modeling and composing service-based and reference process-based multi-enterprise processes. In *Proceeding of 12th International Conference on Advanced Information Systems Engineering (CAiSE)*, Stockholm, Sweden, June 2000. Springer Verlag.
24. E. Sirin, J. Hendler, and B. Parsia. Semi-automatic composition of Web services using semantic descriptions. In *Proceedings of Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS2003*, 2002.
25. R. Waldinger. Web agents cooperating deductively. In *Proceedings of FAABS 2000, Greenbelt, MD, USA, April 5–7, 2000*, volume 1871 of *Lecture Notes in Computer Science*, pages 250–262. Springer-Verlag, 2001.
26. D. Wu, E. Sirin, J. Hendler, D. Nau, and B. Parsia. Automatic Web services composition using SHOP2. In *Workshop on Planning for Web Services*, Trento, Italy, June 2003.

Enhancing Web Services Description and Discovery to Facilitate Composition

Preeda Rajasekaran, John Miller, Kunal Verma, and Amit Sheth

LSDIS Lab, Computer Science Department, University of Georgia, Athens, 30602
{preeda,jam,verma,amit}@cs.uga.edu

Abstract. Web services are in the midst of making the transition from being a promising technology to being widely used in the industry. However, most efforts to use Web services have been manual, thus slowing down the ever changing and dynamic businesses of today. In this paper, we contend that more expressive descriptions of Web services will lead to greater automation and thus provide more agility to businesses. We present the METEOR-S front-end tools for source code annotation and semantic Web service description generation. We also present WSDL-S, a language created for incorporating semantic descriptions in the industry wide accepted WSDL, by extending WSDL 2.0.

1 Introduction

Adoption of Service Oriented Architecture (SOA) is expected to allow enterprises to contract-out their non-critical functions. In the new world economy business processes typically transcend departmental as well as organizational boundaries. Web services are expected to provide the ideal platform to automate these processes as they allow integration of disparate platforms and systems. As these processes become more complex, languages like BPEL4WS [1] are required to represent them and control their execution. Current technology requires hard-coding of the processes, as a result it is difficult to incorporate the latest and better solutions available during runtime. The reason for not being able to accommodate new solutions dynamically is the difficulty in automatically discovering and integrating new services for the processes. To allow automatic and dynamic composition of business processes, faster and more effective methods for representing services and suitable means to automatically identify them are needed.

Though companies are eager for seamless integration solutions, they lack standards to expose expressive representations of their service. This incurs disadvantages in terms of failure of being identified by potential clients, unexpected exceptions during execution and other misinterpretations about the functionality of the service. In this paper, we suggest means of overcoming this by providing richer descriptions about the services being offered. To facilitate understanding by any third party, these descriptions are expressed as a standardized conceptualization of the application domain (ontology). This is the core concept behind Semantic Web Services (SWS). This paper discusses the types of semantic content required to describe the functional aspects of a service, means of incorporating such information into service description and advantages in integration provided by this method in a dynamic environment.

At the lower levels, Semantic Web Services utilize regular Web service technologies such as SOAP – Simple Object Access Protocol (for messaging) and WSDL - Web Services Description Language [2] (for service description). At the higher level, semantic and more expressive descriptions are used to describe the services. In this paper, we propose mechanisms for augmenting WSDL to provide semantic descriptions and enhancing UDDI-Universal Description Discovery and Integration [3] to provide semantic discovery. Fig 1 illustrates the SOA architecture adapted to suit the needs of Semantic Web Services (SWS), which includes Annotated WSDL files, an Enhanced-UDDI registry and the corresponding API's in the Service Registry and Provider.

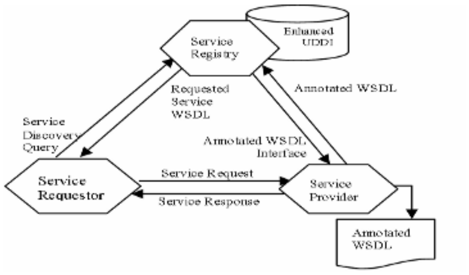


Fig. 1. SOA Architecture

Service requestors depending on business needs can discover Web services published in UDDI Registries. The currently implemented version of UDDI (UDDIv2) provides search capabilities based on keyword and taxonomy. The search results are based on match between keywords present in the description of the published services and the search string. Pure keyword based search fails to retrieve services which are described using synonyms of the search string. Moreover, singular/plural word forms used in the service description also affect the search result. Employing wild characters (e.g. '%') for search helps to increase the recall rate, but necessitates human judgment to filter out relevant services.

The recall and precision of keyword-based search is unsuitable for automation and dynamic composition. The main reason being dynamic composition and automation involves discovering new services at run time by software components without human interaction. In keyword-based search, when the search results are unsatisfactory, the user needs to redefine the keyword (to narrow down the search) to more precisely define the requirement. This requires manual filtering of returned services, to choose the service, which is in the same context as the service requestors request. To enable automation of this process we require 1) meaningful description of the service and its parameters that can be processed automatically by tools and 2) means to process the context of description by discovery engines. This paper discusses the METEOR-S[4] (METEOR-Managing End-To-End OpeRations: for web Services) discovery engine, an improvement over MWSDI [5]. The discovery engine is provided with features to incorporate search based on syntax (keyword matching) or semantics (meaning) or both.

Consider the following scenario in the use-case – 'Dynamic QoS based Supply Chain' [6], where a service requestor is searching for a service to 'return a Quote for a

Hard Drive' using the keyword 'getQuote'. A syntax-based search would return all services with the word 'getQuote' in their description/inputs/outputs/operation name. 'getQuote' is a generic term and a similar service can be offered by many providers such as Electronics Dealers, Hardware Manufactures and Whole-Sale Dealers for their respective businesses. As the context in which 'getQuote' is searched for is absent in syntax based search, we lose precision in our search, and the required service might be lost amidst large number of returned results. Moreover, in keyword search if the users employ very specific terms, e.g., 'getQuoteForComputerHardDrive', the search results returned can be empty, as different service providers may follow different naming conventions for their services. Naming conventions are specific to organizations and developers and hence cannot be generalized.

While employing semantic search, the requestor is not required to guess the name of the service being offered, but is required to provide the context in which the service is used. The search query for 'getQuote' is annotated with the concept 'ComputerParts:#getHardDriveQuote'. This helps to identify those services offering the required functionality, though they follow different naming conventions. For example,'getHardDriveQuoteInformation' is our required service advertised in UDDI, for obvious reasons we can see why the above syntax-based search will fail. If this service is annotated with the concept 'ComputerParts:#getSCSIDriveQuote' or similar concept, by employing reasoning methods (subsumption-relations) we can identify this service as a potential candidate. The reason being 'ComputerParts:#getHardDriveQuote' is the direct parent of 'ComputerParts:#getSCSIDriveQuote' in the domain ontology and hence is closely related to the service being searched. Making use of semantics of inputs and outputs of operation can further refine the search results. This paper elaborates on the use of such semantic information to enhance discovery of services for composition.

Currently, companies are starting to make use of e-business process definition standards such as RosettaNet [7] and ebXML [8] to achieve inter-operability. They are used to provide standardized representation of service functionalities and message exchange formats. Although such standards provide concrete e-business transaction format, they lack the logical reasoning inherent in ontological representations. To overcome this issue METEOR-S employs the use of ontologies based on standards like RosettaNet. The Web Ontology Language (OWL) [9] is used to represent the ontologies. This approach helps Semantic Web services to incorporate the advantages extended by e-business standards into its framework.

While the industry focuses on inter-operability issues by means of existing e-business standards, academic research on the other hand, has turned its focus towards developing approaches tailored for better service representation and reasoning. Identifying potential in the research of Semantic Web Services, two committees have been formed in 2003 to streamline the research ideas in this field. OWL-S [10], WSMO [11] and METEOR-S are active research initiatives in this direction. While the former two develop their own solutions to this problem. METEOR-S, developed at the LSDIS lab of The University of Georgia aims to resolve this by reinforcing current industry standards with the power of semantics.

The paper is organized as follows: Section 2 gives an overview about the METEOR-S architecture. It discusses the various modules that make up the METEOR-S system. The Semantic Web Service Designer module and the output generated by it (annotated source code) are discussed in Section 3. The focus of the next section is on the Semantic Description Generator and WSDL-S - a enhancement of WSDL 2.0 [12]. Sections 5 and 6, elaborate on the Publishing and Discovery modules of the METEOR-S framework. Implementation of the front-end of METEOR-S is presented in Section 7. Research related to the work presented in this paper is discussed in Section 8. Section 9 concludes by giving an overview of the contributions of the paper and future work that can be employed in this direction of research.

2 METEOR-S

The METEOR project at the LSDIS Lab, University of Georgia, focused on workflow management techniques for transactional workflows [13]. Its follow on project, which incorporates workflow management for semantic Web services is called METEOR-S. A key feature in this project is the usage of semantics for the complete lifecycle of semantic Web processes, which represent complex interactions between Semantic Web Services.

The main stages of creating semantic Web processes have been identified as development, annotation, discovery, composition and orchestration. A key research direction of METEOR-S has been exploring different kinds of semantics, which are present in these stages. We have identified data, functional, Quality of Service and execution semantics as different kinds of semantics and are working on formalizing their definitions. A detailed explanation of the underlying conceptual foundation of METEOR-S is present in [14].

From an architectural point of view, we divide METEOR-S in two main parts – the front end and the back end. The front end, which is the focus of this paper, covers the development, annotation and discovery stages. The main components of the front-end are the 1) Semantic Web Service Designer, 2) Semantic Description Generator, 3) Publishing Interface and 4) Discovery Engine. The back end of METEOR-S which covers composition is discussed in [15]

3 Semantic Web Service Designer

The Semantic Web Service Designer of METEOR-S is a GUI to design and develop Semantic Web Services. Using this tool, interface design of services and incorporation of semantic description into the service can be developed simultaneously. This is achieved by means of source code annotations discussed in detail in the next section. This user interface is being developed as an Eclipse plug-in. It provides the user with a tree representation of the interface and an ontology browser, the source of semantic information. The user provides associations between service parameters and ontological concepts. An equivalent representation of the associations - annotated source code is the output of this module.

The semantic description present in the interface of the service, provides the details which any implementation of interface should satisfy. Complete description about the semantics of an operation involves semantic description of inputs, outputs, constraints to be satisfied and exceptions thrown by each operation and the functional description of the operation.

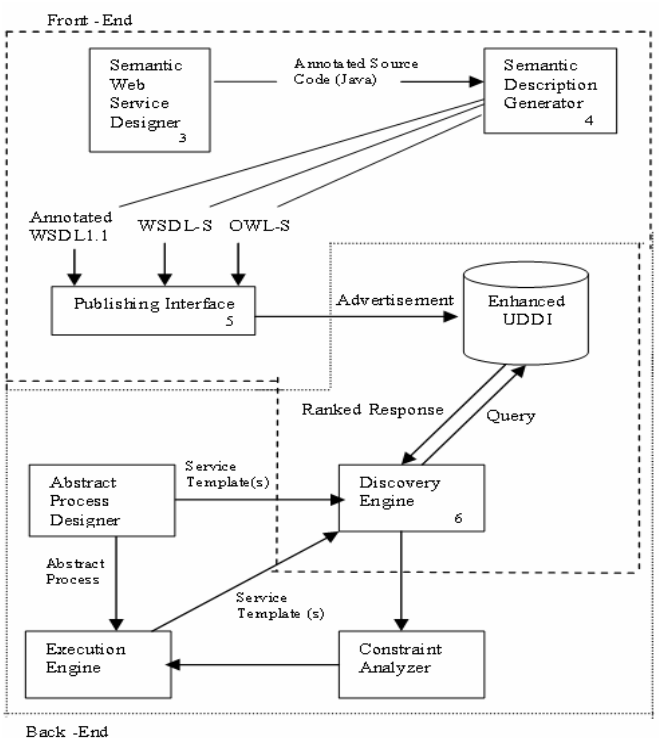


Fig. 2. METEOR-S Architecture

3.1 Source Code Annotation

The output of the 'Semantic Web Service Designer' is the annotated source code. Oracle and C#.NET offers features to add annotations to source code via javadoc comments and inbuilt metatags, respectively. Here we discuss source code annotation with relation to Java, but in general the source code could be any suitable language such as C#.NET. We represent annotations in Java, by employing the meta-tag feature of the new j2sdk, jdk1.5 [16]. These tags have been introduced into the language according to specifications of JSR 175. A Metadata Facility for Java Programming Language [17] and JSR 181-Web Services Metadata for Java Platform [18]

Representation of semantic content in the source code is to provide convenience for developers of Semantic Web Services. The current practices of developing Web services start by processing source code. To adhere to the same standard for developing Semantic Web Services we include annotations at the source code level. A complete example of annotated source code of an interface is presented in Appendix I. The annotation tags and their corresponding semantic significance are discussed next.

@operation Tag - Value of the 'action' attribute provides the functional semantics of the operation
@parameters Tag – It consists of two meta-tags:

@inParam – for input parameters and @outParam –for output parameters. Value of the 'type' attribute is used to refer to the semantic type that closely defines the input/output parameter. The user needs to ensure semantic and data-type match before annotating.
@exceptions Tag – It consists of @exception meta-tags. This is to represent multiple exceptions that be thrown by an operation.
@constraints Tag – It consists of two meta-tags: @pre – for preconditions and @post – for post-conditions. The value of the 'condition' attribute is used to define the constraint the operation has to satisfy before (pre)/after(post) the execution of the operation. Format of the pre and post conditions in the annotated source code is adapted from Design By Contract [19] of JML [20] (Java Modeling Language). It discusses various issues to be considered in the representation of pre and post conditions. The constraints can alternatively be represented using rule languages like SWRL. SWRL 0.6 [21] discusses the built-in features and the syntax of the language. A detailed analysis and processing of rules to utilize the features offered by SWRL is pending.
@interface Tag - The attributes of the tag provide interface specific annotations. These attributes are valid for all implementations of the interface. Attributes such as descriptions can be extended according to provider's need.
@service Tag - The attributes of the tag serve as service specific annotations. A service described by one interface can be implemented by different service providers. This tag is used to represent provider specific parameters such as 'location', 'QoS' (Quality of Service) and 'reliability'.

4 Semantic Description Generator

A basic tenet of Web services is that any service requestor, based on the description in the WSDL files, can invoke them. WSDL provides information about the service such as the operations present, the expected inputs and outputs for an operation. With our requirements for richer description we find this information insufficient for user in METEOR-S. We propose extensions to Web service description in two ways, 1) Annotated WSDL 1.1 and 2) WSDL-S files. Both these files can be generated from the annotated source code by the 'Semantic Description Generator module'.

Annotated WSDL 1.1, is a WSDL 1.1 document with semantic features added to it via permissible extensibility elements present in the language. The semantic extensions are used within the METEOR-S framework, to enhance discovery and composition. At the same time, as the generated Annotated WSDL 1.1 file adheres to the current industry standard, it can be also used outside the METEOR-S framework by service requestors unaware of semantics. This flexibility demonstrates the lightweight approach of the methodology used.

The features of WSDL-S language and the motivation behind its creation are discussed in the next section. The third type of file generated by this module is the set of OWL-S files associated with the annotated source code. As mentioned earlier OWL-S is another research initiative in the direction of developing Semantic Web Services. We propose to show the completeness of semantic description in our system by generating OWL-S files (profile, grounding and partial process model). OWL-S files provide a more complex representation of the semantic descriptions. By generating the OWL-S files from the annotated source code we present means of modeling business processes using a simpler approach.

OWL-S provides semantic information about a service in four files:

1. Profile (.owl) - Describes the functional (input, output, preconditions and effects) and non-functional aspects of the service.
2. Process (.owl) - Describes the service's operations and the interaction protocol of the service.
3. Grounding (.owl) – Provides mapping from abstract (Process model) to concrete (WSDL) representation.
4. WSDL (.wsdl) file for the service.

These files are required by the DAML-S/OWL-S 'Web Service Composer' [22] to execute DAML-S services. Currently, we have integrated the profile and the grounding with the WSDL descriptions. We are investigating approaches of representing the interaction protocol of services. One such approach involves the use of timed-automata based state machines to represent the interaction of services.

4.1 WSDL-S

As discussed, one of the outputs of the semantic description generator is WSDL-S, which is a semantically enriched WSDL 2.0 document. In this section, we describe the motivation and features of WSDL-S. One of the central purposes of WSDL is to describe interfaces (formerly known as port-types) to Web services. In general, service providers/implementers could use a standard interface, extend a standard interface or develop their own.

Broadly speaking, an interface contains a set of operations. Each operation has a signature, which includes an operation name, input, output and exception messages. These messages have types that are defined using some XML-based schema language. The schema language that is commonly used is XSD (XML Schema Definition) [23], although OWL is an alternative. In WSDL 2.0, types are pushed more completely outside the standard, since types systems are complex to define and there exist at least two well-accepted type systems in the XML world: XSD and OWL.

A client of a Web service will look to the interface to find out what it will do. This enables, interface descriptions to help discover candidate Web services. Such descriptions are therefore critical to proper discovery and use of Web services. This makes adding semantics to interfaces an important task.

In WSDL 2.0, OWL and UML/XMI are possible type systems, along with XSD. In WSDL-S, the inputs and outputs are expressed using OWL types from the Rosetta Net Ontology instead on XSD [24]. Round-tripping allows mapping form one type-system to another and is important for maintaining data integrity when the type systems used by the providers and requestors are different. Transformation between (language) Java primitive types and XML-Schema can be achieved by employing some relaxations on the primitives used. A similar mapping between XML Schema and OWL, OWL and Java is not a simple issue. Due to the richness of OWL, we may have to employ complex transformations and work-around to switch between these different type systems. A complete mapping between these different type-systems is an open research issue in this area.

By employing basic transformation rules WSDL-S can be employed in Web service composition, where the individual Web services are used in larger Web processes. With the new WSDL 2.0, WSDL creators are provided features to use an external type system in their document. This raises many research questions with relation to type system round-tripping. The most commonly used type systems are OWL and XML Schema, whereas Web services are developed using languages like C# .NET and Java. Complex and user defined data-types require the service provider to provide the appropriate transformations/mapping to XSD types. A discussion of mapping OWL to Java data types is presented in [25].

While annotating, the developer of the service must provide 'type' information. The 'type' should match the data-structure and semantic-meaning of the concept it is used to annotate. If the user is unable to find a suitable type, then the developer can define their own types as extensions to the existing types. This makes it necessary to provide transformation rules to map between user defined types and standardized/recognized types. Simple transformations such as rupees to dollars may be specified in SWRL.(e.g., Dollar = Rupee * 'http://www.xmethods.net/sd/2001/ CurrencyExchangeService.wsdl getRate USA India')

The parameter 'http://www.xmethods.net/sd/2001/CurrencyExchangeService.wsdl getRate USA India'- is used to represent 1) The Web service with operation 'getRate' to return the exchange rate required for the transformation and 2) Operation input parameters (USA and India). More complex transformations may be specified using XSLT (Extensible Stylesheet Language) transformation. The developer is provided with the following choices to define the type, 1) Use a type from a recognized ontology, 2) Extend such a type and provide at least a downcast operation, or 3) Create their own type and provide mappings to standardized/recognized types. Without adhering to these transformation rules, interoperation between partners will be error-prone.

5 Publishing Interface

Once the semantic descriptions are generated they need to be advertised, so that they are readily accessible by service requestors. UDDI Registries offer support for publishing service descriptions. However, the current version of UDDI (UDDIv2) offers little support for exposing semantic information [26]. This has motivated the development of Enhanced-UDDI, essentially a layer above UDDI, which is capable of handling semantic data. The upcoming UDDIv3 provides better support to organize the semantic information.

Enhanced-UDDI is organized so as to decrease search time and increase the precision of operations like service discovery. The internal organization of UDDI data-structures are modified to act as place holders of semantic information [27]. The data structures of UDDIv2 are discussed in detail in [28]. Category Bags in UDDI are a list of name-value elements, in our implementation we have used the 'value' attribute to be the place holder of semantic content. In METEOR-S binding templates holds Location and Domain specific T-models. This enables direct search of services that function in a particular Geographic Location and Domain.

The category bag associated with the Business Service, serves as a placeholder for the operation /inputs /outputs /exceptions /constraints oriented semantics. Service specific semantic information is stored in the Binding Template, which falls under Business Service. This abstraction of data helps to organize the information for effective retrieval during discovery. An advertisement built from the annotated source code semantic descriptions serves as the input to the Publishing interface. The discovery Engine employs a query similar to the advertisement to find the information from the Enhanced-UDDI[1].

6 Discovery Engine

As shown in Fig 2, both the front-end and back-end require the use of the Discovery Engine module. Currently, discovery in UDDIv2 supports keyword and taxonomical based search. As mentioned earlier this is insufficient in a dynamic/automated environment. In METEOR-S, the discovery method is based primarily on the semantic descriptions and constraints advertised by the service provider. While supporting the current keyword-match on Web services description, the Discovery Engine improves upon this by employing heuristics based on subsumption-relations, data-type matching between requestor specified constraints and provider-advertised concepts, common ancestor, properties and subclass match between concepts. Inferencing can be employed on the constraints published by the service provider to filter the results of discovery. This reasoning helps to deal with the constantly changing needs of a dynamic environment.

A query template is used to construct the query that specifies the functional aspects of the required service. The query template consists of specifics about a service such as operation name, operation action (functional semantics), input/output name and (semantic) type, exception, pre/post conditions, domain, location. Such a query may be generated by automated tools or built manually by users. The Discovery Engine processes the query to discover the appropriate services. The user-query provides the Discovery Engine with the specifications of the user, further annotated with semantic type information.

The effectiveness of the METEOR-S Discovery Engine is greatly attributed to the organization of Enhanced-UDDI. The Discovery Engine uses the classes subsumption-relation to compare the ontological concepts specified in the query to those advertised in the registry. It also engages the use of metrics [30] obtained by comparing the properties of the concepts, matching the cardinality and the data type, distance from the common parent, etc., in ranking the relevant services discovered.

Discovery results returned by the user/tool are ranked according to the degree of match. Other specifics about a service such as reliability, Quality of Service, etc. can also be used in deciding the final rank of services returned. Constraints on operation play an important role in ranking the services. These service parameters descriptions and constraint analysis are used extensively in composition of business flows. The use of the METEOR-S Discovery Engine in composition is discussed in detail in [15].

[1] An alternative to using a UDDI like registry is to use a service ontology, based on logic. In this way, logical subsumption can be employed to find appropriate matches during discovery [29].

7 Implementation of the System

An overview of the implementation details of the above-discussed modules is presented in this section. The Semantic Web Service Designer (SWS Designer) provides the interface required to create associations between the various service parameters and ontological concepts. The Semantic Web service designer represents the service interface in the form of a tree. The input and output parameter nodes are organized under the corresponding operation nodes. An ontology browser is provided to the user helping them navigate through an ontology and choose the appropriate semantic concept. Once the basic annotations are generated, the user can view the annotated source code via a Java editor. Direct editing of the source code is optional if the user is familiar with the format of the annotations. The color scheme of the Java editor is changed to highlight the annotations embedded in the source code. A syntax checker for the annotations is employed before the user can save the annotated source code.

The main modules of the Semantic Description Generator are the Document Generator, Type Converter and Validator. The semantic description generator takes as input the annotated source code. The annotations are extracted from the source code, by means of the Annotations API that is incorporated into Java reflection in jdk1.5. Depending on the users preference Annotated WSDL1.1 or WSDL-S or OWL-S can be generated. A table driven document generation approach is adopted for implementation. The tags associated with WSDL are stored in a table, which are used during document generation. This helps in code maintenance and for accommodating possible changes in tag names.

For Semantic Web services to be successfully invoked, we need system-supported mappings between the different type systems. The main reason being, service descriptions are available to requestors via WSDL documents and WSDL offers support to varied type systems. After the service requestor discovers the appropriate WSDL file, mapping WSDL/XSD data-types to appropriate Java types is essential for successfully invocation of the service.

Recursive definition of complex data-types is provided in the ‘types’ tag of the WSDL documents. The execution engine is provided with the same hashtable to perform inverse look-up during service execution. Correctness of the generated WSDL documents is checked with validators. WSDL4J [31] API is used to check the validity of the generated WSDL code.

The Publishing interface can have two different sources, the annotated WSDL file or annotated source file. If an annotated source code is provided, an appropriate WSDL file is generated before the service is actually published in the UDDI. The publisher builds a service advertisement, which contains all the required semantic information of the service. The publisher is equipped to handle (publish) annotated as well as un-annotated representations of the service.

The discovery engine provides two interfaces for interaction. One User Interface suited for humans to build the query template and to view the results and an API to be used by composition tools. The functionality extended by both the interfaces is the same, but the representation of the former is to suit human interaction. Different criteria for discovery, the relaxation constraints and ranking schemes can be customized according to the user/tools employing the discovery engine. The discovery engine when called by a tool such as Execution Engine is customized to perform more stringent matching. This is because automation requires near prefect service match for seamless execution and the absence of human intervention.

The backend of the METEOR-S framework is dedicated to using the features provided by the front-end for composition and execution of business processes. The Abstract Process model helps to capture semantic descriptions of the services with the help of ontologies. Users can also specify local constraints for each service and global constraints for the complete process. These constraints are based on generic QoS criteria [32] such as cost, availability and reliability as well as any domain specific QoS criteria that may be relevant. After process specification, Enhanced UDDI is used to get candidate services for all the service templates in the process. [33] gives an overview of how the semantic descriptions can help in resolving inter-service dependencies based on domain constraints captured in ontologies. The modules of the back-end are explained in [15].

8 Related Work

In this paper, we have presented an approach to attach semantic descriptions to services at design time, through source code annotations. We have also discussed the changes needed to incorporate these descriptions into standards like WSDL and service registries like UDDI, to enhance discovery. This section presents ongoing research related to the work presented in this paper.

[26] and [30] describe the methods to semantically enhance UDDI to support service descriptions. An approach to define the functionality of a Web service as the transformation of inputs to outputs is discussed in [26]. MWSDI [5] presents the use of service templates to discover suitable services during composition of business flow.

Our discussion on the Semantic Description Generator gave an overview of how our work is closely related to OWL-S. OWL-S defines a approach to enable Semantic Web services. We believe that our approach is more lightweight and easier to apply. We have developed tools to generate OWL-S files from WSDL-S file. Our approach tries to adhere to the current standards while trying to maximize semantic representations required for automation. The other research initiative in this area is based on the work done in WSMF (Web Services Modeling Framework) [34]. WSMO (Web Services Modeling Ontology) is developed to encompass the different aspects of Web service Development. It aims to solve the interoperability issue by means of mediators and goals (pre and post conditions) described using F-logic statements. The complexity of F-logic can serve as a disadvantage to users who are unfamiliar with rule languages. Our approach involves representing constraints as boolean expression in annotated source code and converting the same to SWRL rules in WSDL-S documents. The former representation enables the developers to easily understand the constraints, while the later is used for logical querying using inference engines.

9 Conclusion and Future Work

In this paper, we have presented an approach, which allows software developers to incorporate semantic descriptions of Web services during code development. This approach leverages the annotation mechanism provided by the Java programming language. We have verified our ideas by implementing a Semantic Web Service designer for source code annotation and Semantic Description generator for generation of rich descriptions of Web services. In addition, we present the WSDL-S language, which has been created by extending WSDL 2.0. This work has been done as part of the METEOR-S project at the University of Georgia. We have endeavored to add more expressivity to Web service descriptions, while staying close to well accepted industry standards.

Future work in this area will involve deciding the annotations required in the other phases of Web service development like protocol specification, transaction management, security, etc. Capturing the behavioral aspects (process modeling) of Web services is also a part of future work. A validation framework to simulate and validate e composed workflows will be developed as a part of METEOR-S.

References

1. Specification: Business Process Execution Language for Web Services Version 1.1http://www-106.ibm.com/developerworks/library/ws-bpel/
2. Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001-http://www.w3.org/TR/2001/NOTE-wsdl-20010315
3. UDDI Version2 Specifications-http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm #uddiv2
4. METEOR-S:Semantic Web Services and Processes, http://swp.semanticweb.org, 2002.
5. Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S. and Miller, J:, METEOR–S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services, Journal of Information Technology and Management (to appear), (2004).
6. Verma, K., Sheth, A., Miller, J., Aggarwal, R.: Dynamic QoS based Supply Chain, Semantic Web Services Initiative Architecture Committee (SWSA),Use Case, April 2004.
7. RosettaNet – Lingua Franca for e-Business , http://www.rosettanet.org/ RosettaNet/ Rooms/ DisplayPages/LayoutInitial
8. Core Component Dictionary, ebXML Core Components, 10 May 2001, Version 1.04, www.ebxml.org/ specs/ccDICT.pdf
9. OWL Web Ontology Language Overview- http://www.w3.org/TR/2004/REC-owl-features-20040210/
10. The DAML Services Coalition, DAML-S: Web Service Description for the Semantic Web, The First International Semantic Web Conference -ISWC, Italy.
11. Roman, D., Keller, U., Lausen, H.: WSMO – Web Service Modeling Ontology (WSMO), DERI Working Draft 14 February 2004, http://www.wsmo.org/ 2004/d2/v0.1/20040214/
12. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language-http://www.w3.org/TR/2003/WD-wsdl20-20031110/

13. Sheth, A., Kochut, K., Miller, J., Worah, D., Das, S., Lin, C., Palaniswami, D., Lynch, J., Shvchenko, I.: Supporting State-wide Immunization Tracking using Multi-Paradign WorkflowTechnology, Proceedings of the 22nd Intl. Conf. on Very Large Databases (VLDB96) September 1996.
14. Semantic Web Process Lifecycle: Role of Semantics in Annotation, Discovery, Composition and Orchestration, Invited Talk, WWW 2003 Workshop on E-Services and the Semantic Web, Budapest, Hungary, May 20, 2003.
15. Aggarwal, R., Verma, K., Sheth, A., Miller, J., Milnor, W.: Constraint Driven Web Service Composition in METEOR-S (submitted to 2004 IEEE International Conference on Services Computing).
16. jdk 1.5 Java Development Kit- http://java.sun.com/j2se/1.5.0/index.jsp
17. JSR 175 Java Specification Requests - http://www.jcp.org/en/jsr/detail?id=175
18. JSR 181 Java Specification Requests, http://www.jcp.org/en/jsr/detail?id=181.
19. Jézéquel, J. and Meyer, B. Design by Contract: The Lessons of Ariane. IEEE Computer, 30(1), 129-130.
20. Design by Contract with JML, 2004.
21. SWRL: A Semantic Web Rule Language Combining OWL and RuleML, Draft Version 0.6 of 23 March 2004 http://www.daml.org/rules/proposal
22. Sirin, E., Hendler, J., Parsia, B.: Semi-automatic Composition of Web Services using Semantic Descriptions, Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS2003, April 2003 (pp –17-24).
23. XML Schema Part 0: Primer http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/
24. Rajasekaran, P., Miller, J., Verma, K., Azami, M., Sheth, A.: Cost-Benefit Analysis of Adding Semantics to Web Service Description (in preparation).
25. Kalyanpur, A., Pastor, D., Battle, S., Padget, J.: Automatic mapping of OWL ontologies into Java - http://www.mindswap.org/aditkal/www2004_ OWL2Java. pdf.
26. Paolucci, M., Kawamura, T., Payne, T., Sycara, K.: Semantic Matching of Web Services Capabilities. Proceedings of the ISWC 2002, First International Semantic Web Conference, Sardinia, Italy, June 2002. Springer
27. Paolucci, M. and Kawamura, T. and Payne, T., Sycara, K.: Importing the Semantic Web in UDDI. Proceedings of Web Services, E-Business and Semantic Web Workshop, CAiSE 2002 (pp 225-236).
28. UDDI Data structure reference-http://www.hpmiddleware.com /downloads/pdf/Web_services_datastructure_v1.pdf
29. Sycara, K., Paolucci, M., Ankolekar, A., Srinivasan: Automated discovery, interaction and composition of Semantic Web services, Web Semantics: Science, Services and Agents on the World Web, Dec 2003, (vol: 1,no. 1, pp. 27-46).
30. Akkiraju, R., Goodwin, R., Doshi, P., Roeder, S.: A Method For Semantically Enhancing the Service Discovery Capabilities of UDDI, In Proceedings of the Workshop on Information Integration on the Web, IJCAI 2003, Mexico, Aug 9-10, 2003.
31. WSDL4J Project,http://www-124.ibm.com/developerworks/projects/wsdl4j/
32. Cardoso, J., Sheth, A., Miller, J., Arnold, J., and Kochut, K.: Quality of Service for Workflows and Web Service Processes, Journal of Web Semantics April 2004, (vol. 1,no.3, pp 281-308).
33. Verma, K., Akkiraju, R., Goodwin, R., Doshi, P., Lee, J.: On Accommodating Inter Service Dependencies in Web Process Flow Composition, AAAI Spring Symposium on Semantic Web Services, (pp 37-43).
34. Web Services Modeling Framework Electronic Commerce: Research and Applications, (2002) 113-137-http://www.wsmo.org/papers/publications/wsmf.paper.pdf

APPENDIX I - Annotated Source Code (Java)

```
     import java.lang.annotation.*;
     import java.lang.reflect.*;

     @namespaces ({ @namespace ( name = "rosetta", service_extension = true,
                        url = " http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-s/pips.owl " ) })
     @interface ( domain = "naics:Computer and Electronic Product Manufacturing" ,
                   description = "Computer PowerSupply Battery Buy Quote Order Status ",
                   businessService ="Computer   Parts Supplier")

public interface BatterySupplier {
     @operation ( name = "getQuote", action = "rosetta:#RequestQuote" )
          @parameters ({
               @inParam ( name = "qRequest", element = "rosetta: #QuoteRequest" ),
               @outParam ( name = "quote",  element =  "rosetta: #QuoteConfirmation" )
          })
     QuoteConfirmation getQuote (QuoteRequest qRequest );
     @operation  ( name = "placeOrder", action  = "rosetta: #RequestPurchaseOrder" )
          @parameters ({
             @inParam ( name = "order", element = "rosetta: #PurchaseOrderRequest" ),
             @outParam ( name = "orderConfirmation", element ="rosetta: #PurchaseOrderConfirmation" )
            })
         @exceptions ({
               @exception ( element = "rosetta:#DiscountinuedItemException" )
          })
         @constraint({
             @pre( condition = "order.PurchaseOrder.PurchaseOrderLineItem.RequestedQuantity  > 7" )
        })
     }//end of class
```

Compensation in the World of Web Services Composition

Debmalya Biswas

Dept. of Computer Science, Memorial University of Newfoundland,
St. John's, NL, Canada A1B 3X5
debmalya@cs.mun.ca

Abstract. Industry and researchers acknowledge Web services as being the next generation of distributed computing. However, several issues especially the reliability aspect needs to be addressed before Web services can deliver its promise. Due to their heterogeneous, autonomous and long-lived nature, traditional ACID (Atomicity, Consistency, Integrity, Durability) based models are not sufficient for providing transactional guarantee to Web services compositions. To overcome this limitation, many extended transaction models have been proposed based on the concept of compensation. In this paper, we stress on the importance of two aspects, the Cost of Compensation and End User Involvement, which are missing from most of the transaction models proposed until now. We also show how industry standards like BPEL4WS, WS-Transaction can be augmented to facilitate the above aspects. Finally, we propose a simple classification towards describing compensating operations.

1 Introduction

Web Services Composition (WS-Composition) relates to combining the capabilities of primitive services so as to deliver a new (more capable) service. Due to their heterogeneous, autonomous and long-lived nature, traditional ACID based models are not sufficient for providing transactional guarantee to WS-Compositions. To overcome this limitation, extended/enhanced transaction models have been proposed which allow component Web services to complete irrespective of the outcome of the composite Web service. In such a scenario, atomicity property is preserved by using *compensating* transactions, which semantically undo the effects of the completed component Web services, in case of transaction abort.

Compensating transactions have been studied extensively in the multi-database environment [1], [2]. However, research related to the application of compensating transactions in the field of Web services has failed to acknowledge the importance of the following two aspects:

1. Cost of Compensation: Real world activities often have a cost associated with them. As such, it becomes essential to select and perform the most *optimum* compensation.
2. End User involvement: Again, most discussions on compensating transactions consider compensation from a service provider point of view. Typically, compensating transactions are not focused in the design of Web service transaction models, and implementation of this functionality is left to the application developer (Web

service developer). Given this scenario, it is not possible to consider the end user's point of view while defining the compensating transactions. For example, a cancellation operation might be a successful compensation for a reservation operation from a service provider point of view; however, it might not be so from the end user's perspective if a cancellation charge is involved. While we do not foresee a radical change in the way Web services compensations are defined, we do stress that there is a need for constant end user involvement. The best developers can do is to give multiple compensation options, but at the end of the day, the end user is the best judge of which option is most suitable to him/her. Also, as we discuss later, there is a need for end user involvement for taking intermediate decisions, suggesting alternatives while implementing complex recovery schemes [3], [4].

Facilitating the aspects discussed above in turn implies maintaining sufficient information about the current state of execution, compensation options, etc. as well as having enough flexibility within the model to accommodate run-time decisions. This is further complicated by the fact that WS-Compositions may be nested up to any depth, with only level i having the required knowledge (current state of execution, alternatives available, etc.) about level i+1 (if we consider level 0 as the topmost level).

In this paper, we stress on the importance of the aspects in real life practical systems as well as give details regarding how existing specifications can be augmented to provide the above discussed features. Please note that the implementation approach proposed here is not independent, rather it builds on the infrastructure provided by industry standards like BPEL4WS [12], WS-Transaction [10]. In a way, the dependence on industry standards proves the practicality/ feasibility of the proposed solution.

While considering the implementation details, we felt the need to be able to describe the compensating operations semantically. For the end user to make an informed decision, it is not sufficient to describe an operation as simply compensable/non-compensable. Here, we provide a simple categorization for compensating operations. Please note that this is a very primitive attempt at trying to capture the semantic effects of compensating operations and needs further refinement.

The rest of the paper is organized as follows. In section 2, we provide a brief introduction to compensation in a Web services context. We elaborate on the Cost of Compensation and End User Involvement aspects in section 3. Section 4 discusses semantic representation of compensating operations. Section 5 explains how the infrastructure provided by some of the standards like BPEL4WS, WS-Transaction can be augmented to facilitate cost based compensation selection and end user involvement. Section 6 concludes the paper and provides directions for future work.

2 Compensation and Web Services

An excellent survey of the extended transaction models which have been proposed until now for Web services is provided in [8]. Here, we suffice to provide an overview of the transaction models which consider compensation in some detail. [9] describes how compensating transactions can be modeled based on the active database concept of triggers, basically as Event-Condition-Action (ECA) rules. [4] presents a forward recovery based transaction model. It introduces the concept of co-operative recovery (in the context of Web services). They also acknowledge the importance of end user involvement in the recovery process. However, in their scenario, the end user is involved only as the last resort. For example, when it is not possible to get flight or hotel reservation, then the end user is consulted to suggest another travel date. They do not consider the use of end user input for taking intermediate decisions, selecting among possible compensation options etc. In [5], Pires et. al. propose a framework (WebTransact) for building reliable Web services compositions. Their framework, based on the concept of forward recovery, allows for specification of composition properties like atomicity and guaranteed termination. According to [5], "the WebTransact framework defines four types of transaction behaviors of remote services, which are: compensable, virtual-compensable, retriable, or pivot. A remote operation is compensable if, after its execution, its effects can be undone by the execution of another remote operation. The virtual-compensable remote operation represents all remote operations whose underlying system supports the standard 2PC protocol. These services are treated like compensable services, but, actually, their effects are not compensated by the execution of another service, instead, they wait in the prepare-to commit state until the composition reaches a state in which it is safe to commit the remote operation. A remote operation is retriable, if it is guaranteed that it will succeed after a finite set of repeated executions. A remote operation is pivot, if it is neither retriable nor compensable". [3] presents a conceptual, multi-level service composition model, which extends the above model [5] to allow specification of atomicity and guaranteed termination properties at different levels of abstraction. [8] uses something called alternative participant approach for failure recovery. Basically, the system maintains a ranked list of Web services providing similar functionality. In case of failure, the system abandons the failed participant and invokes another service providing similar functionality from its backup list (to complete the work left unfinished by the failed participant). As such, there is no question of any roll back or compensation.

On the standards front, both WS-Transaction (WS-T) and the more recent WS Transaction Management (WS-TXM), part of the WS Composite Application Framework (WS-CAF) [11], provide support for compensation based long running activities (called Business Activities in the WS-T context and Long Running Actions in WS-TXM terminology). The WS-TXM framework also proposes a Business Process (BP) transaction protocol which uses a set of interposed coordinators to provide transactional guarantee across business domains. Basically, the overall business activity is split into domain specific tasks where each domain might be using a different transaction model. No WS-Composition standards discussion would be complete without mentioning the Web Ontology Language for Services (OWL-S) specification [14]. OWL-S currently does not define a recovery protocol. Since transaction frameworks like WS-T and WS-TXM do not make any assumptions regarding the underlying Web services orchestration and choreography technology, they can be used to complement OWL-S to provide the reliability aspect.

In the next section, we discuss the Cost of Compensation and End User Involvement aspects in more detail.

3 Cost of Compensation and End User Involvement

3.1 Cost of Compensation:

As mentioned earlier, real-life operations often have a cost associated with them. If we try to draw an analogy within current standards, a policy (WS-Policy) document describing the terms and conditions associated with a service is probably the closest. However, even if we assume the existence of a very descriptive policy document, there are two aspects which current specifications don't consider:

1. End user involvement: Basically, someone has to select the most optimum compensation, and who better can do this than the end user himself. Please note that people often equate compensation selection with fault (exception) handler selection. However, there is a subtle difference between the two. Fault handler selection depends on the type of fault that has occurred and as such can be done by the system. Compensation selection, on the other hand, is much more complex. During compensation, we are trying to semantically undo the effects of an operation which has earlier completed successfully. There are many variable factors like extent to which the results can be undone, associated cost, etc. to be considered which make compensation selection quite complex.
2. Multi-level compensation: If we consider a nested composite Web service, compensation may be possible at different levels with different costs.

For example, let us consider the classical travel booking scenario (Fig. 1). If a hotel or flight booking needs to be compensated, then it can be achieved by either invoking the compensating (cancellation) operations at the hotel or flight booking sites respectively or by invoking the compensating operation (Cancel Travel) at the composite travel booking site. Now, if we assume that the user is a premier member of the composite travel booking site and as such gets a 15% discount on all cancellation charges, then it is beneficial (from the user's point of view) to invoke the Cancel Travel operation at the composite travel booking site.

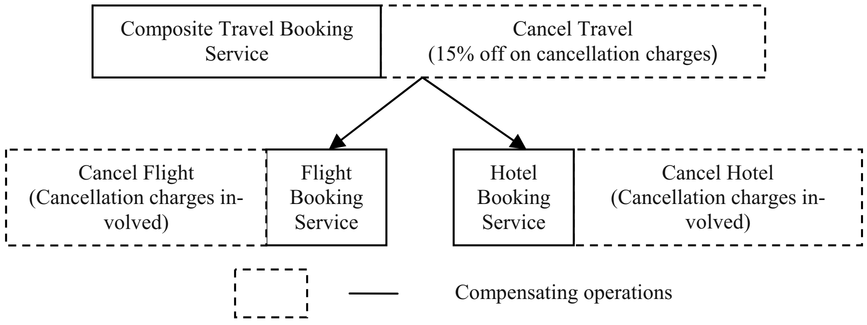

Composite Travel Booking Service
Cancel Travel (15% off on cancellation charges)
Cancel Flight (Cancellation charges involved)
Flight Booking Service
Hotel Booking Service
Cancel Hotel (Cancellation charges involved)
Compensating operations

Fig. 1. Travel booking scenario

On the same lines, let us consider an extended travel booking scenario (Fig. 2). Basically, we add another level of composition, the Composite Travel & Shipping Service which is composed of a Composite Travel Booking Service and a Shipping

Service. Now, let us assume that the shipping service booking fails (due to some reason) and the travel booking at the Composite Travel Booking site needs to be compensated (cancelled). However, most travel agents consider cancellation as a separate activity (similar to booking), and would charge their commission in addition to the cancellation charges applicable. Given this scenario, it makes sense (again from the user's point if view) to cancel (compensate) at the hotel (Cancel Hotel) and flight (Cancel Flight) booking sites directly rather than invoke the Cancel Travel compensating operation.

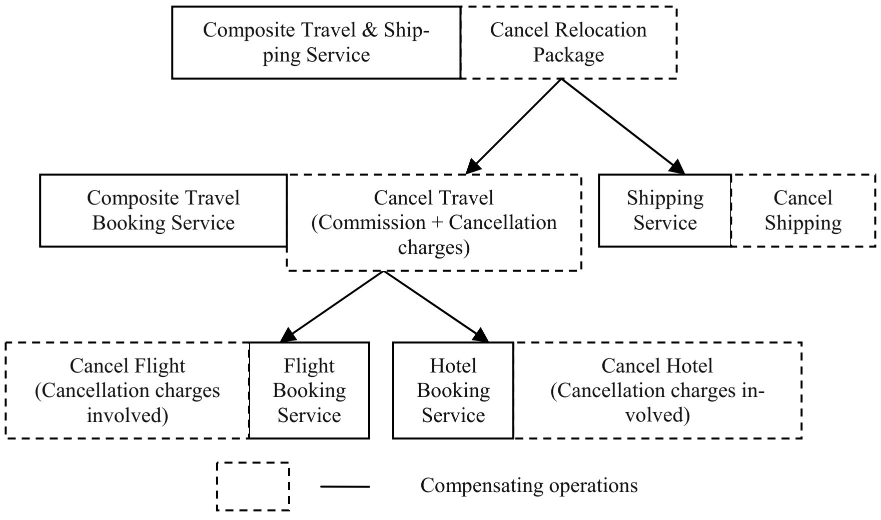

Composite Travel & Shipping Service
Cancel Relocation Package
Composite Travel Booking Service
Cancel Travel (Commission + Cancellation charges)
Shipping Service
Cancel Shipping
Cancel Flight (Cancellation charges involved)
Flight Booking Service
Hotel Booking Service
Cancel Hotel (Cancellation charges involved)
Compensating operations

Fig. 2. Extended travel booking scenario

The above examples illustrate that often the compensation option, at the level at which compensation is required, may not be the most optimum one. However, the concept of multi-level compensation as explained above is not feasible using current specifications. In a conventional scenario, the Web services execution environment (middleware) at each site is responsible for performing the compensation at that site. Even if we assume that the middleware at each site is intelligent enough to select the most appropriate compensation, there is no middleware which has the *overall* picture of the compensations possible at different levels (sites). The "throw" mechanism followed by compensation/fault handlers in case of BPEL4WS can probably be considered as the closest (among current specifications) to the concept of multi-level compensation as discussed above. Although, the "throw" concept allows multi-level propagation of compensations/faults, there are certain differences:

1. If a scope does not have a compensation handler or the compensation fails, the "throw" mechanism simply transfers the responsibility for compensation to the enclosing scope. As evident, it is still based on the concept of trying to perform the first possible compensation rather than the most optimum one.
2. Compensations possible at lower levels are not considered.

3.2 End User Involvement

As discussed above, end user involvement is required for selecting the most optimum compensation (assuming more than one compensation options are available).

At the other extreme, we have non compensable operations. In such a case, the end user might help by suggesting alternate acceptable outcomes. For example, if the flight reservation cannot be compensated (non refundable tickets), the user might change his initial requirement of both flight and hotel reservation to just flight tickets saying he can get the hotel reservation from some other travel agency.

Researchers have also proposed forward recovery schemes as a means of overcoming the limitations posed by non-compensable operations [3], [4], [5]. However, forward recovery does not mean that there is no need for compensation. In fact, [3] acknowledges that forward recovery might not be always possible, as such, it might be required to compensate up to some extent and then try forward recovery again. [4] relies on the concept of co-operative recovery. For example, in the case where either the flight or hotel booking fails, the component service raises an exception that is *cooperatively* handled at the next higher level (Composite Travel Booking Service, in our case). While conceptually absolutely feasible, the question we need to ask here is "Are current software systems intelligent enough to take such decisions on their own?" Probably the only systems capable of showing such intelligence are Multi-Agent Systems (MAS) [6]. Even if we assume MAS, it is probably more feasible to have an agent acting on behalf of the end user [7] rather than two agents trying to negotiate a recovery. The reason is that an agent acting on behalf of the end user needs to keep only the user's interests in mind while two agents trying to negotiate would involve a clash between the interests of the respective parties the agents represent (which is definitely much more complex).

Even if we assume MAS or as [4] does, involve the user as a last resort, there is still a need to be able to present the overall picture (state of execution, extent of compensation possible, cost of compensation, etc.) to the end user. That brings us to the need for semantic representation of compensating operations.

4 Semantic Representation of Compensating Operations

A lot of research has gone into how Web services operations can be described semantically. OWL-S is probably the most promising (as well as the one having maximum industrial support) of the specifications proposed until now. However, the same cannot be said for describing compensating operations. In fact, one of the goals of the OWL-S specification is "to have the ability to find out where in the process the request is and whether any unanticipated glitches have appeared". But this part of OWL-S is still a goal and has not yet been defined. The same goes for BPEL4WS, which lists the mechanism for exposing the state of executing processes as a "future direction". As such, it's still early days to predict how much more work would be required to describe not only any unexpected glitches but also the rectifying (compensating) options available.

Below we describe a simple classification for compensating operations as well as give pointers regarding how they can be represented using OWL-S. We can draw relief from the fact that by the time the Web services lifecycle stage, which requires describing the compensating operations, is reached the descriptions of the operations (they compensate) would already be known. As such, the compensating operations can be described relative to the operations they compensate. This is in contrast to the need for semantic description of the Web services operations during the binding stage.

The classification for compensating operations is as follows:

- Fully compensable: In OWL-S terminology, the "effect" of a fully compensating operation would be the exact negation of the "effect" of the operation it compensates.
- Conditionally compensable: By conditionally compensable, we refer to those compensating operations which allow full compensation (both from the end user's and provider's points of view), but have a condition associated with them. For example, often vendors instead of refunding money allow the customer to purchase something equal in worth to the returned item. So, for conditionally compensable operations, the provider needs to specify the following conditions: 1. offer validity period, 2. applicable goods/items for exchange (valid against) and 3. location/companies where the offer is valid (valid at). While the "effect" of a conditionally compensable operation is the same as that of a fully compensable operation, the above conditions (a, b and c) need to be associated with the operation's description. The conditions can be specified either as "preconditions" or as conditions associated with the "effects" using the ConditionalEffect class.
- Partially compensable: Of all the categories, these are probably the most difficult to describe semantically. Studies [13] have shown that most refund policies can be described as a conditional relationship between price, quantity and time. We find the factor customer relationship/history (premium member, credit rating etc.) as being equally important and assume all compensation options as described in terms of the factors: price, quantity, time and customer relationship. The "effect" of a partially compensable operation can be expressed as a percentage of the "effect" of the operation it compensates. For example, if the "effect" of a booking operation is "Ticket booked and $x paid", then the "effect" of a partially compensating operation satisfying "quantity (>5 tickets) & time (>14 days in advance) -> money (refund 90%)" would be "Ticket cancelled (-100%) and $y received (-90%)". The conditions, as before, can be expressed either as "preconditions" or using the ConditionalEffect class.
- Non compensable (similar to pivot operations in [3] and [5]).

It is possible for combinations of the above categories to exist. We provide examples of the practical usage of the above categories in the next section.

5 Implementation Infrastructure

From an implementation point of view, we just need to be concerned about how to

1. present information regarding possible compensation options at all levels to the end-user.
2. permeate user decision to the applicable (lower) level.

BPEL4WS and WS-Transaction provide most of the infrastructure support needed. The BPEL4WS specification provides a feature called Long Running Transactions (LRT) which addresses the problem of the order in which the compensating operations need to be invoked. However, the notion of LRT is purely local and does not provide distributed coordination among multiple-participant services to reach a consensus regarding the outcome. The problem of distributed agreement for a business process spanning multiple vendors and platforms is solved by using WS-Transaction. As discussed earlier, WS-Transaction provides the Business Activity (BA) protocol for long-lived compensation based activities. Fig. 3 gives a simple composite schema while Fig. 4 gives the BA protocol execution for part of the schema. Points to consider regarding Fig. 4 (All steps refer to the step numbers in Fig. 4):

1. The Create CC message allows specifying a relationship between a newly created activity and an already existing activity i.e. establish a parent-child relationship between the activities (Steps 7, 16). This technique of using proxy coordinators is called *interposition*. Basically, the coordinators of D and B (C-D, C-B) would be registered as proxy coordinators with the coordinators of B and A (C-B, C-A) respectively.
2. Each participant registers with a coordinator (Steps 11, 20), specifying the transaction protocols and properties supported by itself, to execute one of the coordination type's protocols. The registration allows *a compensation operation* to be specified which would be invoked on failure.
3. The child's lifecycle does not end even after sending the Completed message to its parent (Step 23). Instead, it waits until it receives a Close or Compensate message from the parent. If it receives the compensate message (Step 25), the child performs the registered compensation.

Now, that we have discussed the relevant points of the WS-Transaction specification, let us consider the *extensions* we need. We complement the extensions discussion with a running example (Fig. 3) detailing what would happen if the activity E (being coordinated by B) fails, and as a result activity D needs to be compensated. We refer to the coordinator, coordinating the failed business activity, as the initiating coordinator (C-B for the example).

1. During proxy coordinator registration, allow the parent coordinator to return selective end user information to the proxy coordinator. Since, OWL-S currently does not define an ontology for specifying the contact information, let us assume that the end user information is represented using the (now deprecated) Actor class (which allows specifying the end user's name, phone, fax, email, physical address and web URL). Depending on the transparency restrictions, the coordinator passes part of the information to the proxy coordinator. Also, during participant activity registration, allow all compensation options along with their description (as given in Fig. 3) to be registered.
2. Whenever a business activity fails and some operations need to be compensated, the initiating coordinator sends a message to all the proxy coordinators registered with it asking for information regarding the compensation options at their level. C-B sends a message to C-D asking for information regarding compensation options.
3. The proxy coordinators in turn send messages to the proxy coordinators registered with them (if any). This continues recursively until the leaf coordinators are reached. Otherwise, they simply return the information regarding the compensation options at their level. Since there are no proxy coordinators registered with C-D, so there is no further propagation of the information seeking message. C-D sends information regarding the compensation options at F and G to C-B. C-B, obviously is aware of the compensation options at D itself.

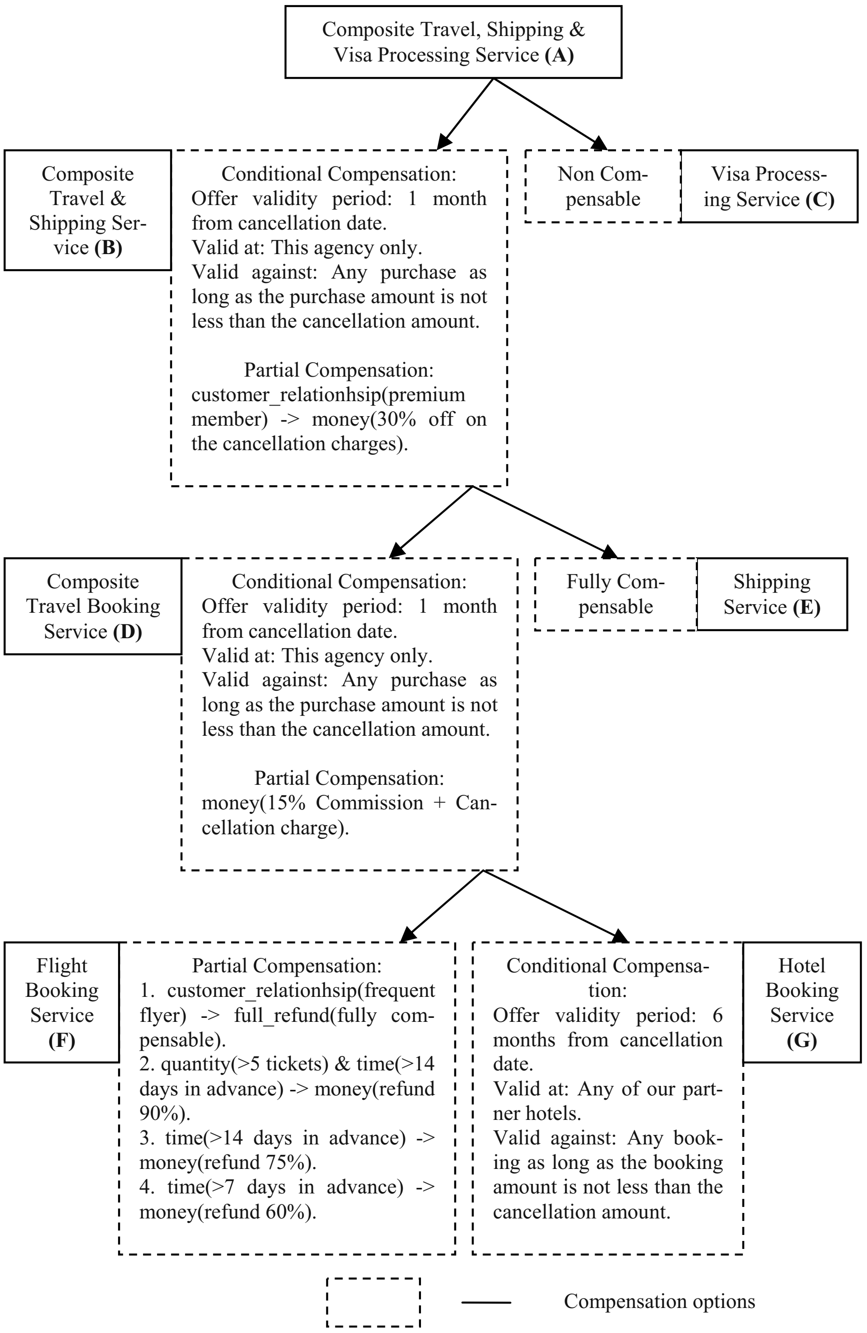

Composite Travel, Shipping & Visa Processing Service (A)
Composite Travel & Shipping Service (B)
Conditional Compensation: Offer validity period: 1 month from cancellation date. Valid at: This agency only. Valid against: Any purchase as long as the purchase amount is not less than the cancellation amount.
Partial Compensation: customer_relationhsip(premium member) -> money(30% off on the cancellation charges).
Non Compensable
Visa Processing Service (C)
Composite Travel Booking Service (D)
Conditional Compensation: Offer validity period: 1 month from cancellation date. Valid at: This agency only. Valid against: Any purchase as long as the purchase amount is not less than the cancellation amount.
Partial Compensation: money(15% Commission + Cancellation charge).
Fully Compensable
Shipping Service (E)
Flight Booking Service (F)
Partial Compensation: 1. customer_relationhsip(frequent flyer) -> full_refund(fully compensable). 2. quantity(>5 tickets) & time(>14 days in advance) -> money(refund 90%). 3. time(>14 days in advance) -> money(refund 75%). 4. time(>7 days in advance) -> money(refund 60%).
Conditional Compensation: Offer validity period: 6 months from cancellation date. Valid at: Any of our partner hotels. Valid against: Any booking as long as the booking amount is not less than the cancellation amount.
Hotel Booking Service (G)
Compensation options

Fig. 3. Example Composite Schema (Extended travel scenario discussed earlier)

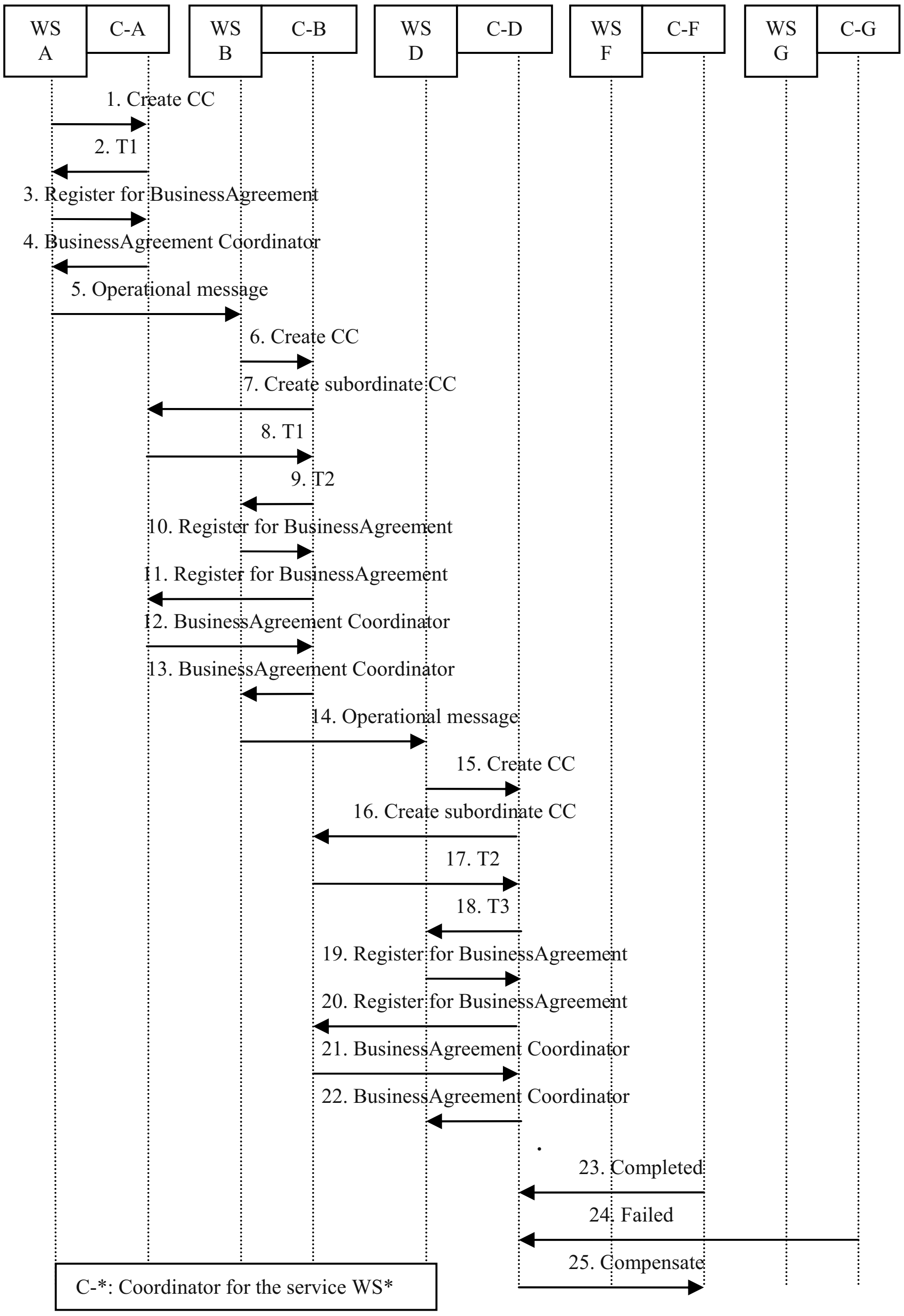

WS A
C-A
WS B
C-B
WS D
C-D
WS F
C-F
WS G
C-G
1. Create CC
2. T1
3. Register for BusinessAgreement
4. BusinessAgreement Coordinator
5. Operational message
6. Create CC
7. Create subordinate CC
8. T1
9. T2
10. Register for BusinessAgreement
11. Register for BusinessAgreement
12. BusinessAgreement Coordinator
13. BusinessAgreement Coordinator
14. Operational message
15. Create CC
16. Create subordinate CC
17. T2
18. T3
19. Register for BusinessAgreement
20. Register for BusinessAgreement
21. BusinessAgreement Coordinator
22. BusinessAgreement Coordinator
23. Completed
24. Failed
25. Compensate
C-*: Coordinator for the service WS*

Fig. 4. An execution of the distributed Business Activity (WS-Transaction) protocol

4. Once the initiating coordinator has got all the information regarding the compensation options possible at lower levels, it passes the information to its parent coordinator (if any). C-B passes the information to C-A.
5. (Optional) The initiating coordinator's parent propagates the information to the root coordinator. For this example, the initiating coordinator's parent (C-A) which needs to communicate with the end user is the root coordinator. However, it might be the case that there is no need to propagate the information regarding compensation options to the root coordinator. It would be sufficient if a lower level coordinator is able to contact the end user directly. Now, whether the lower level coordinator is capable of contacting the end user directly or not, would depend on the user transparency at different levels. Basically, it would depend on the user information passed by the coordinators to the proxy coordinators registered with them (Step 1). One of the benefits of propagating the information to the root coordinator is that the complete picture regarding the state of execution can be presented to the end user.
6. Present the information to the end user and similarly permeate the user's decision to the concerned coordinators after performing the required consistency checks. C-A presents the compensation options possible at F, G, D, and B to the end user. If the user selects F, then G also needs to be selected (consistency check). Inform the concerned coordinators (C-F and C-G) regarding the decision.

Please note that as information regarding the possible compensation options is gathered at run-time, the above algorithm would work irrespective of the binding mechanism (static/dynamic) used.

6 Conclusion and Future Work

While most transaction models acknowledge that compensation is required, it is also one of the most neglected aspects of any transaction model specification. Most transaction models suffice to say that specifying the proper compensation depends on the business logic and as such, is the responsibility of the service provider. While we agree with the above argument, we also stress that compensation is a complex process and needs more infrastructure support. We specifically stress on two aspects, Cost of Compensation and End User Involvement, which are missing in most of the Web services transaction models/specifications. We also show how current standards like BPEL4WS, WS-Transaction can be extended to facilitate the two aspects. We believe that there is as much a need to be able to describe compensating operations semantically as the operations themselves. Towards that end, we specify a simple classification scheme for compensating operations.

Given the current state of technology, we believe that current systems are still not intelligent enough to take complex decisions on their own and human intervention is required. Going forward, we envision systems where it would not be required to define the compensating operations statically, rather, it would be possible for systems to figure out the most optimum compensation dynamically at run-time. We see MAS research as showing a lot of promise in this direction.

Acknowledgement

I would like to thank Dr. K. Vidyasankar and the referees for several valuable suggestions that helped to improve the work in this paper considerably.

References

1. Garcia-Molina, H. and Salem, K. SAGAS. in Stonebraker, M. ed. Readings in database systems, San Francisco, California, 1987, 290-300.
2. Farrag, A.A. and Özsu, M.T. Using semantic knowledge of transactions to increase concurrency. ACM Transactions on Database Systems, 14 (4). 503-525.
3. G. Vossen, K. Vidyasankar. A Multi-Level Model for Web Service Composition. In Proceedings of ICWS 2004.
4. Ferda Tartanoglu, Valérie Issarny, Alexander Romanovsky, Nicole Levy. Coordinated Forward Error Recovery for Composite Web Services. In Proceedings of the 22nd Symposium on Reliable Distributed Systems (SRDS'2003) October 2003.
5. Paulo F. Pires, Marta L.Q. Mattoso, Mário Roberto F. Benevides. Building Reliable Web Services Compositions. Web, Web-Services, and Database Systems 2002, Springer LNCS 2593, ISBN 3-540-00745-8, pp. 59-72, 2003.
6. Gerhard Weiss. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. The MIT Press.
7. Kuno H, Sahai A. "My Agent Wants to Talk to Your Service: Personalizing Web Services through Agents". HPL-2002-114.
8. Tao Jin, Steve Goschnick. Utilizing Web Services in an Agent Based Transaction Model (ABT). In proceedings of AAMAS '03, Melbourne, Australia.
9. Randi Karlsen, Thomas Strandenæs. Trigger-Based Compensation in Web Service Environments. ICEIS (1) 2003: 487-490.
10. Specification: Web Services Tranaction (WS-Transaction). http://www-106.ibm.com/developerworks/webservices/library/ws-transpec/.
11. OASIS Web Services-Composite Application Framework (WS-CAF) Primer. http://www.webservices.org/index.php/article/articleview/1297/1/24/.
12. Specification: BPEL4WS. http://www-106.ibm.com/developerworks/library/ws-bpel/.
13. B. Grosof, Y. Labrou, and H. Chan. A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML. Proc. 1st ACM Conf. on Electronic Commerce (EC-99) Denver, Colorado: ACM Press (1999) 68-77.
14. OWL-S: Semantic Markup for Web Services. http://www.daml.org/services/owl-s/.

Trust Negotiation for Semantic Web Services

Daniel Olmedilla[1], Rubén Lara[2], Axel Polleres[2], and Holger Lausen[3]

[1] L3S and University of Hanover, Germany
olmedilla@l3s.de
[2] DERI Innsbruck, Austria
{ruben.lara, axel.polleres}@deri.at
[3] DERI Galway, Ireland
holger.lausen@deri.ie

Abstract. Semantic Web Services enable the dynamic discovery of services based on a formal, explicit specification of the requester needs. The actual Web Services that will be used to satisfy the requester's goal are selected at run-time and, therefore, they are not known beforehand. As a consequence, determining whether the selected services can be trusted becomes an essential issue. In this paper, we propose the use of the Peertrust language to decide if trust can be established between the requester and the service provider. We add modelling elements to the Web Service Modeling Ontology (WSMO) in order to include trust information in the description of Semantic Web Services. In this scenario, we discuss different registry architectures and their implications for the matchmaking process. In addition, we present a matching algorithm for the trust policies introduced.

1 Introduction

Semantic Web Services [12] aim at providing automatic support for discovery, composition and execution of Web Services by means their explicit semantic annotation, overcoming the limitations of current Web Service technologies. One of the features of Semantic Web Services is that the functionality they provide may depend on the invocation of other services that are dynamically located and, therefore, their characteristics are not completely known at design time. In such a dynamic and open environment, where the interacting parties can be determined at run-time, trust becomes an essential issue. As Semantic Web Services provide P2P interactions between services, trust establishment mechanisms based on a simple client/server approach, in which the requester has to register and/or unconditionally disclose his (maybe private) information to the provider in order to gain access to the service [4], are not appropriate. However, some mechanisms must be in place to determine if trust between the requester and the provider can be reached. Policy languages appear as a solution to bring trust to Semantic Web Services. A policy is a rule that specifies in which conditions a resource (or another policy) might be disclosed to a requester. Related work in [8] uses a trusted matchmaker where services must register providing not only the services description but also the policies associated to that service. A user/agent includes its policy together with its request and the matchmaker filters the available services according to the requester's functional goals together with the requester and service compatibility according to their policies.

That match is performed using goals and authorization policies from the requester and the service providers. However, we believe that there are two types of policies at any entity: sensitive (and therefore the owner will not disclose them) and non-sensitive (which might be made public). A centralized matchmaker assumes that all the parties involved will disclose their policies to it. If parties do not fulfill this requirement, the matchmaker results will not be accurate. In addition, delegation becomes important when more than one entity is involved while taking a decision. For example, suppose Alice wants to buy book at Uni-Book store. Uni-Book offers a discount to any student registered at any university in the region of Lower Saxony (e.g, Hanover University). Alice is a student and she has her student id card but Uni-Book might want to verify that she did not withdrew after she registered. Therefore, Uni-Book delegates its authorization decision to Hanover Registrar (the entity in charge of student registration at Hanover University). Centralized approaches assume that services (and policies) of Hannover Registrar must be available at the registry together with Uni-Book services in order to allow delegation. Furthermore, access control is not longer a one-shot, unilateral affair found in traditional distributed systems or recent proposals for access control on the Semantic Web [6, 22]. The distributed and open nature of the Web requires trust to be established iteratively through a negotiation process where the level of trust increases in each successful iteration. This iterative process has not been taken into account in previous work on semantic web services.

In this paper, we propose an architecture based on a distributed registry and a matchmaking process which provides a solution to the limitations or assumptions described above. We use the PeerTrust language [4] which provides access control through Trust Negotiation to determine whether the establishment of an appropriate trust level between the requester and the provider is possible at discovery time.

Section 2 presents different possible registry architectures that influence what information is made available to the matchmaker and under which assumptions. Section 3 briefly describes trust negotiation and the Peertrust policy language. The inclusion of information disclosure policies in the modeling of Semantic Web Services is discussed in Section 4. Section 5 presents our implementation of the algorithm for the matching of trust policies. Finally, conclusions and future work are presented in Section 6.

2 Registry Architectures

The use of Matchmakers together with service registries has been proposed in order to allow users/agents to find services that fulfill their goals [11, 17]. Service descriptions and matchmakers do not usually take into account trust policies during the process of identifying matching services. However, many useless service invocations (because e.g. access is denied to him) that do not lead to the user's expected results can be avoided by considering trust policies during the matchmaking process. In this section we will only consider issues purely related to the algorithm of determining if trust can be reached between the requester and the provider. Existing security architecture proposals for semantic web services involve the use of a matchmaker where both the requester and the service provider policies must be available [8]. While this approach has several advantages, it is also built on some assumptions that should be reviewed. We believe that service providers will not disclose sensitive policies to a third entity (and loose control over them) and therefore, this would reduce the accuracy of the matchmaker, which can only make use of non-sensitive policies. Even if we assume that the matchmaker is trusted, many companies would not provide their policies (e.g, a resource protected by a policy requiring an employee id from Microsoft or IBM might suggest a secret project between both companies [23]). Consequently, we believe that delegation and negotiation will play an important role on trust and security for Semantic Web Services. In the delegation process (act of delegating a decision to another entity) two entities are involved: delegator (the entity that delegates the decision) and the delegatee (the entity that receives the delegation and takes the decision). Delegation in a centralized matchmaker might not be possible if delegatee's policies are not available in the matchmaker. In this section we describe different possible matchmaking architectures according to where client and server policies are stored (e.g. locally or 3rd party) and where the matchmaking process is done (client side, server side, trusted matchmaker) and we discuss their advantages and drawbacks.

2.1 Centralized Matchmaking

Typically, service providers must register in a centralized registry/directory (e.g., UDDI) where they describe the properties of their services. A potential requester try to find the appropriate service by looking it up in that registry. If a service that matches its goals is found, it retrieves the complete information of that service and invokes it.

Trusted Matchmaker. The scenario presented above directly suggests to have our trust matchmaker together with the goal matchmaker at the registry and therefore both tasks might be done at the same time (as depicted in figure 1). This approach is easy to implement and the fastest (algorithm's computation is performed locally at the registry and only matching services are retrieved) but it has some disadvantages. First of all, the matchmaker must be fully trusted because requester and service providers must

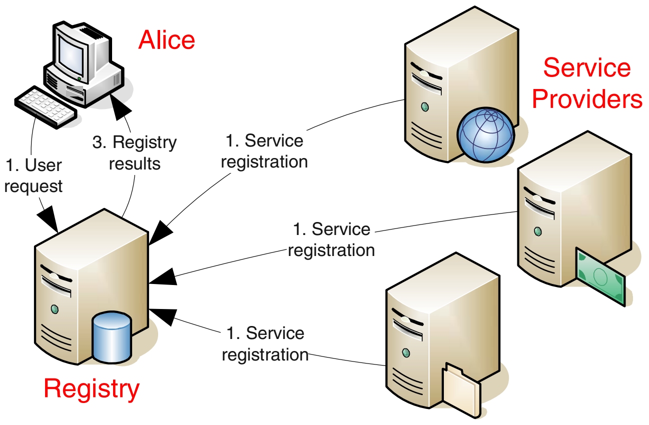


Fig. 1. Centralized Registry and Matchmaker

provide their policies (which may include confidential information) in order to find the matches for the request. This first assumption might be a problem for many users or providers who do not feel comfortable loosing control of their policies. A solution is to distinguish between non-sensitive policies (e.g., a book seller might want or at least does not mind to publish that it gives a discount to students) and sensitive policies (e.g., in the policy "in order to access Bob's health record the requester must be an employee of the Psychology department of a Hospital" someone could infer that Bob has some psychological problems) and only provide non-sensitive policies to the registry. While this solution gives more flexibility to users and providers, it also reduces the accuracy of the matching algorithm. Now some private policies are missing and therefore some possible matches will not be selected (reduction of recall) and many matches will be selected although they will not be usable (reduction of precision). The second big disadvantage is related to delegation. An entity might delegate decisions to other entities (e.g, a client gets the status of preferred client if he is already a client of our company's partners). A centralized matchmaker would then need to have the policies of all the entities which could be involved in the process (the company's partners) or to have mechanisms to retrieve automatically such information. A possible mechanism could be to expect all the company's partners to publish a service that provide access to their policies. Although this seems to be difficult, if we reached to have such a service at each delegatee entity, this service must be as well protected with some policies in order to not allow anyone to retrieve those policies. A list of allowed matchmakers might be provided or a policy language (e.g., Peertrust) could protect them. In any case, delegation might become a time consuming task and decrease the performance of the algorithm. Batch processes or caching might be some possibilities to minimize this problem.

Local Client's Matchmaker. A different approach is to have the matchmaker locally at the client side. It must retrieve all services information (and policies) from the registry and run the trust policies matchmaking algorithm locally. While this approach allows the use of all the client's private policies and certificates/credentials, it still has the disadvantage that providers would disclose only their public policies reducing the accuracy of the algorithm. Furthermore, this approach seems not to be scalable to registries with a high number of services. Service descriptions and policies must be retrieved from the registry to the user computer with the corresponding network overload, and this approach requires a client machine powerful enough to run the algorithm in a reasonable time.

2.2 Distributed Matchmaking

Above we have described how a centralized and trusted registry might store the policies from users and service providers and the implications for the trust policies matchmaking process. It turned out that many disadvantages appear when relying on such an architecture. In this section we propose an alternative architecture where service providers do not need to register at a specific registry and, most important, they do not need to provide their policies to any third entity (trusted or not).

In [21] such an architecture is described where centralized registries are replaced by a Peer-to-Peer network. Whenever a new service provider wants to offer its services, it must just join the network. This approach allows service providers not to loose control over

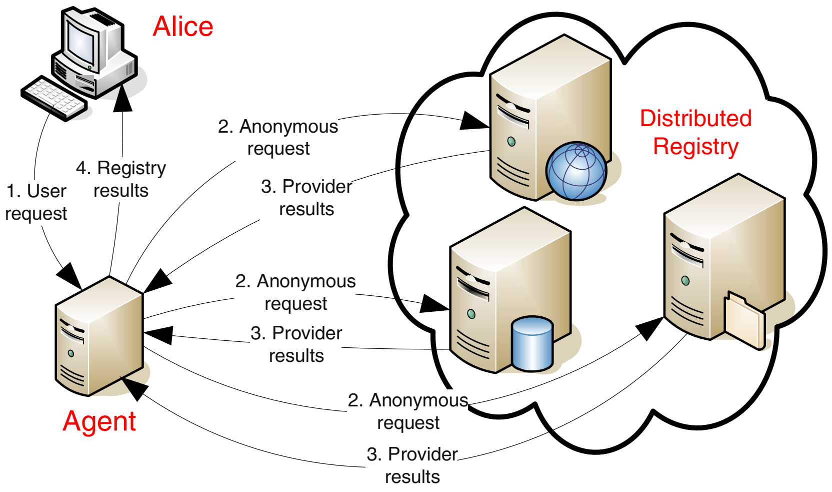


Fig. 2. Distributed Registry and Matchmaker

the descriptions of their services and, in our context, not to disclose private information within their policies to other entities. We propose to follow such an architecture and allow different agent to provide users access to the service descriptions from the providers. A user might then send a query together with his policies to an agent he trusts in[1]. The agent sends the same query to the network. This query is distributed to the peers on the network and each peer on the network applies a matching algorithm. Whenever a peer has matches, it sends them back to the agent which joins the results and give them to the user. This architecture is depicted in Figure 2.

In this context there are several issues that must be solved. The user might not want that each provider knows about his policies. The agent plays an important role here. It is not only a mediator between the user and the distributed registry, but it is also in charge of making the query anonymous so that the providers do not know the real owner of the query they receive (making the policies anonymous too). A second problem is that we moved the matching algorithm to the service provider side. The question that then arises is how to proceed if the provider returns matches that are not real matches. Although the possibility of a provider faking the match results does exist, the provider would not benefit from that behaviour, because those false matches will lead to failed invocations. With false matches a service provider only obtains extra, unnecessary and unsuccessful invocations.

The advantages of this architecture are summarized as follows:

– *Distributed registry service.* A distributed registry service allows service providers to keep control over the description of their services as well as the policies associated

[1] Here it is important to note that different groups of users might use different trusted agents (e.g., the university might set up an agent for its students and professors while a company could use a different one).

to them. Additionally, we gain the nice properties of distributed environments like e.g. no single point of failure and better scalability .

- *Distributed matchmaking.* The matching algorithm might be computationally expensive. In a distributed architecture the computation time is shared by the different providers improving performance and scalability.
- *Privacy kept on service provider policies.* We believe that it is not realistic to ask service providers to disclose their (maybe very sensitive) policies to a centralized registry (even if it is trusted). In a distributed approach, servers keep those policies locally and private.

3 Peertrust and Trust Negotiation

In the previous sections we have mentioned policies and their importance to improve the matchmaking process. Although a policy language (REI,[7]) was already used in [8], in this paper we use the Peertrust language instead because it is especially designed to enable delegation and trust negotiation. WS-Policy [2] cannot be directly used for our purposes, as it does not describe trust policies but a general framework to describe policies for Web Services. Describing Peertrust policies in this framework will be considered in the future.

Peertrust consists on a set of guarded distributed logic programs. The Peertrust language [14] is a policy language based on Definite Horn clauses with some extensions to provide e.g. delegation. Rules are of the form

$$lit_0 \leftarrow lit_1, \ldots, lit_n$$

In the remainder of this section, we concentrate on the syntactic features that are unique to the PeerTrust language and we will consider only positive authorizations.

In our previous example, Alice is a student and she wants to buy a book at Uni-Book. Uni-Book has a policy where it states which requirements any buyer must fullfil. The policy looks:

```
discount(BookTitle) $ Buyer ←
  studentId(Buyer) @ University @ Buyer,
  validUniversity(University),
  studentId(Buyer) @ University.
```

Uni-Book's policies could be much more complex, but this simple policy will help us to introduce the syntax of the Peertrust language. Three conditions must be fullfiled before any buyer gets a discount at Uni-Book (represented in the body of the policy). Starting with the simplest one, the second condition checks if a university is a valid university in order to get the discount (if it belongs to the region of Lower Saxony). The following is a list of Lower Saxony universities which are valid universities in our context.

```
validUniversity(“Hanover University”).
...
validUniversity(“Bremen University”).
```

In the third condition, @ *Issuer* represents a delegation process on Issuer. In this case, Uni-Book delegates on the university the proving of the buyer's student status (if she is still a student or if she registered and afterwards withdrew). In addition, the *Issuer* argument can be a nested term containing a sequence of issuers, which are evaluated starting at the outermost layer. In the first condition, two "@" are nested, which means that when Uni-Book receives the request from Alice, it asks her to prove that she is a student at a university and this proof must contain a digital credential signed by the university. In our case, University of Hanover issued a digital credential to Alice when she registered (like credentials in real life where the university issues a student card to registered students). As Alice already possesses this credential, she sends it to Uni-Book. If she had not had it, she should have had to send a request to "Hanover University" (which in this case would be *studentId("Alice") @ "Hanover University"*) to get such a credential.

On the head of the policy above, a symbol "$" appears. This *$ Requester* represents the party that sent us the query to allow parties to include the party that sent the query into the policy. The *Requester* argument can be nested, too, in which case it expresses a chain of requesters, with the most recent requester in the outermost layer of the nested term.

Summarizing, when Alice requests Uni-Book for a discount on a book, Uni-Book asks Alice to prove that she is a student in a university. Alice is a student at Hanover University so she discloses her credential (signed by University of Hanover) to Uni-Book. Uni-Book checks that the university is a valid university and sends a request to Hanover University in order to check if Alice is still a student there (she has not withdrew after her registration). If Hanover University answers that Alice is still a student, she will get the discount.

Using the *Issuer* and *Requester* arguments, we can delegate evaluation of literals to other parties and also express interactions and the corresponding negotiation process between parties. In this paper we will not use other features of the Peertrust language like *local rules and signed rules*, *guards* and *public and private predicates*. For more details, we refer to [14] for a detailed description.

Continuing with our example, Uni-Book requires Alice to prove that she is a student at a university. However, Alice is not willing to disclose her student id card to anyone who requests it. Contrary, she will disclose her credential only to entities that are members of the "Better Business Bureau". Therefore she has the following policy:

```
student('Alice') $ Requester ←
  member(Requester) @ 'Better Business Bureau' @ Requester.
```

Describing policies on both sides (Alice and Uni-Pro) allows a negotiation process where at each iteration trust is increased. After Uni-Book proves Alice that it belongs to the Better Business Bureau, Alice knows enough to disclose her student card what makes the negotiation succeed and, therefore, Alice gets the discount. After several iterations (where also other entities like Hannover University might be involved) the level of trust is enough to perform the transaction.

4 Application to Semantic Web Services

A Web Service provider can act as requester for other services in order to provide its declared functionality. In the general case, a provider will specify subgoals that have to be accomplished in order to achieve its overall functionality. These subgoals are defined in the orchestration of the service, which in addition describes related issues such as the control flow and data flow among the subgoals. These subgoals have to be resolved at run-time, and actual Web Services that fulfill the defined subgoals have to be located. In general, the actual Web Services that will be used to provide the functionality required by the requester can be located at run-time based on a formal, explicit definition of the requester requirements. Therefore, an essential aspect to determine what services are applicable to fulfill the requester's goal is to be able to decide which candidate services can be trusted.

Consequently, trust information must be part of the description of Semantic Web Services, and this information has to be exploited during the discovery process in order to determine matching services.

We use the Web Service Modelling Ontology (WSMO)-Standard v0.3 [18] as the modelling means to describe Semantic Web Services and we situate the information disclosure policies into the appropriate modelling elements in order to exploit it during the discovery process [9]. Our main reasons to choose WSMO instead of other proposals such as OWL-S [16], IRS-II [13] or METEOR-S [20] are: 1) It allows the use of arbitrary logical expressions in the description of the service functionality, thus providing more complete descriptions than the other approaches, and 2) It uses logic programming (F-Logic [10]) to describe the logical expressions used in the description of the service, which makes possible, in the future, the alignment of the trust policies described in the Peertrust language and the functionality descriptions in WSMO.

In WSMO-Standard, goals describe the objectives that a client may have, while capabilities describe the functionality of the service. A requester describes his goal by specifying its postconditions (the state of the information space that is desired) and effects (the state of the world that is desired). Capabilities also describe postconditions of the service (the information results of its execution) and effects (the state of the world after its execution). With this information, the discovery process can match the requester's goal against the available service capabilities and determine what services provide the required functionality.

However, the service capability also needs to describe what information the Web Service requires to provide its service (preconditions) and its assumptions. Therefore, the preconditions is where the policies described in the previous sections come into play. The Web Service will state in its preconditions what information the requester has to disclose (including credentials) to gain access to the service. Credentials can be described using the ontology described in [3] and included in the preconditions of the service. In addition to enumerate all the information items that the service requires for its execution, we add the Peertrust expressions that describe the exact policies the service employs. Since in WSMO-Standard preconditions are described via axiom definitions, we need to extend this description to add the relevant policies. Figure 3 depicts our modelling in F-Logic [10]. A precondition includes any number of axiom definitions and any number of policies (encoded as strings that represent a Peertrust formula). Notice that we use F-Logic, as it is the language used in WSMO-Standard to describe preconditions.

```
precondition [
  axiom =>> axiomDefinition,
  policies =>> string
]
```

Fig. 3. Definition of precondition

By modelling preconditions in this way in the service capability, we capture both the information that the service requires from the requester and what policies apply to gain access to the service. Therefore, we are modelling not only the functional description of the service in terms of preconditions, assumptions, postconditions and effects but also what are the policies that describe what information the requester must disclose in order to be trusted by the provider. As discussed in Section 2, the provider might not want to make these policies available. For this reason, we propose to follow the distributed architecture described in Section 2.2, where the policies are kept private at the provider side. In addition, we have to model at requester's side the information disclosure policies of such requester i.e. what information he is willing to disclose and under what conditions.

The description of the requester's information disclosure policies is modelled in F-Logic in figure 4. A policy contains a set of information items, for which the actual data (an ontology instance) to be disclosed and the disclosure policy (a Peertrust formula) are specified. Information items that are unconditionally disclosed will have an empty Peertrust formula. Notice that the data disclosed can be an instance of any ontology concept, including a credential.

```
infoDisclosurePolicy [
  infoItems =>> infoItem [
    data => ontologyInstance,
    peerTrustExpression => string
  ]
]
```

Fig. 4. Definition of requester's information disclosure policies

Having such information disclosure policies described at the requester side and the preconditions at the provider side, all the declarative elements needed to determine if the trust establishment is possible are in place. We know the conditions the requester must fulfill to be trusted by the provider (described using Peertrust in the preconditions definition), and what information the requester will disclose and under what conditions. Using the respective Peertrust policies and the matching algorithm described in section 5 we can determine if trust can be reached.

It is important to notice that what we determine is whether trust can be reached between the requester and the provider based on published policies. The actual interchange of messages (credentials) to really establish trust at invocation time, and the modelling of the service choreography [19] for that interchange is out of the scope of this paper.

After modelling the elements above, a relevant issue is determining how the matchmaker can access the information described by the requester and the provider. In the distributed architecture proposed in section 2.2, the description of the services is kept on the provider, and the (anonymous) request is sent by the user agent to the peers. In this approach, the matchmaking process is performed at every provider and the results returned to the user agent. Therefore, the provider policies will be available to the matchmaker, as the matchmaking process will take place on the provider's side. However, not only the requester's goal but also its information disclosure policies or the subset relevant for this goal i.e. the information that the requester is willing to disclose (under certain conditions) to achieve the goal has to be submitted to the peers by the user agent, as these policies are necessary to determine whether trust can be reached between the parties.

An obvious drawback of this approach is that the requester might be willing to disclose a big set of information that is not sensitive, and submitting this information to all the peers creates an information overload. Therefore, we propose an alternative solution in which the matchmaker requests to the user agent that submitted the query the information it needs to satisfy the service requirements. The user agent will have access to the requester information disclosure policies, and will send back to the matchmaker the relevant information together with the relevant (anonymous) policies. To do so, the user agent will expose a Web Service that receives the requests from the matchmakers and send back the appropriate policies. This service will only be accessible to providers that are part of the P2P network of providers, that are assumed to be trusted.

5 Algorithm Implementation

In this section we present our implementation of an algorithm that performs the matching of trust policies. Each of the service providers has this algorithm and it runs it locally whenever a new request (query hereafter) from an agent arrives.

We limit ourselves to the evaluation of the policies described in previous sections i.e. the matching of the Peertrust policies described for the requester and the provider. For more details about matching services with requests we refer the reader to [9] and [11].

Guarded distributed logic programs can be evaluated in many different ways. This flexibility is important, as different registries, and even different service providers within the same registry, may prefer different approaches to evaluation. As we provide explicit delegation in our policies the service provider might include other entities (peers) to delegate decisions. We will present a simple evaluation algorithm for PeerTrust that is based on the cooperative distributed computation of a proof tree, with all peers employing the same evaluation algorithm. The algorithm assumes that each peer uses a queue to keep track of all active proof trees and the expandable subqueries in each of these trees. The proof trees contain all information needed, including used rules, credentials and variable instantiations. Peers communicate with one another by sending queries and query answers to each other.

The following sketch of the algorithm uses EITHER:/OR: to express a non-deterministic choice between several possible branches of the algorithm.

```
Let $TreeList$ denote the structure with all active proof trees
Set $TreeList := []$
Let $Tree$ denote the structure holding $Query\$Requester$ and $Proof$
        both of which may still contain uninstantiated variables
Loop
    EITHER:
        Receive $Tree$: a query to answer / a goal to prove
        Add_New_Tree($Tree$, $TreeList$)
    OR:
        Receive $Answer(Tree)$
        Add_Answers($Answer(Tree)$,$TreeList$)
    OR:
        Receive $Failure(Tree)$ from peer
        Send $Failure(Tree)$ to Requester
        Remove_Tree($Failure(Tree)$, $TreeList$)
    OR:
        Process_a_Tree($TreeList$)
end Loop
```

At each step a peer can receive a new query from another peer, receive answers, learn that there are no answers for a query it previously sent to a peer, or selects one of its active trees for processing . If this tree is already complete, the answers can be returned to the peers who requested this evaluation. If the tree contains subqueries which still have to be evaluated, the peer selects one of them and tries to evaluate it.

```
Process_a_Tree($TreeList$)
Let $NewTrees$ denote the new proof trees
Set $NewTrees := []$
Select_Tree($Tree$, $TreeList$, $RestOfTreeList$)
IF all subqueries in $Tree$ are already evaluated
THEN
    Send $(Answer(Tree))$ to Requester
    $TreeList := RestTreeList$
ELSE
    Select_Subquery ($SubQuery$,$Tree$)
    IF $SubQuery$ can be evaluated locally
    THEN
        Loop while new local rules are found
            Expand $SubQuery$ into its subgoals
            Update_Tree($Tree$,$NewSubgoals$)
            Add_Tree($Tree$,$NewTrees$)
        End loop
    ELSE //if it is a goal with an “@ Issuer” suffix,
        // indicating remote evaluation
        IF peer has $Signed\_Rule(SubQuery)$
            Loop while new signed rules are found
                Expand $SubQuery$ into its subgoals
                Update_Tree($Tree$,$NewSubgoals$)
                Add_Tree($Tree$,$NewTrees$)
            End loop
        ELSE
            Send $Request(SubQuery)$ to $Issuer$)
            Update_Status($Tree$, $waiting$)
        END IF
    END IF
    IF no local or remote expansion for $SubQuery$ was possible
        Send $(Failure(Tree))$ to Requester
    ELSE
        Add_New_Trees
            ($NewTrees$,$RestTreeList$,$NewTreeList$)
    END IF
    $TreeList := NewTreeList$
END IF
```

Expansion of subqueries is done either locally (using the peer's rules and signed rules) or by sending the subquery to a remote peer (in case of delegation). Many queries per proof can be active (i.e., awaiting answers and being processed) at any time. Each new query from a remote peer starts a new proof tree while answers from remote peers are "plugged into" existing proof trees. An example of a query expansion in a proof tree is depicted in figure 5, where a tree is expanded into two and then three trees. Each tree structure contains at least root and leaves, plus any additional information from the proof, including credentials, that we want to keep and/or return to the requester. If one proof tree for the original query is completed, then the negotiation is over and the requester obtains access to the desired resource.

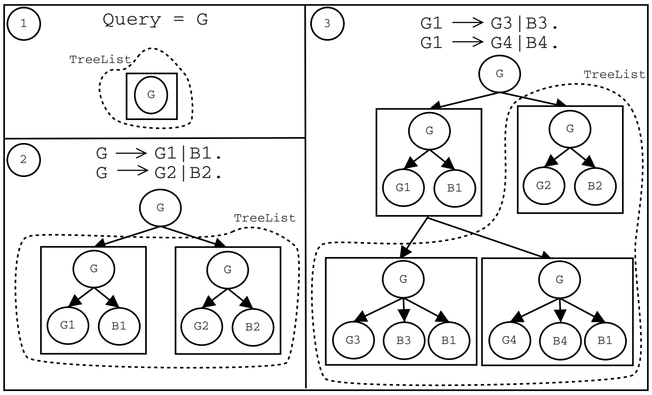


Fig. 5. Resource access control scenario

This algorithm can be extended and improved in many different ways. For example, it can be made more efficient and effective at run time by generalizing the definition of a query, allowing iteration through a set of query answers, allowing intensional query answers, support for caching of query answers, and prioritization of rules a la [5]. Alternatively, the algorithm can be revamped in ways that will allow different peers to choose different evaluation algorithms, along the lines of [23], or to provide provable guarantees of completeness and termination, as offered by the algorithms of [23]. No matter what revisions are made, however, at its heart any evaluation algorithm will be working to construct a certified proof tree.

6 Conclusions and Future Work

Semantic Web Services bring dynamism to current Web Service technologies, and the actual services that will be employed to satisfy a goal are determined at run time. Therefore, requesters interact with services that they do not know beforehand, and they have to determine if they can trust such services. In this context, modelling trust information in Semantic Web Services becomes necessary. In this paper, we include trust policies in WSMO-Standard, together with the information disclosure policies of the requester, using the Peertrust language. Peertrust provides the means to perform trust negotiation and delegation. As the matchmaker needs to have access to the requester and provider policies, in order to match not only the requester functional requirements but also trust information, the architecture of the registry and matchmaker becomes a relevant issue. We have proposed a distributed registry and matchmaker architecture that allows the service providers to keep their policies private, thus not forcing them to disclose sensitive information. It also improves the efficiency and scalability of the solution. We have also implemented an algorithm that matches the requester and provider Peertrust policies to determine if trust between them can be established. Future work includes the integration of this algorithm with the functional matching algorithm (matching of the requester goal and the service capability) in our P2P network (Edutella, [15]). We are also studying the possibility of extending our work to Web Services on Grid environments. Some previous work on using Peertrust on Grid environments can be found in [1]. At this point, we use strings to model the Peertrust formulas in the description of the service and in the description of the requester information disclosure policies. However, Peertrust expressions can be modelled directly as F-Logic formulas, extending their semantics to include Peertrust features such as delegation. As part of the integration of the functional matching and the trust matching algorithms, we plan to model Peertrust expressions as F-Logic formulas. We will also investigate other possibilities to better integrate policies in WSMO. Finally, we plan to refine the approach we propose to give the provider access to use the requester's information disclosure policies, in which we assumed the providers requesting access to these policies is trusted.

Acknowledgments. This research is partially funded by the projects ELENA (http://www.elena-project.org, IST-2001-37264), REWERSE (http://rewerse.net, IST-506779), Knowledge Web (http://knowledgeweb.semanticweb.org/, FP6-507482), DIP (http://dip.semanticweb.org/, FP6-507683) and SWWS (http://swws.semanticweb.org/, IST-2001-37134).

References

1. J. Basney, W. Nejdl, D. Olmedilla, V. Welch, and M. Winslett. Negotiating trust on the grid. In *Proc. of 2nd Workshop on Semantics in P2P and Grid Computing*, New York, 2004, May 2004.
2. D. Box, F. Curbera, M. Hondo, C. Kaler, D. Langworthy, A. Nadalin, N. Nagaratnam, M. Nottingham, C. von Riegen, and J. Shewchuk. Web services policy framework (ws-policy). http://www-106.ibm.com/developerworks/library/ws-polfram/, May 2003.
3. G. Denker, L. Kagal, T. Finin, M. Paolucci, and K. Sycara. Security for daml web services: Annotation and matchmaking. In *Proceedings of the 2nd International Semantic Web Conference*, Sanibel Island, Florida, USA, Oct. 2003.
4. R. Gavriloaie, W. Nejdl, D. Olmedilla, K. Seamons, and M. Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *Proc. of the 1st European Semantic Web Symposium*, Heraklion, Greece, May 2004.
5. B. Grosof. Representing e-business rules for the semantic web: Situated courteous logic programs in RuleML. In *Proceedings of the Workshop on Information Technologies and Systems (WITS)*, New Orleans, LA, USA, Dec. 2001.
6. L. Kagal, T. Finin, and A. Joshi. A policy based approach to security for the semantic web. In *Proceedings of the 2nd International Semantic Web Conference*, Sanibel Island, Florida, USA, Oct. 2003.
7. L. Kagal, T. Finin, and A. Joshi. A policy language for a pervasive computing environment. In *4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2003)*, Lake Como, Italy, June 2003.
8. L. Kagal, M. Paoucci, N. Srinivasan, G. Denker, T. Finin, and K. Sycara. Authorization and privacy for semantic web services. In *AAAI 2004 Spring Symposium on Semantic Web Services*, Stanford University, Mar. 2004.
9. U. Keller, R. Lara, A. Polleres, and H. Lausen. Inferencing support for semantic web services: Proof obligations. http://www.wsmo.org/2004/d5/d5.1/v0.1/, Apr. 2004. WSML working draft.
10. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.
11. L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proceedings of the 12th International Conference on the World Wide Web*, Budapest, Hungary, May 2003.
12. S. McIlraith, T. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems, Special Issue on the Semantic Web*, 16(2):46/53, March/April 2001.
13. E. Motta, J. Domingue, L. Cabral, and M. Gaspari. Irs-ii: A framework and infrastructure for semantic web services. In *2nd International Semantic Web Conference (ISWC2003)*. Springer Verlag, October 2003.
14. W. Nejdl, D. Olmedilla, and M. Winslett. PeerTrust: automated trust negotiation for peers on the semantic web. Technical Report, Oct. 2003.
15. W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, and T. Risch. Edutella: A P2P networking infrastructure based on RDF. In *Proceedings of the 11th International World Wide Web Conference (WWW2002)*, Hawaii, USA, June 2002.
16. OWL-S services coalition. OWL-S: semantic markup for web services. http://www.daml.org/services/owl-s/1.0/owl-s.pdf, November 2003.
17. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In I. Horrocks and J. Handler, editors, *1st Int. Semantic Web Conference (ISWC)*, pages 333–347. Springer Verlag, 2002.
18. D. Roman, H. Lausen, and U. Keller. Web service modeling ontology - standard. http://www.wsmo.org/2004/d2/v0.3/, Mar. 2004. WSMO working draft.

19. D. Roman, L. Vasiliu, C. Bussler, and M. Stollberg. Choreography in wsmo. http://www.wsmo.org/2004/d14/v0.1/, Apr. 2004. WSMO working draft.
20. K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller. Adding semantics to web services standards. In *1st International Conference on Web Services (ICWS'03)*, pages 395–401, June 2003.
21. U. Thaden, W. Siberski, and W. Nejdl. A semantic web based peer-to-peer service registry network. Technical report, Learning Lab Lower Saxony, 2003.
22. G. Tonti, J. M. Bradshaw, R. Jeffers, R. Montanari, N. Suri, and A. Uszok. Semantic web languages for policy representation and reasoning: A comparison of KAoS, Rei and Ponder. In *Proceedings of the 2nd International Semantic Web Conference*, Sanibel Island, Florida, USA, Oct. 2003.
23. T. Yu, M. Winslett, and K. Seamons. Supporting Structured Credentials and Sensitive Policies through Interoperable Strategies in Automated Trust Negotiation. *ACM Transactions on Information and System Security*, 6(1), Feb. 2003.

An Efficient Algorithm for OWL-S Based Semantic Search in UDDI

Naveen Srinivasan, Massimo Paolucci, and Katia Sycara

Robotics Institute, Carnegie Mellon University, USA
{naveen, paolucci, katia}@cs.cmu.edu

Abstract. The increasing availability of web services demands for a discovery mechanism to find services that satisfy our requirement. UDDI provides a web wide registry of web services, but its lack of an explicit capability representation and its syntax based search provided produces results that are coarse in nature. We propose to base the discovery mechanism on OWL-S. OWL-S allows us to semantically describe web services in terms of capabilities offered and to perform logic inference to match the capabilities requested with the capabilities offered. We propose OWL-S/UDDI matchmaker that combines the better of two technologies. We also implemented and analyzed its performance.

1 Introduction

Web Services have promised to change the Web from a database of static documents to an e-business marketplace. Web Service technology are being adapted by Business-to-Business applications and even in some Business-to-Consumer applications. The widespread adoption of web services is due to its simplicity and the data interoperability provided by its components namely XML [7], SOAP [10] and WSDL [11].

With the proliferation of Web Services, it is becoming increasingly difficult to find a web service that will satisfy our requirements. Universal Description, Discovery and Integration [8] (here after UDDI) is an industry standard developed to solve the web service discovery problem. UDDI is a registry that allows businesses to describe and register their web services. It also allows businesses to discover services that fit their requirement and to integrate them with their business component.

While UDDI has many features that make it an appealing registry for Web services, its discovery mechanism has two crucial limitations. First limitation is its search mechanism. In UDDI a web service can describe its functionality using a classification schemes like NAISC, UNSPSC etc. For example, a Domestic Air Cargo Transport Service can use the UNSPSC code 78.10.15.01.00 to describe it functionality. Although we can discover web services using the classification mechanism, the search would yield coarse results with high precision and recall errors. The second shortcoming of UDDI is the usage of XML to describe its data model. UDDI guarantees syntactic interoperability, but it fails to provide a semantic description of its content. Therefore, two identical XML descriptions may have very different meaning, and vice versa. Hence, XML data is not machine understandable. XML's lack of explicit semantics proves to be an additional barrier to the UDDI's discovery mechanism.

The semantic web initiative [5] addresses the problem of XML's lack of semantics by creating a set of XML based languages, such as RDF and OWL, which rely on ontologies that explicitly specify the content of the tags. In this paper, we adopt OWL-S [3], an OWL [15] based ontology, that can be used to describe the capabilities of web services. Like UDDI, OWL-S allows a web service to describe using the classification schemes. In addition, OWL-S provides a capability-based description mechanism [6] to describe the web service. Using capability-based description we can express the functionality of the web service in terms of inputs and precondition they require and outputs and effects they produce. Capability-based search will overcome the limitations of UDDI and would yield better search results.

In this paper we propose an OWL-S/UDDI Matchmaker which takes advantage of UDDI's proliferation in the web service technology infrastructure and OWL-S's explicit capability representation. In order to achieve this symbiosis we need to store the OWL-S profile descriptions inside an UDDI registry, hence we provide a mapping between the OWL-S profile and the UDDI data model based on [1]. We also enhance the UDDI registry with an OWL-S matchmaker module which can process the OWL-S description, which is present in the UDDI advertisements. The matchmaking component is completely embedded in the UDDI registry. We believe that such architecture brings both these two technologies, working toward similar goals, together and realize their co-dependency among them. We also added a *capability port* to the UDDI registry, which can used to search for web services based on their capabilities.

The contributions of this paper are an efficient implementation of the matching algorithm proposed in [2], an architecture that is tightly integrated with UDDI, an extension of the UDDI registry and the API to add capability search functionality, preliminary experiments showing scalability of our implementation and an update of the mapping described in [1] to address the latest developments in OWL-S and UDDI.

The rest of the paper is organized as follows; we first describe UDDI and OWL-S followed by the UDDI search mechanism. In Section 3 we describe the architecture of the OWL-S/UDDI matchmaker and an updated mapping between OWL-S profile and UDDI. In Section 4 we present our efficient implementation of the matching algorithm, followed by experimental results comparing the performances of our OWL-S/UDDI Matchmaker implementation with a standard UDDI registry and finally we conclude.

2 UDDI and OWL-S

UDDI [8] is an industrial initiative aimed to create an Internet-wide network of registries of web services for enabling businesses to quickly, easily, and dynamically discover web services and interact with one another. OWL-S is an ontology, based on OWL, to semantically describe web services. OWL-S is characterized by three modules: *Service Profile*, *Process Model* and *Grounding*. Service Profile describes the capabilities of web services, hence crucial in the web service discovery process. For the sake of brevity of this paper, we are not going into the details of OWL-S and UDDI we assume that the readers are familiar with it, for more information see [3] and [8].

2.1 UDDI Search Mechanism

UDDI allows a wide range of searches: services can be searched by name, by location, by business, by bindings or by TModels. For example it is possible to look for all services that have a WSDL representation, or for services that adhere to Rosetta Net specification. Unfortunately, the search mechanism supported by UDDI is limited to keyword matches and does not support any inference based on the taxonomies referred to by the TModels. For example a car selling service may describe itself as "New Car Dealers" which is an entry in NAICS, but a search for "Automobile Dealers" services will not identify the car selling service despite the fact that "New Car Dealers" is a subtype of "Automobile Dealers". Such semantic matching problem can be solved if we use OWL, RDF etc instead of XML.

The second problem with UDDI is the lack of a power full search mechanism. Search by Category information is the only way to search for services, however, the search may produce lot of results with may be of no interest. For example when searching for "Automobile Dealer", you may not be interested in dealers who don't accept a pre-authorized loan or credit cards as method of payments. In order to produce more precise search results, the search mechanism should not only take the taxonomy information into account but also the inputs and outputs of web services. The search mechanism resulted in combining the semantic base matching and the capability search is far more effective than the current search mechanism. OWL-S provides both semantic matching capability and capability base searching, hence a perfect candidate.

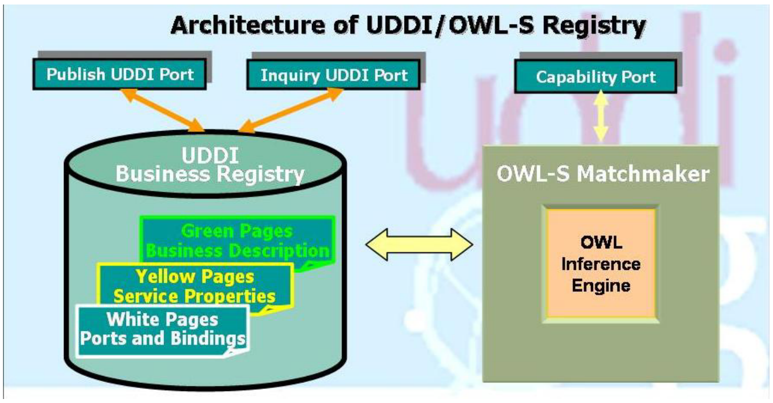

Architecture of UDDI/OWL-S Registry
Publish UDDI Port
Inquiry UDDI Port
Capability Port
UDDI Business Registry
Green Pages Business Description
Yellow Pages Service Properties
White Pages Ports and Bindings
OWL-S Matchmaker
OWL Inference Engine

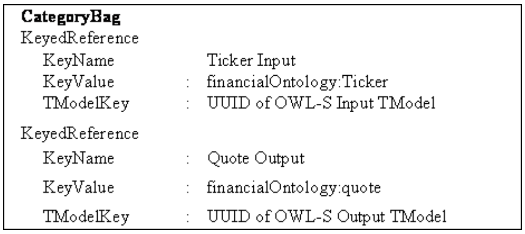
Fig. 1. Architecture of OWL-S / UDDI Matchmaker

3 OWL-S / UDDI Matchmaker Architecture

In order to combine OWL-S and UDDI, we need embed an OWL-S profile description in a UDDI data structure (we discuss this embedding in Section 3.1), and we need to augment the UDDI registry with an OWL-S Matchmaking component, for processing OWL-S profile information. The architecture of the combined OWL-S/UDDI registry is shown in Fig 1. The matchmaker component in this architecture, unlike the previous version discussed in [2], is tightly coupled with the UDDI registry. By tightly coupled we mean the matchmaker component relies on the UDDI registry's ports (publish and inquiry) for its operations.

On receiving an advertisement through the publish port the UDDI component, in the OWL-S/UDDI matchmaker, processes it like any other UDDI advertisement. If the advertisement contains OWL-S Profile information, it forwards the advertisement to the matchmaking component. The matchmaker component classifies the advertisement based on the semantic information present in the advertisement.

A client can use the UDDI's inquiry port to access the searching functionality provided by the UDDI registry, however these searches neither use the semantic information present in the advertisement nor the capability description provided by the OWL-S Profile information. Hence we extended the UDDI registry by adding a *capability port* (see Fig 1) to solve the above problem. As a consequence, we also extended the UDDI API to access the capability search functionality of the OWL-S/UDDI matchmaker. Using the capability port, we can search for services based on the capability descriptions, i.e. inputs, outputs, pre-conditions and effects (IOPEs) of a service. The queries received through the capability port are processed by the matchmaker component, hence the queries are semantically matched based on the OWL-S Profile information. The query response contains list of *Business Service* keys of the advertisements that match the client's query. Apart from the service keys, it also contains useful information, like matching level and mapping, about each matched advertisement. The *matching level* signifies the level of match between the client's request and the matched advertisement. The *mapping* contains information about the semantic mapping between the request's IOPEs and the advertisement's IOPEs. Both these information can be used for selecting and invoking of an appropriate service from the results.



CategoryBag		
KeyedReference		
KeyName		Ticker Input
KeyValue	:	financialOntology:Ticker
TModelKey	:	UUID of OWL-S Input TModel
KeyedReference		
KeyName	:	Quote Output
KeyValue	:	financialOntology:quote
TModelKey	:	UUID of OWL-S Output TModel

Fig. 2. TModel for Stock Quote Service

3.1 Embedding OWL-S in UDDI

The OWL-S/UDDI registry requires the embedding of OWL-S profile information inside UDDI advertisements. We adopt the OWL-S/UDDI mapping mechanism described in [1]. The mechanism uses a one-to-one mapping if an OWL-S profile element has a corresponding UDDI element, such as, for example, the contact information in the OWL-S Profile. For OWL-S profile elements with no corresponding UDDI elements, it uses a T-Model based mapping. The T-Model mapping is loosely based on the WSDL-to-UDDI mapping proposed by the OASIS committee [13]. It defines specialized UDDI TModels for each unmapped elements in the OWL-S Profile like OWL-S Input, Output, Service Parameter and so on. These specialized TModels are used just like the way NAICS TModel is used to describe the category of a web service. Fig 2 illustrates an OWL-S/UDDI mapping of a Stock Quoting service whose input is a company ticker symbol and its output is the company's latest quotes.

In our work we extended the OWL-S/UDDI mapping to reflect the latest developments in both UDDI and OWL-S. Fig 3 shows the resulting OWL-S/UDDI mapping. Furthermore we enhanced the UDDI API with the OWL-S/UDDI mapping functionality, so that OWL-S Profiles can be converted into UDDI advertisements and published using the same API.

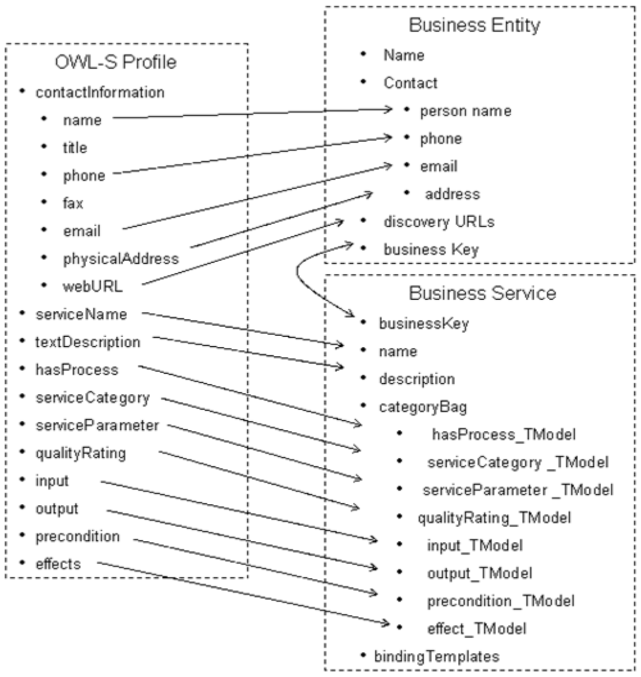


Fig. 3. Mapping between OWL-S Profile and UDDI

4 Achieving Matching Performance

A naive implementation of the matching algorithm described in [2] would match the inputs and the outputs of the request against the inputs and the outputs of all the advertisements in the matchmaker. Clearly, as the number of advertisements in the matchmaker increases the time taken to process each query will also increase. To overcome this limitation, when an advertisement is published, we annotate all the ontology concepts in the matchmaker with the degree of match that they have with the concepts in each published advertisement. As a consequence the effort need to answer a query is reduced to little more than just a lookup. The rational behind our approach is that since the publishing of an advertisement is a one-time event, it makes sense to spend time to process the advertisement and store the partial results and speed up the query processing time, which may occur many times and also the query response time is critical. First we will briefly discuss the matching algorithm, then our enhancements in the publish and the query phase.

4.1 Matching Algorithm

The matching algorithm we used in our matchmaker is based on the algorithm presented in [2]. The algorithm defines a more flexible matching mechanism based on the OWL's subsumption mechanism. When a request is submitted, the algorithm finds an appropriate service by first matching the outputs of the request against the outputs of the published advertisements, and then, if any advertisement is matched after the output phase, the inputs of the request are matched against the inputs of the advertisements matched during the output phase.

In the matching algorithm, the *degree of match* between two outputs or two inputs depends on the match between the concepts that represents by them. The matching between the concepts is not syntactic, but it is based on the relation between these concepts in their OWL ontologies. For example consider an advertisement, of a vehicle selling service, whose output is specified as Vehicle and a request whose output is specified as Car. Although there is no exact match between the output of the request and the advertisement, given an ontology fragment as show in Fig 4, the matching algorithm recognizes a match because Vehicle subsumes Car.

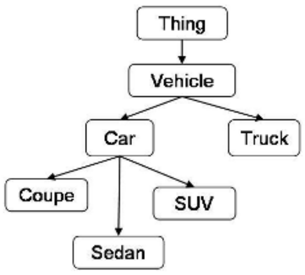


Fig. 4. Vehicle Ontology

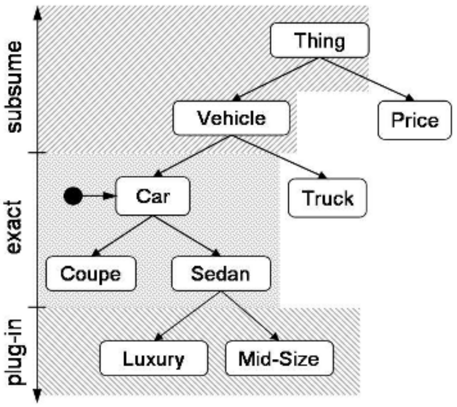


Fig. 5. Advertisement Propagation

The matching algorithm recognizes four degrees of match between two concepts. Let us assume OutR represents the concepts of an output of a request, and OutA that of an advertisement. The degree of match between OutR and OutA is as follows.

exact: If OutR and OutA are same or if OutR is an immediate subclass of OutA. For example given the ontology fragment like Fig 4, the degree of match between a request whose output is `Sedan` and an advertisement whose output is `Car` is exact.

plug in: If OutA subsumes OutR, then OutA is assumed to encompass OutR or in other words OutA can be plugged instead of OutR. For example we can assume a service selling Vehicle would also sell SUVs. However this match is inferior to the *exact* match because there is no guarantee that a Vehicle seller will sell every type of Vehicle.

subsume: If OutR subsumes OutA, then the provider may or may not completely satisfy the requester. Hence this match is inferior than the *plug in* match.

fail: A match is a *fail* if there is no subsumption relation between OutA and OutR.

4.2 Publishing Phase

Publishing of a Web service is not a time critical task; therefore we attempt to exploit this time to pre-compute the degree of match between the advertisement and possible requests. To perform this pre-computation, the matchmaker maintains a taxonomy that represents the subsumption relationships between all the concepts in the ontologies that it loaded. Each concept in this taxonomy is annotated with a two lists *output_node_ information* and *input_node_information* that specify to what degree any request pointing to that concept would match the advertisement. For example, output_node_information is represented as the following vector [<Adv1,exact>, <Adv2,subsume> , …], where AdvX points to the advertisement and "subsume" specify the degree of match. The advantage of the pre-computation is that at query time the matchmaker can extract the correct value with just a lookup with no need of inference.

More in details, at publishing time, the matchmaker loads the ontologies that are used by the advertisement's inputs and outputs and updates its taxonomy. Then, for each output in the advertisement, the matchmaker performs the following steps.

- The matchmaker locates the node corresponding to the concept, which represents the output, in the hierarchical structure let us call this node *curr_node*. The degree of match between the curr_node's concept and the output of the advertisement is *exact*, so the matchmaker updates the *output_node_ information*. Let us assume Fig 5 represents the hierarchical structure maintained by the matchmaker and let an output of an advertisement Adv_1 be 'Car'. The matchmaker updates the *output_node_information* of the 'Car' node that it matches Adv_1 *exactly*.
- The matchmaker updates the *output_node_ information* of all the nodes that are immediate child of the *curr_node* that the published advertisement matches them *exactly*. Because the algorithm states that the degree of match between output and the concepts immediate subclass are also *exact*. Following our example the matchmaker will updates the *output_node_ information* of the 'Coupe' node and the 'Sedan' node that it matches the advertisement Adv_1 exactly.
- The matchmaker updates the *output_node_ information* of all the parents of the *curr_node* that the degree of match between the nodes and the published advertisement is *subsume*. Following our example, we can see that the degree of match between the Adv_1's output concept 'Car' and the parent nodes of the *curr_node* 'Thing' and 'Vehicle' are *subsume*.
- Similarly the matchmaker updates the *output_node_ information* of all the child nodes of *curr_node* that the degree of match between the node and the published

advertisement is *plug-in*. Following our example, we can see that the degree of match the advertisement's output and the child nodes of *curr_node* 'Luxury' and 'Mid-Size' is *plug-in*.

Similar steps are followed, for each input of the published advertisement the matchmaker updates the *input_node_information* of the appropriate nodes.

As we can observe, we are performing most of the work required by the matching algorithm during the publishing phase itself, thereby spending a considerable amount of time in this phase. Nevertheless, we can show that the time spend during this phase, does not depend linearly on the number of concepts present in the data structure but in the order of *log (number of concepts)* in present in the tree structure, and hence showing that our implementation is scalable.

Since we use hierarchical data structure, the time required to insert a node will be in the order of $\log_d N$, where d is the degree of tree. Similarly time required to traverse between any two nodes in a particular branch will also be in the order of $\log_d N$. The time required for publishing an advertisement will be equal to the time required for classification of the ontologies used by inputs and outputs of the advertisement, plus the time required to update the hierarchical structure with the newly added concepts, plus the time required to propagate information about the newly added advertisement to the hierarchical structure. And in a best case scenario, when no ontology needs to be loaded, the publishing time will be time required for updating and propagating.

$$\text{Time}_{\text{publish}} = \text{Time}_{\text{Classification}} + \text{Time}_{\text{Update}} + \text{Time}_{\text{propagate}} \qquad (1)$$

The time required by Racer for classifying neither directly depended on the number of concepts nor the number of advertisements present in the matchmaker. The time required by the other two operations, update and propagate, will be in the order of ($\log_d N$). Hence the publishing time does not linearly depend on the number of concepts or the advertisements present in the matchmaker.

4.3 Querying Phase

Since most of the matching information is pre-computed at the publishing phase, the matchmaker's query phase is reduced to simple lookups in the hierarchical data structure. We also save time by not allowing a query to load ontologies. Although loading ontologies required by the query appears to be a good idea, we do not allow it for the following three reasons: first, the loading of an ontology is an expensive process, furthermore the number of ontologies to load is in principle unbounded. Second, if the request requires the loading of a new ontology, it is very likely that the new concepts will have no relation with the concepts that are already present in the matchmaker, therefore the matching process would fail anyway. Third, the ontologies loaded by the query may be used only one time, and over time we may result is storing information about lot of unused concepts. Note that the decision of not loading ontologies at query time introduces incompleteness in the matching process: it is possible that the requested ontology bares some relations with the loaded ontologies, therefore the matching process may succeed.

Still, the likelihood of this event is small, and the cost of loading ontologies so big that we opted for not loading them.

When the matchmaker receives a query it retrieves all *output_node_informations,* the sets of advertisements and its degree of match with the concept, of all the nodes corresponding to the outputs of the request. For example, if the outputs of the request are ‘Car’ and ‘Price’, the matchmaker fetches the *output_node_informations* of car ONI_1 and of price ONI_2. The matchmaker then finds the advertisements that are common between the sets of advertisements retrieved, i.e. $ONI_1 \cap ONI_2$. If no intersection is found then the query fails. If common advertisements are found say ADVSo, they are selected for further processing.

The matchmaker performs a lookup operation and fetches all the *input_node_informations*, the sets of advertisements and degree of match with the concept, of all the nodes corresponding to the inputs of the request. The matchmaker keeps only the input_node_information of the advertisements that were selected during the output processing phase, other advertisements are discarded. For example let IN_1, IN_2, and IN3 be the input_node_information, then only input_node_information of advertisements $ADVSo \cap IN_1 \cap IN_2 \cap IN_3$ are kept. This input_node_information and match level of each output is used to score the advertisements that were selected during the output processing phase, i.e. ADVSo.

We can see that the time required for processing a query does not depend on the number of advertisements published in the matchmaker. As we also see the querying phase involves lookups and intersections between the selected advertisements. In our implementation lookups can performed in constant time. Hence time to process a query depends on the time to perform intersections between the selected advertisements.

$$\text{Time}_{\text{query}} = (\text{Num}_{\text{out}}+\text{Num}_{\text{in}})*(\text{Time}_{\text{Lookup}} + \text{Time}_{\text{Intersection}}) \quad (2)$$

Table 1. Publishing Time without loading ontologies

	Time ms	Standard Deviation
UDDI	163.98	86.17
OWL-S/UDDI	1050.77	167.96

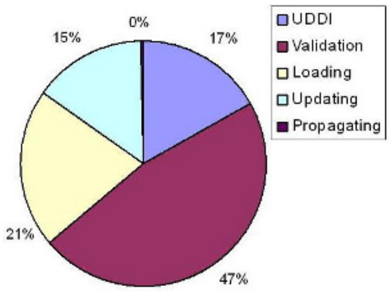


Fig. 6. Time distribution during publishing an advertisement

Where the Num_{out} and Num_{in} are the number of inputs and outputs of an advertisement, $\text{Time}_{\text{Lookup}}$ is the time required to extract information about an input or an output, and $\text{Time}_{\text{Intersection}}$ is the time to compute intersection between the lists extracted during the lookup time. We can see that the computation required for the querying process does not depend on the number of advertisements, and therefore it is scalable.

5 Preliminary Experimental Results

We conducted some preliminary evaluation comparing the performances of our OWL-S/UDDI registry and a UDDI registry, to show that adding an OWL-S matchmaker component does not hinder the performance and scalability of a UDDI registry. We extended jUDDI [14] an open source UDDI registry with the OWL-S matchmaking component. We used RACER [4] as to perform OWL inferences. In our experiments, we measured the processing time of an advertisement by calculating the difference between the time the UDDI registry receives an advertisement and the time the result is delivered, to eliminate the network latency time.

5.1 Performance – Publishing Time

In our first experiment we compared the time take to publish an advertisement in an OWL-S/UDDI registry and in a UDDI registry. We assumed that the ontologies required by the inputs and outputs of the advertisements are already present in the OWL-S/UDDI registry. The advertisements may have different inputs and outputs but they are present in one ontology file, hence the ontology has to be loaded only once, however our registry still have to load 50 advertisements. Table 1 shows the average time taken to publish 50 advertisements in a UDDI registry and an OWL-S/UDDI registry. We can see that the OWL-S/UDDI registry spends around 6-7 times more time, since publishing it a one-time event we are not concerned about the time taken.

However, we took a closer look at the time taken to publish an advertisement by the OWL-S/UDDI registry. Fig 6 shows the time spent in different phases of the publishing an advertisement. Following are the 5 phases in the publishing process:

- UDDI - time required by the UDDI component to process an advertisement.
- Validation - time required by Racer to validate the advertisement.
- Loading - time required by Racer to load the advertisement
- Updating - time required to extract the ontology tree from Racer.
- Propagating - time required to propagate the input/output information.

As we can see, most of the time is spent in loading and validating ontology (around 70%) when compared to the matchmaking operations.

5.2 Performance – Ontology Loading

In the second experiment, we analyzed the performance of our registry when we load advertisements that required loading new ontology and hence significantly updating the taxonomy maintained by the matchmaking component. We published 50 advertisements that use different ontologies to describe their inputs and outputs, in our OWL-S/UDDI registry and measured the time taken to publish each advertisement. Each of these advertisements has three inputs and one output and requires loading an ontology containing 30 concepts.

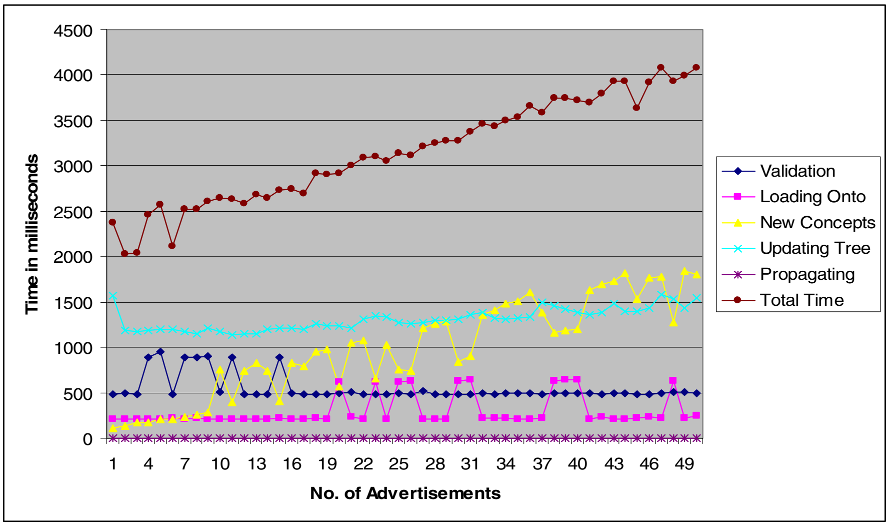


Fig. 7. Publishing time for advertisements that requires to load new ontologies

In Fig.7, we can see that the time take to publish an advertisement increases linearly with the number of advertisements, and we can also see that this linear increase is contributed by 'new-concept'. This linear increase of 'new-concept' is attributed to a limitation of the Racer system. Whenever we load a new ontology into Racer we have determine if we need to update the taxonomy maintained by the matchmaker, if so, what concepts should be updated. The Racer system does not provide any direct means to give this information. Hence we need to find out this information through a series of interactions. The new-concept in Fig 7 represents the time required to perform this operation. We can substantially reduce the time required for publishing if either Racer can provide the information directly or if we could have direct access to Racer and we maintain the taxonomy inside Racer itself. We can see that if ignore the time taken by 'new-concept', the resulting graph would not have such drastic increase in the publishing time, concurring to our discussion in Section 4.2.

Table 2. Query processing time

	Time in ms	Standard Deviation
OWL-S/UDDI	1.306	.54

5.3 Performance – Querying Time

In our final experiment, we calculated the time required to process a query. The queries we used do not load new ontologies into the matchmaker, they use the ontologies that are already present in the matchmaker. We used 50 queries each with three inputs and one output. Table 2 shows the average time required to process these queries. The small standard deviation shows that the time required to process the queries is almost constant, consistent with our discussions in Section 4.3

6 Literature Review

In the last few years, discovery of OWL-S Web services has been a very active field of research in the context of the Semantic Web. A comprehensive review of the algorithms that have been proposed is beyond the scope of this paper, but a few of these projects have concentrated on enhancing the UDDI registry with OWL-based semantic information or OWL-S descriptions. In this paper we will review these attempts.

An approach to semantic discovery in UDDI has been implemented as an extension of the NTT UDDI UBR[1] which is the public UDDI registry, maintained by the NTT, the Japanese telephone company [16]. An important contribution of this work is VOC (Voice of the Customer) analysis of the requirements of potential users of UDDI. The result of that analysis shows that the main concern of customers is the interoperability with the current UDDI API and system maintenance. Although our design decisions were not guided by this VOC analysis that was not available yet, our approach is consistent with these results because we were very careful not to break or overload the UDDI API, preferring instead to provide an extension to that API. What differs our approach from the work presented in [16] is the approach to discovery and the mapping to UDDI. Rather than providing the indexing of advertisements that we describe here, they provide a filtering mechanism that progressively reduces the set of advertisements that are potential candidates to match the request. The filtering mechanism used has its roots in the Larks [6] matchmaker. Larks is also the starting point of our work, and we believe the indexing described in this paper essentially accomplishes the pruning tasks that were performed by Larks, while exploiting the structure of the OWL ontologies. Nevertheless, a complete exploration of the tradeoffs between the two approaches is a matter of future research. The second difference is in the representation of Web services, whereas we use OWL-S, [16] relies on a semantic extension of WSDL that they name WSSP. Despite the superficial differences both approaches describe the semantic signature of the Web service and they ultimately have the same expressive power.

Another approach for a Semantic UDDI registry is presented in [18] is based on [1] and [2]. This work enhances the semantic search mechanism presented in [2] in couple of ways, first it extends the UDDI Inquiry API by enabling users to specify semantic inquires based on web services capabilities, secondly it enhances the matching algorithm with a planning functionality, which is capable of satisfying users requests by composing two or more service descriptions. Despite many similarities between our work and this work, the difference lies in implementation of the matching algorithm. While our work concentrates on providing an efficient implementation of the matching algorithm proposed in [2], the matching algorithm in this work seems

[1] See http://www.ntt.com/uddi/index-e.html for the NTT UDDI UBR and http://www.agent-net.com/refer.htm for details on the semantic matching engine.

to be a straight forward implementation of the algorithm proposed in [2]. Another work involving semantic UDDI is presented in [17], it presents a flexible mechanism to enhance the UDDI search mechanism, by integrating multiple external matching engines to support multiple service description languages. The primary focus of this work is to develop a mechanism to facilitate integration and co-ordination of multiple matching engines with UDDI. Although this work is orthogonal to our work, our matching engine could be easily integrated in this framework to provide matching service for service descriptions expressed in OWL-S.

Meteor-S [23] presents a framework for adding semantics directly to existing Web Services standards, like WSDL and UDDI. It allows users to semantically annotate their WSDL and UDDI descriptions of their web services with DAML and publish these descriptions in their enhanced UDDI. Their matching algorithm [24] extends the work present in [2] in two ways, first they extend the subsumption based matching mechanism by adding information retrieval techniques to find similarity between the concepts when it is not explicitly stated in the ontologies, and secondly they added a mechanism to match on preconditions and effects of service descriptions. From the literature review of Meteor-S, we speculate that our optimization technique presented in this paper could improve the efficiency of their matching process.

In this paper we make a strong case in favor of careful indexing of advertisements to speed up the matching process. A similar case is made by [19] who shows how the lack of appropriate indexing provides a matching process that is proportional to the number of advertisements and therefore not scalable in the long run. The difference between this paper and [19] is the indexing algorithm. While we rely on the structural properties of the matching algorithm and of the OWL ontologies, they define a Generalized Search Tree [20]. The efficiency trade-offs between the two approaches are a matter of empirical analysis that goes beyond the current paper. The other difference is that they consider the possibility that answers to queries can be the result of the composition of multiple advertisements. We do not consider this possibility because it can result in a combinatorial explosion of possible matching in which a query could be decomposed in many different ways to fit the existing services.

7 Conclusions and Future Work

In this paper we have described the importance of web service discovery and the shortcomings of the UDDI's discovery mechanism. We adapted a solution to use OWL-S in combination with UDDI, to take advantage of both these technologies. We believe such an architecture is very important in bring the effort of both Web Service and Semantic Web community together. We presented our OWL-S/UDDI matchmaker architecture and its extensions to perform capability search. We also conducted some preliminary experiments to show the scalability of our implementation.

We are extending the current work in multiple directions. The matching process that we are using so far is restricted to the inputs and outputs of the Service Profile, while the functional capabilities in OWL-S extend to Preconditions and Effects. This restriction was originally grounded on the lack of a condition language that combined with OWL, but the publication of the Semantic Web Rule Language (SWRL) [21] and the development of DRS [22]. We are currently working on an extension of the matching process to Preconditions and Effects in the context of OWL-S 1.1. The second limitation of this work is the lack of any matching on service parameters and service categories; we are currently extending our matching process to include them. In the context of this work, we are also attempting to integrate the matching of the type of service so that a requester may be able to express the type of service required explicitly rather than implicitly through input, output, preconditions and effects. The last development work that we are pursuing is rigorous testing with increasing advertisement and request load to evaluate the scalability of our algorithms.

The techniques proposed in this work provide algorithms for the efficient use of OWL-S ontologies in UDDI, but we believe it can be easily applied to any OWL ontology. In this sense, the algorithms provided in this paper may provide a valuable ground for an efficient and scalable implementation of the proposed semantic search in UDDI [8]. We are currently exploring the implementation of the algorithms proposed here in the context of a semantic extension of the JUDDI [14].

References

1. Paolucci et al: Importing the Semantic Web in UDDI. In Proceedings of Web Services, E-business and Semantic Web Workshop, 2002
2. Paolucci et al; Semantic Matching of Web Services Capabilities. In Proceedings of the 1st International Semantic Web Conference (ISWC2002)
3. Anupriya et al: DAML-S: Web Service Description for the Semantic Web. In Proceedings of The First International Semantic Web Conference (ISWC), 2002.
4. Volker Haarslev, Ralf Möller: RACER System Description. In Proceedings of International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, 2001, Siena, Italy.
5. Tim Berners-Lee and James Hendler and Ora Lassila: The Semantic Web. Scientific American ,volume 284, Number 5, pages 34-43, 2001
6. Katia Sycara et al : Larks, Dynamic matchmaking among Heterogeneous Software Agents in Cyberspace, AAMAS, 5, 173-203, 2002.
7. W3C: Extensible Markup Language (XML) 1.0 (Second Edition). http://www.w3.org/TR/2000/REC-xml-20001006,2000,
8. UDDI: The UDDI Technical White Paper, http://www.uddi.org, 2000
9. Rosetta Net, http://www.rosettanet.org, 2000
10. W3C: “SOAP Version 1.2, W3C Working Draft 17 December 2000”, http://www.w3.org/TR/2001/WD-soap12-part0-20011217/ , 2001
11. Erik Christensen et al: “Web Services Description Language (WSDL) 1.1”, http://www. w3.org/TR/2001/NOTE-wsdl-20010315, 2001
12. ISO/IEC 11578:1996: Information technology -- Open Systems Interconnection -- Remote Procedure Call. http://www.iso.ch/ , 2001.
13. Colgrave et al: Using WSDL in a UDDI Registry, Version 2.0., UDDI TC Note, 2003.
14. jUDDI: http://ws.apache.org/juddi/
15. W3C: Web Ontology Language. http://www.w3.org/2001/sw/WebOnt/
16. Kawamura et al: Public Deployment of Semantic Service Matchmaker with UDDI Business Registry . International Semantic Web Conference (ISWC2004)
17. Colgrave et al: External Matching in UDDI, In Proceedings of the International Conference on Web Services ICWS 2004.

18. Akkiraju et al: A Method For Semantically Enhancing the Service Discovery Capabilities of UDDI, Workshop on Information Integration on the Web IJCAI 2003.
19. Constantinescu et al: Efficient Matchmaking and Directory Services, International Conference on Web Intelligence (WI'03)
20. Hellerstein et al: Generalized search trees for database systems. In Proceeding of 21st International Conference on Very Large Databases, VLDB, pages 562-573, 1995
21. Horrocks et al: SWRL: A Semantic Web Rule Language Combining OWL and RuleML, http://www.w3.org/Submission/SWRL/
22. Drew McDermott: DRS: A Set of Conventions for Representing Logical Languages in RDF, http://www.cs.yale.edu/homes/dvm/daml/DRSguide.pdf
23. METEOR-S: http://lsdis.cs.uga.edu/Projects/METEOR-S/
24. Cardoso, J. and A. Sheth : Semantic e-Workflow Composition . Journal of Intelligent Information Systems (JIIS), 2003

A Semantic Approach for Designing E-Business Protocols*

Ashok U. Mallya and Munindar P. Singh

Department of Computer Science, North Carolina State University,
Raleigh, NC, USA 27695-7535
{aumallya, singh}@ncsu.edu

Abstract. Business processes involve interactions among autonomous partners. We propose that these interactions be specified modularly as protocols. Protocols can be published, enabling implementors to independently develop components that respect published protocols and yet serve diverse interests. A variety of business protocols would be needed to capture subtle business needs. We propose that the same kinds of conceptual abstractions be developed for protocols as for information models. Specifically, we consider (1) *refinement*: a subprotocol may satisfy the requirements of a superprotocol, but support additional properties; and (2) *aggregation*: a protocol may combine existing protocols. In support of the above, this paper develops a semantics of protocols and an operational characterization of them. This supports judgments about the potential subclass-superclass relations between protocols, which are a result of protocol refinement. It also enables protocol aggregation by splicing a protocol into another protocol.

1 Introduction

Modern e-business processes span multiple autonomous entities or business partners. Such processes therefore are based on a rich variety of interactions among software components that are independently designed and configured and which represent independent (sometimes mutually competitive) business interests. Web services provide a basis for realizing such processes by enabling businesses to interoperate in a standardized manner. This has led to interest in technologies such as coordination and process flows, distributed transactions of various flavors, and conversations. While these approaches have some benefits, they mostly take a centralized perspective, akin to workflow technologies, viewing a process as a series of tasks to be performed. This proves too tedious for reliable modeling and too rigid for enactment, which is the reason workflow technologies have been considered a failure in many practical settings. The present paper relates to all of these efforts, but concentrates on the semantical aspects of the interactions among business partners.

We propose a novel framework for thinking about processes. Simply put, *a process instantiates one or more business protocols* among designated parties. We define a protocol as a specification of a logically related set of interactions. A protocol specifies

* This research is supported by DARPA and by the National Science Foundation under grant DST-0139037.

only the key desired aspects of the interactive behavior; it leaves the details of a local implementation entirely up to those who implement the protocol.

Realistic business settings will need an endless variety of business processes. While some of these processes will be widely deployed, several will be customized to special application domains, industries, and circumstances. While the hard-coded systems of todays process management require a serious integration and configuration effort to accommodate change, we imagine that by employing well-specified, published protocols to compose processes, the various stake-holders can considerably simplify their integration and configuration efforts [1]. Given a set of protocols, they would only need to acquire implementations for the roles of those protocols. RosettaNet is already a step in this direction [2]. It defines over 100 protocols (called PIPs in their terminology). RosettaNet's protocols are limited to two-party implementations and are mostly two-step protocols.

We want general protocols to support flexibility, and specialized protocols to support efficiency, security, or risk management. For example, we can imagine a generic payment protocol as well as specializations of it such as payment by cash, credit card, checks, wire transfer, and so on. Each of these would differ in the steps that each participant takes. Moreover, the protocols only specify the interactions, not the local policies of the participating entities, such as that they don't take cash after sunset. Protocols enable such policies to be inserted but are not directly concerned with the policies. As long as we recognize that these are payment protocols, our top-level design goal, namely, to enable some form of payment, would be satisfied.

The most fundamental computer science approach for dealing with complexity is to enable reuse. Two of the most basic ideas for doing so are to build a specialization-generalization hierarchy and to aggregate components. The objective of this paper is develop notions akin to traditional subsumption and aggregation that are applicable to protocols. We develop two main classes of abstractions: *refinement* (like the subclass-superclass hierarchy) and *aggregation* (like the part-whole hierarchy). We develop a formal semantics to support the hierarchy and propose and algebra to facilitate reasoning about protocols.

Contribution. Traditional workflow technologies are quite rigid in that they allow very little variation from the specified sequence of steps. Hence, composition of new workflows, or creating variants of existing ones involve considerable effort. Our contribution is in developing a basis for easily comparing protocols and an algebra for aggregating them to create business processes. The algebra provides the underpinnings of refinement and aggregation abstractions for protocols. The algebra is a high-level abstraction that relates to real-world interaction protocols, and hence is easy for protocol designers to understand. We also demonstrate how the use of commitments allows reasoning about protocols that leads to richer interaction patterns from existing ones. Further, we outline how a hierarchy of protocols can be generated based on commitments.

2 Technical Motivation

As a running example, we consider a *purchase* interaction in which a customer wants to buy a book from an online bookstore. The bookstore obtains the customers' order for a book if the customer accepts the price quoted by the bookstore for that book. The book is then shipped to the customer, and the bookstore is paid for the book. The actual execution of the process, however could involve many different scenarios, which we shall describe shortly.

Commitments in E-Business Protocols. To talk about how e-business protocols can be aggregated or refined, we must represent not just the behaviors of the participants but also how the contractual relationships among the participants evolve over the course of an interaction. Doing so enables us to determine if the interactions are indeed compliant with the stated protocols. The contractual relationships of interest are naturally represented through *commitments*, which have gained importance in the field of multiagent systems [3]. Commitments capture the obligations of one party to another. For example, the customer's agreement to pay the price for the book after it is delivered is a commitment that the customer has towards the bookstore. Commitments lend coherence to the interactions because they enable agents to plan based on the actions of others. In principle, violations of commitments can be detected and, with the right social relationships, commitments can be enforced. Enforceability of contracts is necessary when the participants are autonomous and heterogeneous [4].

Why Formal Semantics? As explained above, the objective of this paper is develop notions akin to traditional subsumption and aggregation that are applicable to protocols. Doing so presupposes that we have a crisp semantics and can reason formally about protocols. Accordingly, our task is to develop a semantics that facilitates flexible actions.

3 Technical Framework

We represent protocols as transition systems similar in spirit to finite state machines. These protocols generate computations or *runs*, which are sequences of *states* that a valid protocol computation (execution) goes through. State changes are caused by *actions* that the participants perform. We devise a hierarchical classification based on the runs generated by protocols. This classification forms the basis of our work. The remainder of this section introduces commitments, discusses some scenarios from our running example, and then defines the basic technical concepts of propositions, states, actions, runs, and protocols.

3.1 Commitments

Much of our technical development is based on established concepts of distributed computing and temporal logic. Since commitments might be unfamiliar to some readers, we introduce them first, so we can discuss our running example in more depth.

Definition 1. *A commitment $\mathsf{C}(x, y, p)$ denotes that the agent x is responsible to the agent y for bringing about the condition p.*

Here x is called the *debtor*, y the *creditor*, and p the *condition* of the commitment. The condition is expressed in a suitable formal language.

Commitments can also be *conditional*, denoted by $\mathsf{CC}(x, y, p, q)$, meaning that x is committed to y to bring about p if q holds. For example, the conditional commitment $\mathsf{CC}(c, b, pay_p, goods_g)$ means that the customer c is committed to pay the bookstore b an amount p if the bookstore delivers the book g to the customer.

Commitment Operations. Commitments are created, satisfied, and transformed in certain ways. The following operations are conventionally defined for commitments [5].

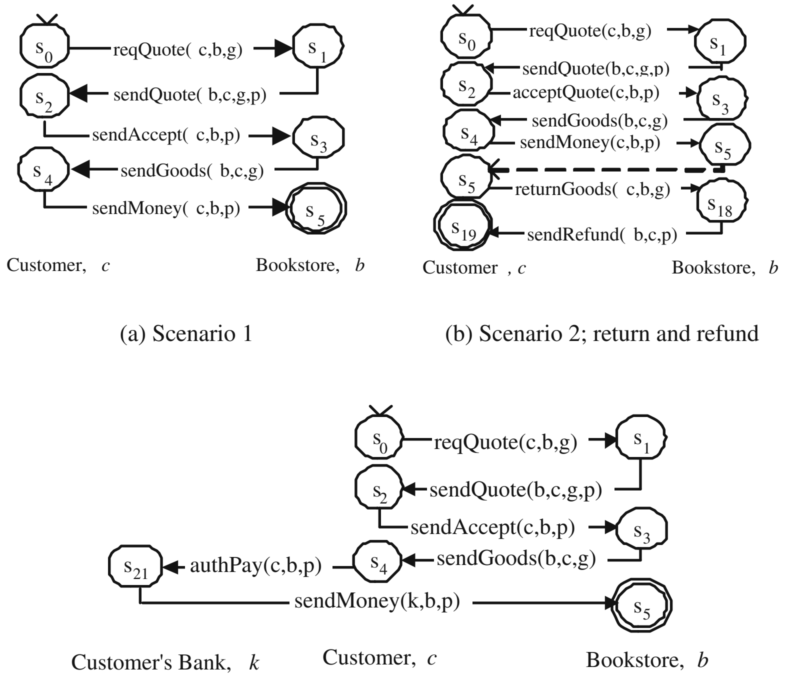
1. CREATE*(x, c)* establishes the commitment *c* in the system. This can only be performed by *c*'s debtor *x*.
2. CANCEL*(x, c)* cancels the commitment *c*. This can only be performed by *c*'s debtor *x*. Generally, cancellation is compensated by making another commitment.
3. RELEASE*(y, c)* releases *c*'s debtor *x* from commitment *c*. This only can be performed by the creditor *y*.
4. ASSIGN*(y, z, c)* replaces *y* with *z* as *c*'s creditor.
5. DELEGATE*(x, z, c)* replaces *x* with *z* as the *c*'s debtor.
6. DISCHARGE*(x,c)* *c*'s debtor *x* fulfills the commitment.

A commitment is said to be *active* if it has been created, but not yet discharged.

3.2 Example Scenarios

We identify four distinct, but related, scenarios that can arise during the above book purchase interaction. Each of these scenarios requires a different amount of effort from the participants in terms of protocol execution, planning, and coordination. Both agents would benefit from being able to compare scenarios to choose the one that best serves their interests. These scenarios are shown in Figures 1(a), 1(b), 1(c), and 3. *Customer* refers to the customer's agent and *Bookstore* refers to the bookstore's agent. Ellipses represent states, named s_i. Solid arrows are labeled by the messages that are passed between the participating agents. These messages correspond to actions that the agents take.

1. Normally, the customer would ask the bookstore for a price quote on the book it wishes to buy, and upon receiving a quote from the bookstore, would accept the bookstore's offer. The bookstore would then send the book, after which the customer would send the payment. This is modeled after the NetBill protocol [6]. Figure 1(a) shows this interaction.
2. The bookstore might be willing to give a refund if the customer returns the book for some reason. This scenario is similar to the scenario described above till the book is delivered to the customer. However, this scenario is longer, since the customer returns the book, and terminates only when the bookstore sends the refund to the customer. Figure 1(b) shows this interaction.
3. The customer might delegate the payment to a third-party, e.g., a bank. Such a situation is not very different from using a credit card to pay for goods, and is shown in Figure 1(c). The customer, after accepting the bookstores price quote and later receiving the book, sends a message to both the bookstore and the bank (although only one arrow is shown in the figure, between states s_4 and s_{21}) indicating that the bank will honor the customers commitment to pay.



(a) Scenario 1

(b) Scenario 2; return and refund

(c) Scenario 3; pay via bank

Fig. 1. Three scenarios of the purchase example

4. The bookstore might have to negotiate with and contract out the actual shipping to a shipper. This might happen, for example, if the customer wants insured shipping, and the bookstore's existing shipper does not insure goods. Here, the bookstore interacts with the shipper, and gets the books delivered to the customer. The shipper is then paid by the bookstore, but only after the book has been delivered to the customer. To complicate matters, the customer pays via its bank in this scenario, just like in the previous one. This scenario is shown in Figure 3 and discussed in detail in Section 5.

Table 2 explains the meanings of the states that the first scenario runs through. Table 4 shows the meanings of the messages passed. In this table, c represents the customer, b represents the bookstore, g represents the book that the customer is interested in buying, and k represents the customer's bank. The *delegate* message relates to corresponding operation that can be performed on commitments. Also,the notation $[gv]$ used in Figures 2(b) and 3 is short for "the good g delivered to v."

3.3 Propositions

Propositions capture facts about what conditions hold, what commitments have been made, and whether these commitments have been fulfilled. *Domain propositions* are protocol-specific atomic propositions such as $goods_{b,c,g}$, which is used in the purchase example to mean b has delivered g to c. Table 1 describes some domain propositions in the purchase example, where b represents the bookstore, c, the customer, and g the goods purchased. *Commitment propositions* represent active commitments. An example of such a proposition is $\mathsf{C}_{c,b,pay_{c,b,p}}$, which means that c is committed to b to pay the amount p. *Commitment operation propositions* represent facts about which commitment operations were performed.

The propositions used in a protocol are assumed to be understood by parties involved in the protocol.The set of propositions is denoted by $\mathbb{P}$. This set includes domain, commitment, and commitment operation propositions.

Table 1. Meanings of propositions in the purchase example

Proposition	Meaning
$reqQuote_{c,b,g}$	c has requested a quote for g from b.
$quote_{b,c,g,p}$	b quotes to c price p for g, i.e., b will deliver if c commits to pay upon delivery. This is represented by $\mathsf{CC}(b, c, goods_{b,c,g}, acceptQuote_{c,b,g,p})$.
$acceptQuote_{c,b,g,p}$	c has accepted the price p that b quoted for g,i.e, c commits to pay if the goods are delivered. This is represented by $\mathsf{CC}(c, b, pay_{c,b,p}, goods_{b,c,g})$
$goods_{b,c,g}$	g has been delivered to c by b.
$pay_{c,b,p}$	The amount p has been paid to b by c.
$return_{c,b,g}$	g has been returned to b by c.
$refund_{b,c,p}$	The amount p has been refunded to c by b.

3.4 States

Definition 2. *A state is an assignment of truth values to members of* $\mathbb{P}$.

Equivalently, a state is labeled with the set of propositions which hold at that state. For example, state s_1 of the purchase example is labeled by the set $\{reqQuote_{c,b,g}\}$ and state s_2 by $\{quote_{b,c,g,p}\}$. We denote the label of a state s by $[s]$. Table 3 gives the labels that are assigned to states in the purchase example. The set of states is denoted by $\mathbb{S}$. We include in this set a unique start state s_ϕ, which is labeled by the set $\{\mathtt{true}\}$. In the purchase examples, $s_0 = s_\phi$.

3.5 Actions

Agents perform actions to bring about changes in the world. In our framework, actions of the agents are modeled by messages that the agents send to others. An agent's action affects the state of a protocol in which it participates. For example, a $sendMoney(c, b, p)$ message sent during the execution of Scenario 1 of the purchase example models the action of the customer c paying the bookstore b an amount p. Messages model actions just like in the real world, where, for example, filling a form and submitting it over the web can cause a transfer of funds.

The set of actions is denoted by $\mathbb{A}$. The meanings of messages used in our purchase example are given in Table 4.

Table 2. Meanings of some states in the purchase example

State	Meaning
s_1	Customer has asked the bookstore the price of the goods. No commitments made.
s_2	Bookstore has quoted a price for the said goods. The bookstore is now willing to send the goods if the customer promises to pay for them
s_3	Customer has agreed to the bookstores price. The customer is willing to pay the price if the books are delivered.
s_4	Bookstore has delivered the book.
s_5	Customer has paid for the book.

Table 3. State labels in the purchase example

State	Associated Label
s_0	$\{\mathtt{true}\}$
s_1	$\{\mathit{reqQuote}_{c,b,g}\}$
s_2	$\{\mathit{quote}_{b,c,g,p}\}$
s_3	$\{\mathsf{C}_{b,c,\mathit{goods}_{b,c,g}}, \mathsf{CC}_{c,b,\mathit{pay}_{c,b,p},\mathit{goods}_{b,c,g}}\}$
s_4	$\{\mathit{goods}_{b,c,g}, \mathsf{C}_{c,b,\mathit{pay}_{c,b,p}}\}$
s_5	$\{\mathit{goods}_{b,c,g}, \mathit{pay}_{c,b,p}\}$
s_{21}	$\{\mathit{goods}_{b,c,g}, \mathsf{C}_{k,b,\mathit{pay}_{k,b,p}}\}$
s_{17}	$\{\mathit{goods}_{b,c,g}, \mathit{pay}_{c,b,p}\}$
s_{18}	$\{\mathit{goods}_{b,c,g}, \mathit{return}_{c,b,g}, \mathsf{C}_{b,c,\mathit{refund}_{b,c,p}}\}$
s_{19}	$\{\mathit{goods}_{b,c,g}, \mathit{return}_{c,b,g}, \mathit{refund}_{b,c,p}\}$

Table 4. Meanings of messages used in the purchase example

Message	Meaning
$\mathit{reqQuote}(c, b, g)$	c asks b what the price of g is.
$\mathit{sendQuote}(b, c, g, p)$	b quotes price p to the c, for g.
$\mathit{sendAccept}(c, b, g, p)$	c accepts the price p quoted by b for g. c is now committed to pay if the book is sent to it.
$\mathit{sendGoods}(b, c, g)$	b sends g to c.
$\mathit{sendMoney}(c, b, p)$	c sends the money p to b.
$\mathit{delegate}(c, k, \mathsf{C})$	c delegates the commitment C to k.
$\mathit{returnGoods}(c, b, g)$	c returns g to b.
$\mathit{sendRefund}(b, c, p)$	b refunds the money p to c.
$\mathit{authPay}(c, b, p)$	c authorizes its bank to pay the amount p to b. Essentially c delegates $\mathsf{C}(c, b, p)$ to k.

3.6 Runs

Definition 3. *A run is a sequence of states* $\langle s_0 \ldots s_i \ldots\rangle$.

In this paper, we consider only nonempty runs. That is, a run must contain an initial state. The operator $\prec_\tau$ orders states temporally with respect to a run τ, so that $s_i \prec_\tau s_j$ implies that s_i occurs before s_j in the run τ.

3.7 Protocols

Definition 4. *A protocol is a tuple,* $\langle \mathbb{A}, \mathbb{S}, s_0, \Delta, \mathbb{F}, \mathbb{R}\rangle$ *where*

– $\mathbb{A}$ *is a set of actions,*
– $\mathbb{S}$ *is a set of states,*
– s_0 *is the initial state,* $s_j \in \mathbb{S}$,

– *Δ is a set of transitions, $\Delta \subseteq \mathbb{S} \times \mathbb{A} \times \mathbb{S}$,*
– *$\mathbb{F}$ is a set of final states, $\mathbb{F} \subseteq \mathbb{S}$, and*
– *$\mathbb{R}$ is a set of roles (or participants) in the protocol.*

Δ contains transitions of the form $\langle s_i, a, s_j \rangle$, where $s_i, s_j \in \mathbb{S}$ and $a \in \mathbb{A}$. Here s_i is the source of the transition and s_j its destination. Such a transition advances a computation that is in state s_i to state s_j based on an action a.

In other words, a run can be generated from a protocol by the successive concatenation of transitions beginning from the initial state of the protocol. The concatenation of a transition to a run appends the destination of transition to the run if the source of the transition matches the last state of the run. Consequently, a run $\langle s_0 s_1 s_2 \ldots s_n \rangle$ can be generated by a protocol whose initial state is s_0, and whose transition set contains the elements $\langle s_0, \_, s_1 \rangle, \langle s_1, \_, s_2 \rangle$ and so on till $\langle \_, \_, s_n \rangle$. $[\![M]\!]$ denotes the set of all runs that a protocol M can generate.

4 Reasoning About Protocols

States form the fundamental components of runs, and are labeled by sets of propositions. Any comparison of states, therefore, must be based on comparing propositions. Section 4.1 introduces two similarity functions for states, both based on commitment propositions, Section 4.2 shows how these help relate different runs, and Section 4.3 uses comparisons of commitment-operation based propositions to relate different protocols.

4.1 Similarity of States

A *state-similarity function* f is a mapping from a state to a set of states, i.e., $f : \mathbb{S} \mapsto 2^{\mathbb{S}}$.

Definition 5. *A state s_i is similar to a state s_j under the state-similarity function f if and only if $s_j \in f(s_i)$.*

State-similarity under the state-similarity function f is denoted by the operator $[f\rangle\!\rangle$. That is, $s_i [f\rangle\!\rangle s_j \iff s_j \in f(s_i)$.

Identity State-Similarity. We define ι as the identity state-similarity function.

$$\iota(s_i) = \{s_j | [s_i] = [s_j]\} \tag{1}$$

That is, $s_i [\iota\rangle\!\rangle s_j$ if and only if s_i and s_j are labeled by same set of propositions. The operator $[\iota\rangle\!\rangle$ is reflexive, symmetric, and transitive.

Creditor State-Similarity. As another state-similarity function, consider σ.

$$\sigma(s_i) = \{s_j | s_j \text{ can be reached by finite number of } \textit{delegate}(\cdot,\cdot,\cdot) \text{ actions from } s_i\} \tag{2}$$

The function σ treats a state s_i as being similar to a state s_j if in the two states all the participants of the protocol have the same commitments being made towards them, regardless of which participant makes it. Here s_j can be thought of as the last state of a run generated by a protocol that has s_i as its initial state, and allows only *delegate*$(\cdot,\cdot,\cdot)$ actions in its transition set. States s_4 and s_{21} are similar under σ because they have active commitments that differ only in their creditors. The operator $[\sigma\rangle$ *is reflexive, symmetric, and transitive*.

4.2 Subsumption of Runs

Let $[\![f\rangle$ denote a *subsumption* operator over runs. The operator $[\![f\rangle$ is an order-preserving mapping from one run to another, and depends on the function f.

Definition 6. *A run τ_j subsumes a run τ_i under function f if and only if, for every state s_i that occurs in τ_i, there occurs a state s_j in τ_j that is similar under f, and s_j has the same temporal order relative to other states in τ_j as s_i does with states τ_i.*

$$\begin{aligned}\tau_j[\![f\rangle\tau_i \iff \forall s_i \in \tau_i, \exists s_j \in \tau_j : s_j \in f(s_i) \text{ and } \forall s_i' \in \tau_i, \exists s_j' \in \tau_j \\ : s_j' \in f(s_i') \Rightarrow (s_i \preceq_{\tau_i} s_i' \Rightarrow s_j \preceq_{\tau_j} s_j')\end{aligned} \quad (3)$$

Longer runs subsume shorter ones, provided they have similar states and in the same order.

Let τ_1, τ_2, and τ_3 be the runs that represent execution of scenarios 1, 2, and 3 of the purchase example respectively, as shown in Figures 1(a), 1(b), and 1(c).

Under ι, run τ_2 subsumes run τ_1, since every state in τ_1 occurs in τ_2 in the same position relative to other states. The converse, however, is not true because τ_2 contains state s_{18} and s_{19}. It is easy to see that $[\![\iota\rangle$ *is reflexive and transitive*.

Under creditor state-similarity function σ, τ_2 subsumes τ_1, because all states in τ_1 have a corresponding σ-similar state that occurs in τ_2, and in the same order. Clearly, $[\![\sigma\rangle$ *is reflexive and transitive*.

4.3 Subsumption of Protocols

When enacting processes using protocols, a protocol that generates only short runs is preferable over a protocol that generates longer runs since short runs speed up the protocol execution. At the same time, a protocol that allows many runs is better than one that allows a few runs, since the many-run protocol affords more choice and flexibility in its execution to the participants. We now develop some results about subsumption of protocols and demonstrate them with examples.

Every protocol M is considered to belong to a *frame* with enough propositions in it to label all states that can occur in the runs generated by $[\![M]\!]$. Frames serve as a common ontology for the propositions used by different protocols. They provide an upper bound on the universe of discourse of a protocol.

Definition 7. *A protocol M_j subsumes a protocol M_i under the function f if and only if, for every run τ_i that M_i can generate, M_j can generate a run τ_j that is subsumed by τ_i under f.*

$$M_j [\![f]\!\rangle M_i \iff \forall \tau_i \in [\![M_i]\!]\ \exists \tau_j \in [\![M_j]\!] : \tau_i [\![f]\!\rangle \tau_j \quad (4)$$

If M_j is a protocol that allows numerous runs and M_i is a protocol that allows only a few runs, then M_j subsumes M_i as long as each of M_j's runs is subsumed by one of M_i's runs. Since long runs subsume shorter ones, protocols that specify a few short runs subsume protocols that specify a large number of long runs. This follows from our intuition that fewer constraints make for more flexible protocols.

The protocol-subsumption relation $[\![\iota]\!\rangle$ *is reflexive and transitive* due to the properties of the ι function.

5 Designing Protocols via Aggregation

Protocols can be developed by aggregating smaller protocols to implement stages. For example, the purchase protocol, at a high level of abstraction, has an initial *negotiation* stage, followed by a *shipment* stage, and finally a *payment* stage. In the purchase scenario shown in Figure 1(a), states s_0, s_1, s_2, and s_3 belong to the negotiation stage, states s_3 and s_4 belong to the shipment stage, and states s_4 and s_5 belong to the payment stage. For simplicity, let us adhere to the negotiation-shipment- payment order even though more flexible protocols might allow the payment to precede the shipping. This three-stage protocol can be refined, for example, by substituting, or *splicing into* the purchase protocol, any of the shipping protocols available. One can ship via regular mail or use return-receipt mail. The splicing works because the purchase protocol is specified as an interface, and the shipping protocol adheres to the interface. For example, the shipping protocol in Figure 2(b) can be spliced into *Purchase* as shown in Figure 3. Similarly, the payment protocol shown in Figure 2(a) can be substituted for the payment stage. Further, the simple Purchase protocol subsumes the resultant protocol. One important observation to make is that a protocol that is spliced into another might itself be spliced

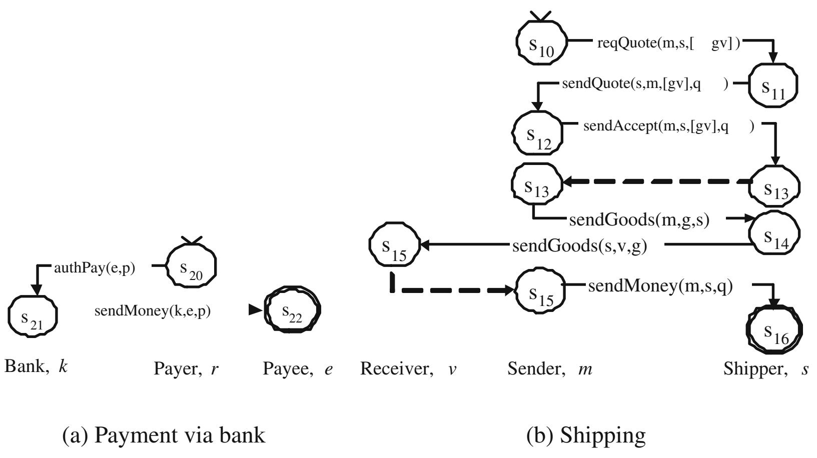


Fig. 2. Two Example Protocols

by the second protocol. In our example above, the Shipping protocol splices the Purchase protocol. However, the shipper is paid by the bookstore only after the customer has paid the bookstore. Therefore, the Shipment protocol has essentially be spliced in between its stages s_{14} and s_{15}.

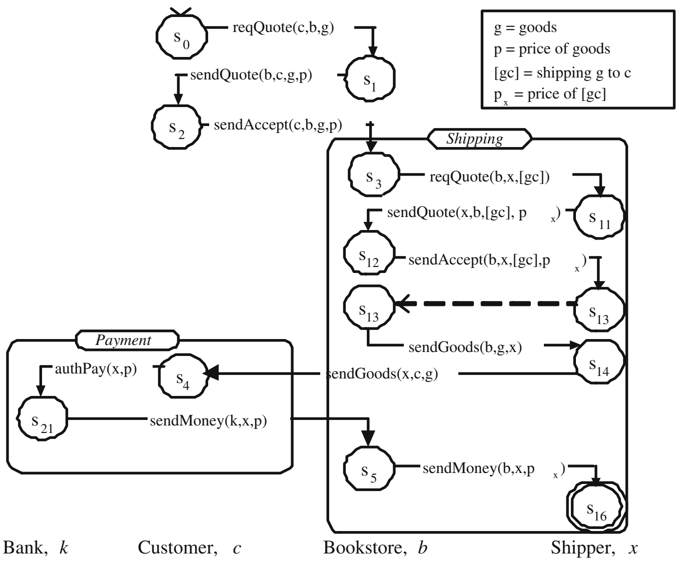


Fig. 3. Refinement of purchase by splicing in shipping and payment

Enabling Splicing. In realistic settings, e-business protocols will be refined by splicing to enable the participants' existing processes to interoperate seamlessly. As a guide, the following are to be borne in mind when designing a protocol.

- A protocol can be spliced into by another if the contractual relationships between the participants in each protocol are preserved.
- In most cases a refined protocol differs from the original only in terms of the creditor or the debtor of commitments that are made in the protocol. State similarity functions therefore need to compare states respecting the commitments made by or to the participants.
- The specification of a protocol should provide clues about states between which another protocol can be spliced in. For example, the purchase protocol specification identifies s_3 and s_4 as the states between which the goods should be delivered.

Commitment-based protocol design helps reason about legal and illegal splicing easily since commitments have a clear operational semantics across domains. A more detailed account of the protocol algebra can be found elsewhere [7].

6 Discussion

We introduced a technical approach for modeling protocols that provides a natural basis for principled methodologies for designing custom protocols for e-business. The main contributions lie in the formalization of protocol specialization and aggregation. This can further be employed to perform subsumption reasoning and to carry out more interesting operations on protocols, such as splicing.

6.1 Literature

Yolum and Singh [8] and Fornara and Colombetti [9] highlight the benefits of a commitment-based approach to interaction protocol design. Johnson *et al.* [10] develop a scheme for identifying when two commitment-based protocols are equivalent. Their scheme, however, is simplistic, classifying protocols based solely on their syntactic structure. Our work provides stronger results from an application point of view and relates better to the Web Services approach.

The MIT Process Handbook [11] is a project that aims to create a hierarchy of commonly used business processes. Based on this hierarchy, Grosof and Poon [12] develop a system to represent and execute business rules.

Fu and colleagues [13] have deveoped formal results about the computational demands of using conversation protocols for web service interactions. Our work relates to methodologies for design of flexible commitment protocols while they concentrate on the execution of state-based protocols.

The Web Services related standards for process composition and interoperability [14], such as the Web Services Choreography Interface (WSCI) are lower level abstractions than ours since they specify flows in terms of message sequences.

RosettaNet [2] (explained in Section 1) and ebXML [15] are two major industry efforts for business protocols. Some of RosettaNet's components are gradually shifting over to using ebXML, e.g., for messaging formats. ebXML's Business Process Specification Schema (BPSS) describes partner roles and the documents they would exchange. Neither approach, however, utilizes semantic abstractions like commitments.

6.2 Directions

We introduced a methodology above, but design tools for applying this methodology would considerably enhance its power. Further, other methodologies based on the above semantics may conceivably be invented. Moreover, the above offers an abstract characterization of protocols. It would help to relate the semantics to more concrete forms of reasoning.

It would help to develop a taxonomy and rules of thumb for dealing with the choice of a protocol refinement that a participant can use to maximize its benefit. A designer may use such rules of thumb to specify a desired composite protocol and a participant may use such rules of thumb to seek out or negotiate for particular refinement based on its needs. A natural challenge is to develop an taxonomy geared toward protocols analogous to the taxonomy of business processes described in the MIT Process Handbook [11].

References

1. Huhns, M.N., Stephens, L.M., Ivezic, N.: Automating supply-chain management. In: Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS), ACM Press (2002) 1017-1024
2. : (Rosettanet) www.rosettanet.org.
3. Castelfranchi, C.: Commitments: From individual intentions to groups and organizations. In: Proceedings of the AAAI-93 Workshop on AI and Theories of Groups and Organizations: Conceptual and Empirical Research. (1993)
4. Singh, M.P.: Agent communication languages: Rethinking the principles. IEEE Computer 31 (1998) 40-47
5. Singh, M.P.: An ontology for commitments in multiagent systems: Toward a unification of normative concepts. Artificial Intelligence and Law 7 (1999) 97-113
6. Sirbu, M.A.: Credits and debits on the Internet. IEEE Spectrum 34 (1997) 23-29
7. Mallya, A.U., Singh, M.P.: A semantic approach for designing commitment protocols. In van Eijk, R., Huget, M.P., Dignum, F., eds.: Proceedings of the AAMAS-04 Workshop on Agent Communication. (2004)
8. Yolum, P., Singh, M.P.: Flexible protocol specification and execution: Applying event calculus planning using commitments. In: Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS), ACM Press (2002) 527-534
9. Fornara, N., Colombetti, M.: Defining interaction protocols using a commitment-based agent communication language. In: Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), ACM Press (2003) 520-527
10. Johnson, M.W., McBurney, P., Parsons, S.: When are two protocols the same? In Huget, M.P., ed.: Communication in Multiagent Systems: Agent Communication Languages and Conversation Policies. Volume 2650 of LNAI. Springer-Verlag, Berlin (2003) 253-268
11. Malone, T.W., Crowston, K., Herman, G.A., eds.: Organizing Business Knowledge: The MIT Process Handbook. MIT Press, Cambridge, MA (2003)
12. Grosof, B.N., Poon, T.C.: SweetDeal: Representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In: Proceedings 12th International Conference on the World Wide Web. (2003)
13. Fu, X., Bultan, T., Su, J.: Realizability of conversation protocols with message contents. In: Proceedings of the 2004 International Conference on Web Services, IEEE Computer Press (2004) 96-105
14. Peltz, C.: Web service orchestration and choreography. IEEE Computer 36 (2003) 46-52
15. ebXML: Electronic business using extensible markup language (2002) Technical Specifications release, URL: http://www.ebxml.org/specs/index.htm.

Towards Automatic Discovery of Web Portals

Semantic Description of Web Portal Capabilities

Haibo Yu[1], Tsunenori Mine[2], and Makoto Amamiya[2]

[1] Graduate School of Information Science and Electrical Engineering,
Kyushu University,
6-1 Kasuga-koen, Kasuga, Fukuoka 816-8580, Japan
yu@al.is.kyushu-u.ac.jp

[2] Faculty of Information Science and Electrical Engineering,
Kyushu University,
6-1 Kasuga-koen, Kasuga, Fukuoka 816-8580, Japan
{mine, amamiya}@al.is.kyushu-u.ac.jp

Abstract. Due to the problem of information overload, locating relevant Web portals precisely based on user requirements is quite an essential task. As the need for application-to-application communication and interoperability grows, providing Web portal services that satisfy human as well as machine requirements is becoming a new challenge for Web portals. However, a Web portal capability expressing mechanism, which enables the precise location of Web portals as well as the automated discovery and invocation of Web portal services, is lacking. In this paper, we investigate how to incorporate Semantic Web technology with Web service technologies to describe the capabilities of Web portals. We also discuss the possibilities of using these descriptions for discovering and using the distributed existing portal resources.

1 Introduction

Web portals are information rich sites that gather a variety of useful information from different resources into a single "one stop" Web page and provide it in a compact and easily consumable form to an end-user [1]. However, locating relevant Web portals is quite a challenging task because of the problem of information overload. It caused us to reconsider user requirements on the Web. The following questions come to mind: (1) Are there any better ways to locate relevant Web portal resources precisely based on users' requirements? (2) Is it possible to make use of the relevant heterogeneous Web portal resources automatically no matter what framework they are based on? (3) Furthermore, is it possible to build a user's own personalized information warehouse (MyPortal) on her/his computer with these Web portal resources and use it conveniently, even sharing it with other people? In this paper, we are trying to answer the first question, as a preliminary step towards answering the latter two.

Locating Web portal resources should be based on a match between user requirements and Web portal capabilities. This requires a mechanism for expressing the capabilities of Web portals. Currently, there are some standards used for describing the capabilities of Web sites [1] [2], but they are generally used for the aggregation of Web sites or portlets into a Web portal, not for application computing purposes. So an explicit description of Web portal capabilities, which can support automatic discovery of the Web portal as well as its services, is lacking.

Semantic Web [3] is an evolving technology which aims to tackle the information overload problem of the current Web. In the Semantic Web, the information is given well-defined meaning, better enabling computers and people to work in cooperation.

Web service mechanisms provide a good solution for application interoperability between heterogeneous environments. They are standard programmatic interfaces between applications that provide a new model which enables Web sites to exchange dynamic information on demand.

In this paper, we investigate how to incorporate Semantic Web technology with Web service technologies to support the description of Web portal capabilities, trying to enable the precise location of relevant distributed existing portal resources and their maximum reuse. The advantage of our approach is that we provide a mechanism for describing Web portal capabilities that not only enables precise and automatic discovery, but also enables the application to use the Web portal resources after they are located. Since we use standard ontology language and Web service technology, common existing applications, tools and resources can be used.

The rest of the paper is organized as follows: Section 2 briefly describes the basic technologies this research is concerned with. Section 3 describes a mechanism for the description of Web portal capabilities. Section 4 examines the semantic matching algorithm. In section 5 we discuss the possibilities of using these descriptions for discovering portal resources, to enable the maximum reuse of existing distributed portal resources. Related work is discussed in section 6 and the concluding remarks will be summarized in section 7.

2 The Basic Technologies

We next briefly introduce some of the technologies this research is concerned with.

2.1 Semantic Web

The Semantic Web [3] is an extension of the current Web. It is trying to change the current Web into a huge knowledge base with well-defined meaningful data enabling machines to cooperate with people to tackle the problem of information overload.

RDF (Resource Description Framework) [4] is a metadata modeling language recommended by W3C(World Wide Web Consortium). It provides a common framework for expressing information so it can be exchanged between applications without loss of meaning. It uses XML as an interchange syntax.

Just as people need a common language to communicate with each other, machines also need one in order to share knowledge and to communicate with each other. Ontology is viewed as a dictionary that can satisfy this requirement. RDFS [5] is an ontology language which can formally describe the meaning of terms used in semantic material, and define their relationships and properties. In order to describe more complicated data relationships and perform useful reasoning tasks, OWL [6] was designed and recommended by W3C. These emerging foundation technologies of the Semantic Web have been accepted gradually, and tools for validation, annotation, authoring and editing have also been developed. The ontologies of certain domains have been developed too.

2.2 Web Services

A Web service is "a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols[7]". Web services transform the Web from a collection of information into a distributed computation device.

The service will be instantiated between a requestor and a provider. Standards for description, registration and location, and accessing of Web services such as WSDL[8], UDDI[9] and SOAP[10] have emerged and are widely used currently. But they are based on keywords and lack semantics, and appropriate semantic description of Web service capabilities is necessary in order to enable automatic service discovery and invocation.

OWL-S[11] is "an OWL-based Web service ontology, which supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-interpretable form". It enables automated Web service discovery, execution, composition and interoperation. OASIS[1] has started to discuss UDDI support for semantic search[12]. Research on Semantic Service Matchmaker[13] with a UDDI business registry is also ongoing.

3 The Description of Web Portal Capabilities

In this section, we explain our mechanism for semantically describing Web portal capabilities.

There are various kinds of Web portals fulfilling various purposes with different levels of functions. In this paper, we only focus on the research community Web portal and target the Semantic Web community domain as a starting point. For the purpose of future flexibility, we generalize the concept of Web portal to any information-rich Web sites that want to publish their resources, including individuals, group or project Web sites as well as community Web portals.

[1] Organization for the Advancement of Structured Information Standards.

3.1 The Structure of Web Portal Capability Description

Here we refer to a Web portal which has certain capabilities as a “provider”, and the user or the application that is searching for Web portal resources as a “requestor”.

The purpose of describing the capabilities of a provider is to facilitate the search and use of requestors. Let’s think about the human searching process: generally we match requests and capabilities hierarchically from general ideas to details. This observation can be applied to machine processing as well.

We describe the capabilities of the Web portal by layers. First, we semantically describe the general capabilities of the Web portal, and we call this a “site capability summary(SCS)”. Second, we describe its “service capabilities”. There is a link from the site capability summary to the service capability description. In order to semantically describe the capabilities and support the concrete realization of services, we express the service capability in two layers: “semantic Web service description” and “Web service description”. So the structure of a capability description can be illustrated as seen in Figure 1.

```
          -------------------------
layer 3  | site capability summary |
          -------------------------
            |   ----------------------------------
layer 2     |->| semantic Web service description |
                ----------------------------------
                  |   -------------------------
layer 1           |->| Web service description |
                      -------------------------
```

Fig. 1. Structure of a capability description

This hierarchical capability-describing mechanism enables semantic capability-describing and matchmaking for different levels. There are links between these description layers, but the communication of each layer between provider and requestor can be done independently. The Web service description layer (layer 1) can be WSDL or any other service description method which is being used by the current Web service system. The semantic Web service description layer (layer 2) can be OWL-S or any other semantic service description method.

With this layered description mechanism, providers can describe simple or complicated capabilities and requestors can discover and invoke potential portal resources according to their preferences. For example, a simple Web site which only provides browsing content capability can be described with a site capability summary without the service capability descriptions. A requestor can locate the Web portal without using its layer 1 and 2 services or only use the services of layer 1 without semantic capability. So it’s flexible and robust, and can satisfy all the possible semantic and non-semantic uses. Here we use the well-known service description methods OWL-S and WSDL for the service description of layer 2 and layer 1 respectively in order to maximize the reuse of current resources. For the details of WSDL and OWL-S, one can refer to the relevant documents [8, 11].

In order to semantically describe Web portal capabilities, we need to construct relevant ontologies to model the real-world concepts, create a WSDL document to describe the Web services, use OWL-S to semantically describe the Web services, and create the site capability summary.

We'll define the description of layer 3 "site capability summary" in the next sub-section, discuss Web portal functionalities after that, and then discuss the relevant ontologies.

3.2 The Web Portal Site Capability Summary(SCS)

We argue that some explicit general ideas (site summary) are strongly required in order to precisely locate Web portals based on user preferences. So a brief capability summary of the Web portal is necessary. The site capability summary gives an explicit overview of the Web portal capabilities, and can be used as the initial filter for judging congruence with user preferences. The detailed semantic representations of the contents such as topics, projects, researchers, publications, can be found from the contents metadata which is constructed based on domain specific ontology. We will omit this part in this paper because of space limitations. We define the items of the site capability summary in Table 1.

3.3 The Functionalities of the Web Portals

As a result of thorough investigation by the JISC Subject Portals [14] and PORTAL [15] projects, the features of a portal (including institutional and commercial ones) have been summarized. We referenced their results and examined some typical Web portals, then extracted the main functionalities that we think should be included in a community Web portal. These are described in Table 2.

3.4 Relevant Ontologies

The description of Web portal capabilities must be based on formally defined vocabularies in order to make them machine-understandable and processable. Ontology is used to formally define terms and the relationships between them. The Web portal capability ontology should also include the following component ontologies as well as the ontology of Semantic Web services.

1) The Site Capability Summary Ontology: In this ontology component, the terms used for the site capability summary such as "type", "location", and the relationship between and restrictions on them are formally defined.

2) The Semantic Web Research Community Ontology: There are some existing ontologies for the computer science research domain such as KA2 [16] and CS AKTive Space [17]. We reuse these ontologies and modify them for our purposes to construct our ontology of a research community domain. The ontology defines terms such as Organization, Person, Publication, Events, Topics etc. The relationships between them such as the subclass, subproperty,

Table 1. The site capability summary

Item	**Description**
Topic	The topic is the subject and main concern of the Web portal. A Web portal can have multiple topics. General topics such as "arts", "business", "computers", "education", "entertainment", "health", "recreation", "reference", "science", "regional", "society" will be defined. And the topics of the specific domain such as "semantic Web", "knowledge management", "ontology", "RDFS", "OWL", will be defined in much more detail.
Type	The type is an important characteristic which can identify the Web portal capability. It can be "individual", "organization" or "verticalPortal". The organization can be an "institution", "enterprise", "government", "community", "association", "project", "group", or "others". The vertical portal can be a "digitalLibrary", "e-Journal", "openSource", "event", "news" or "others" for specific interests. This information would be useful when the user can explicitly express his requirement for certain types of Web portals.
Language	This represents the languages supported by the Web portal. We can use existing standard categories of languages to represent it.
Scale	The scale is a rough estimation result. It can be "small", "medium" or "large" to help user identify approximate scale.
Audience	This is used to identify potential user level. It can be "elementary", "intermediate" or "expert". The user with a specific level can make use of this information to locate the most relevant Web portal.
Location	This will define the location of the Web portal. It will be useful when considering network delay or usability. We can use existing standards such as ubr-uddi-org:iso-ch:3166-2003 to describe country and city.
HomePageLink	This is the URL of the Web portal itself. When a user just wants to locate the URL of a Web portal or when a portal does not support Web services, it will be used to connect the user to the main page of the Web portal. As we know, there are times when users want to go to the main page of a Web site and browse the relevant pages by themselves. In these cases, the user generally wants to make use of a set of information or the aggregated information rather than a piece of it. For these cases, it's better to locate the main page of the Web site. There are also times when a user wants to go directly to the Web page which contains the piece of information that the user wants to use instantly. For these cases, it's better to locate the Web page itself. We can make use of user preferences to determine what kind of address to locate.
ServiceLink	The service link is a URL, used as a connection from the site capability summary to the Semantic Web service descriptions. When a user has located a Web portal and wants to use the Semantic Web services, it will be used to reach the Semantic Web services.
Security	It describes the security rule of the Web portal. It can be "private", "community-Only" or "public".
Functionalities	The functionalities is a list of all functions that the Web portal supports. From this list, one can check if the Web portal can satisfy certain requirements or not.

restriction of range and domain are also defined. The Semantic Web involves various research fields, such as Databases, AI, networks, telecommunication, information retrieval, data mining, programming languages, logic, security, Web services etc. So the Semantic Web community involves a broad range of topics. It includes the foundation languages and framework, tools, knowledge sources, applications, technologies, tutorials and so on.

3) The Web Portal Functionality Ontology: This ontology component defines all the terms, relationships, and restrictions concerning Web portal functionalities such as "browsing", "searching".

Table 2. The functionalities of Web portals

Function Name	**Description**
Browsing	This function is used for browsing the contents on the Web portal based on topics, projects, years and so on.
Searching	This function is used for searching relevant resources based on user requirements. It can search internal or external resources. The searching function can provide simple or advanced searches such as match all in any order, match any, match as a phrase etc. It can search materials based on year(single year/a range of years), topic etc. The powerful search function will support searching bibliographic databases, searching citation databases, searching e-journals and searching events such as conferences.
Personalization	The personalized Web portal will enable users to change various aspects to suit themselves. For example, the user can control the appearance of information on her screen, and can change the alert item too. Personalization is strongly required by users as they want to avoid information overload.
Utilities	Some utilities such as "address book", "calendar", "people finder" can be used to help users.
News/Newsfeeds	World or domestic news, technology news, research subject-specific news, job advertisments and alerts can be communicated by email or other means or be aggregated into the Web portal.
Community Communication	The communication between members of a community or group can be realized by chat, newsletters, message boards or newsgroups.
Advertising	Announcements such as conferences, non-leisure events can be advertised through this function.
Teaching and Learning Information	Web-based learning resources, courses or course announcements can be provided by this function.
Assistance With Site Use	This function will support and guide users in using Web sites. It can consist of immediate help, a help page or a feedback option.
Additional Features	The Web portal can provide other features such as job searching, online resource submission, event schedule management or registration into the portal.

4) The Web Service Ontology: This ontology component defines all the terms, relationships, and restrictions concerning Web services. Here we use OWL-S Web service ontology.

4 The Semantic Matching Algorithm

In this section, we explain the semantic matching algorithm we used to match user requirements and Web portal capabilities.

Locating a Web portal and its web services is a process of semantic matching between the requirements of the user and the capability description of a Web portal. The capabilities of a Web portal can be simple or complicated and the requirements of a user can also be quite varied. The matchmaking needs to deal with all possible situations including non-semantic use. For brevity, we use "R" to represent requestor, "P" to represent provider, "SCS" to represent site capability summary.

4.1 The Provider Capabilities

A Web portal may only support contents which can be used through a browsing interface or may also support Web services. So the capability description of a Web portal can be summarized as follows based on its capability.

```
P provides contents only:
  P contains site capability summary and contents metadata
P provides contents and services:
  P contains site capability summary, contents metadata, semantic
  service description, and service description
```

4.2 The User Requests

A requestor(user) can be semantic-capable or not. She may only want to locate the Web portal or also want to locate and use the Web portal services. So the request from the requestor can be summarized as follows based on her purpose.

```
  R wants to locate Web portal only:
     The request contains user preferences and contents query
  R wants to locate and use Web portal services only:
     The request only contains service request
  R wants to semantically locate and use Web services only:
     The request only contains semantic service request
  R wants to locate Web portal and semantically locate and use
  Web portal services:
     The request contains user preferences, contents query and
     semantic service request
```

4.3 The Matching Process

As we use common standard Semantic Web service mechanisms, the matching of Web service capabilities can use the same methods proposed by other research projects such as [18]. Non-semantic Web portal services can also make use of existing matching methods such as UDDI.

The matchmaking process of the provider can be briefly summarized as follows.

```
Case1: P provides contents only
 (1)R wants to locate Web portal only:
    Comparing the preferences of requestor and SCS of Web portal
    if match    then compare contents metadata with contents query
                if match
                then return HomePageLink or relevant Web page
                else end
    else end
 (2)R wants to locate Web services only:
    end
 (3)R wants to semantically locate Web portal services only
    end
 (4)R wants to locate Web portal and semantic Web portal services:
    The process is same as Case 1: (1)

Case 2: P provides contents and services
 (1)R wants to locate Web portal only:
    The process is same as Case 1: (1)
```

```
(2)R wants to locate non-semantic Web portal services only:
   Comparing non-semantic Web service descriptions and service request
   if match    then return Web services information
   else end
(3)R wants to semantically locate Web portal services only:
   Comparing semantic service description and semantic service request
   if match    then return Web services information
   else end
(4)R wants to locate Web portal and semantic Web portal services:
   Comparing the preferences of requestor and SCS of Web portal
   if match
   then compare semantic service description and request
       if match   then return Web services information
       else response HomePageLink only
   else end
```

5 The Possible Uses of the Description of Community Web Portal Capability

The location of Web portals, and the discovery and invocation of Web portal services can be realized in various ways. The matchmaker can be a distributed P2P system or a centralized system. Here we discuss the possible ways that we think reasonable and mainly focus on the usage of a multi-agent community-based P2P information retrieval system.

5.1 Centralized Solution

UDDI is a standard for registration, navigation and location of Web services. It was widely accepted by the industry and there are many applications that are based on it. But it lacks semantic capability and is limited to value sets and direct matching of values. Though it is in its early stages, and there are no solutions yet, work has begun on supporting semantic capabilities for categorization [12]. Other researches on importing the semantic Web in UDDI [18] and the experimental research on semantic service search with the public UDDI registry [13] is also ongoing. Our Semantic Web community Web portal services can be registered in this registry server and can be discovered and invoked based on a semantic-enabled UDDI standard. The semantic description in OWL-S and the service description in WSDL of Web portal services will all be registered into the UDDI registry. So non-semantic service requests can also be processed. When a user wants to semantically locate a Web portal and make use of its Web services, the request will include information about user preferences, contents query and semantic service request. The requestor will send the Web service request to the UDDI Registry and will get a list of potential service providers. It will also send the information for discovering Web portals to the Web portals and will get responses from relevant ones. Then the requestor will examine the responses, select relevant Web portals and their services, and invoke the relevant services.

We will trace the research situation of semantic UDDI standards and investigate the realization details in the future.

5.2 Peer to Peer Solution

Research community members are generally loosely coupled people, distributed across many locations and organizations. They are the community resource consumers and providers simultaneously. The peer to peer model is very natural for modeling community information concerns. Here we discuss a community-based multi-agent P2P retrieval system designed to discover community Web portal services.

In our system, we use the KODAMA [19] multi-agent system to model the research community. Each community member is modeled as a peer and there is one or more agents to serve each community member. They help the community member to locate the relevant resources and make use of the services that they need.

Location of the Web portal resources should be based on a semantic match between the explicit description of the user requirements and the description of the capability of the Web portal. The capabilities of the Web portal are described in the way we explained in section 3. The requestor and the provider share the same ontology to describe their preferences and capabilities respectively.

If a user wants to locate the Semantic Web community Web portal, with contents that are reasonable for a beginner, and wants to locate and use the semantic service of "searching", then his agent will add his searching preferences Type="community", Audience="elementary", the contents request of "Semantic Web" and semantic service request into his request. The Semantic Web community Web portal will find that its capability matches the request preferences and the contents metadata of topics match the contents request. It will continue to do semantic matchmaking of services and response the relevant service interaction information if it can provide the services what the requestor asked. Then the requestor can invoke the relevant searching services.

In order to reduce the message traffic of the P2P network, we make use of the historic records of queries sent to and received from other agents[20].

6 Related Work

In this section, we discuss some related work that is directly or indirectly a concern of our research work.

OASIS released the Web services for Remote Portlets Specification Version 1.0 [1] in August 2003. It defines a web service interface for accessing and interacting with interactive presentation-oriented web services between producer and consumer. Its goal is to enable an application designer or administrator to pick from a rich choice of compliant remote content and application providers, and integrate them conveniently. They use keywords and function calls which lack the semantics that is essential for automatic machine processing. Our description of Web portal services is based on standard semantic Web service ontology OWL-S, which enables automatic machine processing.

RSS[2] and Atom[21] are lightweight multipurpose extensible metadata description and syndication formats. They are XML-based applications and conform to the RDF specification. A brief description of Web site capability can be summarized with them and the summary can be used for online publication, retrieval and further transmission or aggregation. But the resources of the summarized Web site cannot be used as a computational part of the application. Our description is based on Web service technology, so the resources of the portal can not only be located but also used as a computational part of the application.

The Subject Portals project [14] is a project founded by JISC[2]. It builds portals targeted at human users through a Web browsing interface. They support back-end office services for portals but not for portal services targeted at application computing usage, as far as we understand.

There are Web portals based on Semantic Web technology, such as KA2[16] and SEAL[22], which support a semantic portal solution including ontology-based contents construction and maintenance, but they are generally based on human navigation and searching. SEAL provided an interface for a software agent but only for a crawler. None of them supports Semantic Web portal services at present, as far as we know.

Francisco et al. presented an architecture for an infrastructure to provide interoperability using trusted portals[23] and implemented such an infrastructure based on Thematic Portals. The searching portals use semantic access points based on metadata for more precise searching of the resources associated with the potential sources of information. The proposed architecture supports specific and cross domain searching, but only provides semantic representation for the capabilities of Web portals not for their services as we understand. We are concentrating on Semantic Web community domain-specific searching but with a strong semantic describing ability not only for the capabilities of Web portals but also for their services.

OWL-S is an ontology of services which provides a mechanism for semantically expressing the capability of Web services. In our approach, we use OWL-S to describe Web portal service capabilities, and we also add another "site capability summary" layer above it. This will help in the precise location of Web portals as well as the discovery and invocation of Web services.

Some other Semantic Web service solutions [24] are also proposed but none of them for Web portal services as far as we know.

7 Conclusion and Future Work

In this paper, we proposed a mechanism for semantically describing the capabilities of community Web portals, enabling automatic discovery of Web portals as well as Web portal services. We also discussed the possible use of these descriptions to discover Web portal resources. In our future work, we would like to realize the details and implement a prototype to reveal the possibility and

[2] Joint Information Systems Committee

effectiveness of our proposed solution. We will also reuse the resources of automatically discovered portals to aggregate them and construct user-personalized warehouses (Myportal) based on a multi-agent P2P information retrieval system. Currently, we assume that all the portals, users and agents in a community agree on a common ontology and use it to represent the semantics of Web portal capabilities and Web services, but it's not easy to get this agreement in reality. So there may be overlapping or different ontologies describing the same domain that need to be mapped onto each other in order to realize interoperability. There are three dimensions of ontology mapping: discovery, representation, and execution. Some researches [25] [26] on ontology mapping are on going. We need to give further consideration to the ontology mapping issues in the future.

References

1. Web services for Remote Portlets Specification Version 1.0, http://www.oasis-open.org/committees/wsia/documents/WSIA-WSRP-Interface-Spec-0.85.html
2. RDF Site Summary (RSS)1.0, http://web.resource.org/rss/1.0/
3. Berners-Lee, T., Hendler, J. and Lassila, O. "The Semantic Web", Scientific American, May, 2001
4. Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation, 22 February 1999, http://www.w3.org/TR/REC-rdf-syntax/
5. Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation, 27 March 2000, http://www.w3.org/TR/PR-rdf-schema/
6. OWL Web Ontology Language Overview, W3C Candidate Recommendation, 18 August 2003, http://www.w3.org/TR/owl-features/
7. Web Services Architecture Requirements - W3C Working Group Note 11 February 2004, http://www.w3.org/TR/2004/NOTE-wsa-reqs-20040211/
8. Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001, http://www.w3.org/TR/wsdl
9. UDDI Spec Technical Committee Specification,19 July 2002, http://www.uddi.org/specification.html
10. Simple Object Access Protocol (SOAP) 1.1, http://www.w3.org/TR/SOAP/
11. OWL-S: Semantic Markup for Web Services, http://www.daml.org/services/owl-s/1.0/
12. UDDI Spec TC V4 Requirement - Taxonomy support for semantics, http://www.oasis-open.org/committees/uddi-spec/doc/req/uddi-spec-tc-req11-14 -semantics-20040205.doc
13. Takahiro Kawamura, Jacques-Alber De Blasio, Tetsuo Hasegawa, Massimo Paolucci, and Katia Sycara, Preliminary Report of Public Experiment of Semantic Service Matchmaker with UDDI Business Registry, http://www-2.cs.cmu.edu/ softagents/papers/ICSOC03.pdf
14. JISC Subject Portals Project, http://www.portal.ac.uk/spp/
15. PORTAL project, http:/www.fair-portal.hull.ac.uk/
16. KA2 Portal, http://ka2portal.aifb.uni-karlsruhe.de/

17. Nigel R.Shadbolt, monica m.c. schraefel, Nicholas Gibbins, Stephen Harris, CS AKTive Space: or How We Stopped Worrying and Learned to Love the Semantic Web, In Proceeding of 2nd International Semantic Web Conference (ISWC2003), Sanibel Island, FL, USA, Oct. 20-23, 2003
18. Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara, Semantic Matching of Web Services Capabilities, In Proceedings of 1st International Semantic Web Conference(ISWC 2002), LNCS 2342, pp.333-347, Springer 2002
19. Guoqiang Zhong, Satoshi, Amamiya, Kenichi Takahashi, Tsunenori Mine and Makoto Amamiya, The Design and Implementation of KODAMA System, IEICE Transactions on Information and Systems, E85-D(4):637-646, April 2002.
20. Tsunenori Mine, Daisuke Matsuno, Koichiro Takaki, Makoto Amamiya: Agent Community based Peer-to-Peer Information Retrieval, In Proceedings of the Third International Joint Conference of Autonomous Agents and Multi-Agent Systems, Jul. 2004
21. Atom, http://www.mnot.net/drafts/draft-nottingham-atom-format-02.html
22. SEmantic portAL (SEAL), http://ontobroker.semanticweb.org/ontos/aifb.html
23. Francisco Pinto, Claudio Baptista, Nick Ryan, Using Semantic Searching for Web Portal Interoperability. http://www.cos.ufrj.br/wiiw/papers/17-Francisco_Pinto(36).pdf
24. Nicholas Gibbins, Stephen Harris, Nigel Shadbolt, Agent-based Semantic Web Services, WWW 2003, May 20-24, 2003
25. Nuno Silva and Joao Rocha, An Ontology MApping FRAmework for the Semantic Web, Proceedings of the 6th International Conference on Business Information Systems, UCCS, Colorado Springs (CO), USA, May 2003.
26. Borys Omelayenko, RDFT: A Mapping Meta-Ontology for Business Integration, Proceedings of Workshop on Knowledge Transformation for the Semantic Web (KTSW2002) at ECAI'2002, Lyon, France, pp60-68. (http://www.cs.vu.nl/ borys/papers/rdft4ktsw02.pdf)
27. Rainer Tellmann, Alexander Maedche, Analysis of Web Services Solutions and Frameworks, Version: 1.3 Date 2003-02-28, SWWS Deliverable ID: D1.2
28. Transflow UK Ltd.: Service Based Architectures for Web Portals - A White Paper http://www.transflow.co.uk/service_based_architectures_for_portals.pdf
29. http://www.semanticweb.org/
30. http://swws.semanticweb.org/swws

METEOR-S Web Service Annotation Framework with Machine Learning Classification

Nicole Oldham, Christopher Thomas, Amit Sheth, and Kunal Verma

LSDIS Lab, Department of CS, University of Georgia, 415 GSRC, Athens, GA 30602
{oldham, cthomas, amit, verma}@cs.uga.edu

Abstract. Researchers have recognized the need for more expressive descriptions of Web services. Most approaches have suggested using ontologies to either describe the Web services or to annotate syntactical descriptions of Web services. Earlier approaches are typically manual, and the capability to support automatic or semi-automatic annotation is needed. The METEOR-S Web Service Annotation Framework (MWSAF) created at the LSDIS Lab at the University of Georgia leverages schema matching techniques for semi-automatic annotation. In this paper, we present an improved version of MWSAF. Our preliminary investigation indicates that, by replacing the schema matching technique currently used for the categorization with a Naïve Bayesian Classifier, we can match web services with ontologies faster and with higher accuracy.

1 Introduction

With the growing popularity of Web services, the discovery of relevant Web services is a problem. The current approach for discovery is to perform a keyword based search of the UDDI. Like other keyword based search techniques this suffers from problems such as ambiguity, synonymy, etc. One solution to this problem is to make Web services descriptions more meaningful by annotating the service descriptions with machine understandable metadata from shared ontologies [5]. Semantic search engines can then take advantage of this metadata. The most common approach to annotation is to relate elements in the WSDL description to domain specific ontologies. This process will enable the discovery, interoperation, and composition of Web services to be much more efficient. The current research of Web service annotation largely focuses on manual annotation [1], which, looking at the growing numbers of Web services and ontologies, will be time consuming and expensive.

Furthermore, individual ontologies can be very large (e.g. the world-fact-book ontology contains more than 1100 concepts. It has also been reported that real world populated ontologies often exceed 1 million instances [6]. This makes even the discovery of the corresponding concepts within the ontology a tedious task. Further complications arise when a WSDL can be matched to multiple ontologies. Taking these problems into consideration, it is imperative that an efficient tool for (semi-)automatic annotation be used.

We describe the architecture, implementation, and functionality of MWSAF as well as our machine learning approach to improve MWSAF in this paper. The main contributions of our work are:

- Addressing the need for semantics in the Web services framework, and providing a detailed approach that identifies four types of semantics for describing Semantic Web services.
- Identifying the technical challenges in (semantic) annotation of Web services.
- Implementing a machine learning approach to quickly classify Web services into domains.

MWSAF is an approach for semi-automatic annotation. It annotates WSDL descriptions of the services with metadata from relevant ontologies. Our approach will use MWSAF for annotation, but will replace the method used by MWSAF for classifying a Web service into a domain.

This paper will provide an explanation of how MWSAF currently tackles the complicated task of automatically matching in Section 2. An evaluation of the accuracy and performance of MWSAF will comprise Section 3. In Section 4 we will present our machine learning approach for enhancing MWSAF and evaluate this approach in Section 5. Finally, in Section 6 we will discuss the related and future work in this area.

2 Meteor-S Web Service Annotation Framework (MWSAF)

Consistent with Jim Hendler's hypothesis "a little semantics goes a long way", METEOR-S augments the current Web services technology by adding semantic metadata to the syntactical WSDL descriptions. Since WSDL is a de facto standard, this method seems more practical to us than completely changing the paradigm of Web services descriptions. As identified in [1], the four categories of semantics in the complete Web process lifecycle are:

- Data Semantics (semantics of inputs / outputs of Web services),
- Functional Semantics (what does a service do),
- Execution Semantics (correctness and verification of execution),
- QoS Semantics (performance/cost parameters associated with service), MWSAF focuses on data semantics.

2.1 MWSAF Matching Issues and Techniques

In order to annotate, concepts from the WSDL must be matched to concepts from an appropriate ontology. Therefore, the suitable ontology must be identified out of an ontology store. A Web service must be categorized into a domain and the ontology most appropriate for annotation must be determined. Due to the difference in expressiveness of WSDL and OWL, it is difficult to directly match the two formats. WSDL files describe bindings for Web services while ontologies represent concepts and the relationships between them. MWSAF proposes to bridge the gap by converting each to a common representation called schemaGraphs. A schemaGraph is simply a graph or tree representation of the XML or DAML-S document. The conversion rules are explained in-depth in [1].

Once the schemaGraphs have been created, matching algorithms are executed on the graphs in order to determine similarities. Once a concept is matched against all the concepts in an ontology, the best mapping according to the criteria is chosen for annotation. Several algorithms have been defined for matching.

2.1.1 ElemMatch and Schema Match

MWSAF can perform both element and structure level schema matching. The ElemMatch function performs the element level matching based on the linguistic similarity of the names of the two concepts. Note that the two concepts must have similar names for the match to be recognized. A Porter Stemmer [7] is used to extract the root word. The synonyms are checked using WordNet®. An abbreviation dictionary is referenced to handle acronyms and abbreviations. ElemMatch also uses an NGram algorithm to determine element level linguistic similarity. The results of these algorithms determine an ElemMatch score.

The SchemaMatch function examines the structural similarity between two concepts. A concept in an ontology is usually defined by its properties, superclasses and subclasses. Since concept labels are somewhat arbitrary, examining the structure of a concept description can give more insight into its semantics. SchemaMatch accounts for this by calculating the geometric mean of Sub-concept Similarity and the Sub-concept Match. The Sub-concept Similarity is the average match score of each individual property of the concept. Sub-concept match can be defined as the fraction of the total number of properties of a concept that are matched.

Both ElemMatch score and SchemaMatch score are then used to determine the final match score. Formulas, implementation details and results are shown in [1].

2.2 Web Service Classification

Once the best set of matches has been determined, the Web service can be classified to a domain based on the previous calculations. A set of mappings is created for each ontology. Two measures are derived from these sets of mappings; the first is the Average Concept Match and the second is the Average Service Match.

The Average Concept match tells the user about the degree of similarity between matched concepts of the WSDL schema and ontology. This measure is used to decide if the computed mappings should be accepted for annotation. The Average Service Match helps to categorize the service into categories. It is calculated as the average match of all the concepts of a WSDL schema and a domain ontology. The domain of the ontology corresponding to the best Average Service Match also represents the domain of the Web service.

2.3 Web Service Annotation

After the matching has been completed, the annotations can be added to the WSDL. The best match set is presented to the user to choose to accept or reject the matches. Concepts can also be matched manually if the automatic technique failed in some or all matches. Upon acceptance, the mappings are written back to the WSDL file. The output of the tool is the semantically augmented WSDL file.

3 Evaluation of MWSAF

MWSAF was tested using the Weather and Geographical domains. Although the ontologies used are not comprehensive enough to cover all the concepts in these domains, they are sufficient enough to serve the purpose of categorization. For overall results for matches, annotations, and more details regarding the test bed see [1]. We will focus on the results of tests of categorizing services into domains.

3.1 Classification Accuracy

The accuracy of the MWSAF classification depends on the number to which a threshold is set. The services are categorized based on the categorization threshold (CT), which decides if the service belongs to a domain. If the best average service match calculated for a particular Web service is above the CT then the service is predicted to belong to the corresponding domain. Figure 1 depicts the categorization obtained by applying the algorithm on a set of 24 (15 Geography, 9 Weather) Web services for different CT values. In the case of CT = 0.5, the recall was 58%. Whereas for CT = 0.4, although all Web services are categorized, two services from the weather domain have been wrongly categorized in the geographical domain. By tweaking the CT value, the user can either improve precision or recall of the system.

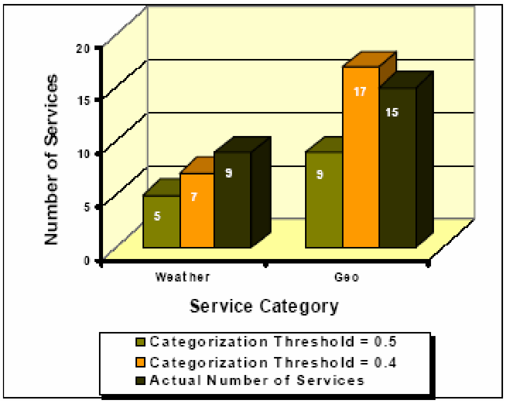


Fig. 1. Categorization statistics of Web services

3.2 Performance

In order to find matches, MWSAF performs an exhaustive search. Each node of a WSDL schemaGraph is compared to every node of each ontology schemaGraph. For this reason the current matching techniques are very expensive in terms of time. This cost is greatly increased as more ontologies are added to the store, thus rendering the algorithm unscalable. We propose replacing this algorithm with a machine learning technique to enhance the speed and efficiency of choosing the appropriate ontology for a Web service by eliminating the node by node matching that MWSAF classification requires.

4 Machine Learning Approach for Web Service Classification

To overcome the apparent drawbacks of the exhaustive search performed in the current version of MWSAF, we decided to replace the schema matching approach for classification with a machine learning approach that uses the Naïve Bayes Classifier implemented in the WEKA toolkit [2] to predict the domain a particular Web service belongs to. This classifier determines the probability that a service belongs to a category by taking the product of the probabilities of each single word belonging to this category. Naïve Bayes Classifiers have quadratic time complexity during the training phase and linear complexity for domain prediction, making this approach highly scalable. Furthermore, these classifiers have been successfully deployed in text classification tasks, even though the independence assumption for distinct features that the classifier makes is clearly violated in natural language, where the context of a word in a sentence is actually quite important.

The features we use to classify the WSDL descriptions into domains are the method names and the argument names, for which we can assume contextual independence, which renders the Naïve Bayes an ideal classifier for our purposes.

The representation we use for a WSDL description as input to the classifier is a feature vector. Each feature represents the frequency of a particular word in the corresponding WSDL file. The position of the word is determined a priori, by parsing all WSDL files and building a dictionary of all the words used in the corpus of WSDL files. Each word has thus a unique ID which corresponds to its position in the feature vector. We use two different measures for word frequency. The first is the raw number of occurrences of a word in the description, the second is a TF-IDF representation [8] of this raw frequency.

4.1 Feature Extraction

The extraction of words from the WSDL descriptions is straightforward. The WSDL XML is parsed. Method names and attribute names are first extracted and then split by our rule of thumb that conventionally a new word in a method name is introduced either by a capital letter or by an underscore/dash. The resulting words are then stemmed [7]. In addition to usual stop words, common terms found in method names such as ‘get’ and ‘set’ are removed. The result of this is a bag of word stems. For every occurrence of a word the corresponding entry in the feature vector is incremented. For TF-IDF, this number is then replaced by the TF-IDF measure.

4.2 Training

Since classification is a supervised learning task, we extract training sets and test sets from our corpus which was provided by Andreas Hess and N. Kushmeric [3]. The training set given to the classifier is a feature matrix. The rows correspond to the feature vectors. Each column of a vector contains the frequency of the word that corresponds to that position of the vector with the exception of the last column which contains the classification of the Web service corresponding to the row vector.

The classifier is then built with this training data. From this, the classifier learns the words and the frequencies of words common to a particular domain.

4.3 Prediction

The second step in our approach is to predict the domain of a given WSDL based on the word frequencies for that WSDL. A set of unclassified WSDL files is given to the classifier.

In order to predict the domain of the given WSDL, the Naïve Bayes classifier only requires a vector of word frequencies. It then compares these frequencies to the training data. During training the classifier learned words that are significant for each domain. Based on the frequencies for the given WSDL and previous training regarding domains and associated words, the classifier then produces numeric predictions for each known domain. The ontology corresponding to the domain with the highest probability is then used for annotation. The classifier generally predicts a domain with almost complete certainty, but in some situations a WSDL will contain high frequencies of words from two domains. In this situation the predictions for both domains are high. This is particularly useful because MWSAF has difficulty handling situations where WSDL matches to multiple ontologies.

5 Application of MWSAF with Machine Learning

Our proposed approach functions by combining all of the phases named in Section 4 with very little involvement from the user. The training of the classifier occurs automatically when MWSAF is loaded. The parser extracts all of the relevant words and passes the frequencies of these words to the classifier along with the associated domain. From this data, the classifier learns which words are highly associated with a domain. Next, the user must load the WSDL file of the Web service to be annotated. The parser then extracts the relevant words from the loaded WSDL and passes the frequencies of those words to the classifier. The classifier uses the Naïve Bayes algorithm to calculate the probability that the Web service might belong to each of the known domains by comparing the word frequencies for this WSDL to what it has previously learned. MWSAF presents the domain for which the classifier calculated the highest probability to the user. If the user accepts the domain, then the corresponding ontology is displayed and the user may then begin selecting matches between the WSDL file and the ontology. If the user rejects the predicted domain, he may chose to predict the domain using the previous technique for classification described in Section 2. When the user has selected the matches between the ontology and WSDL file, the annotations are written to the WSDL.

6 Evaluation of Machine Learning Approach

The machine learning approach described above is advantageous to the MWSAF tool because it is not limited by some restrictions and flaws that limit the MWSAF classification technique. We will discuss the accuracy of the approach for correctly predicting the domain of a service. Then we will discuss why this approach is much faster than the MWSAF approach.

6.1 Classification Accuracy

Due to the fact that the machine learning approach does not rely on an extensive ontology to match to, the accuracy is generally better than the accuracy of classification in MWSAF.

Testing for comparison with MWSAF was done with 37 Web services (16 Weather, and 21 Geography). For quality evaluation purposes, we extracted several training sets of different sizes to measure the quality of the results for cases ranging from much prior knowledge of the domains to be classified (90% training set size) to poor prior knowledge (10% training set size). Five random distributions are then evaluated for every training set size. An average of the five rounds was then calculated for that percentage. The testing results are shown in Graph 2.

Notice that a TF-IDF approach had no positive impact on the results. It is likely that after removing stop words most remaining terms are similarly significant.

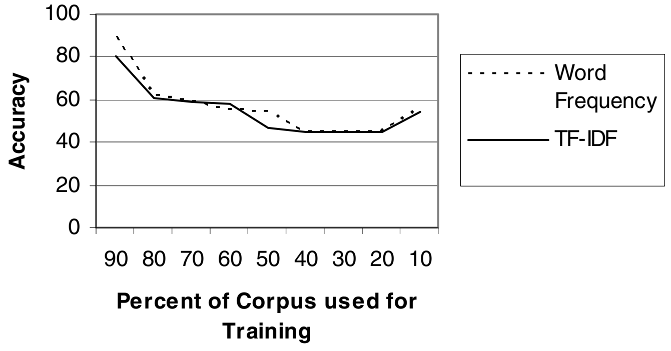


Fig. 2. Machine Learning Classification Statistics

As less training data is provided the accuracy drops first, but remains stable then. This is an encouraging result, because it shows that the number of training examples does not have much impact on the classifier, making it suitable for classifying large sets of unknown services. Note that there are some situations where a WSDL contains many words from both domains. In that type of situation the classifier may give the number 55 for Weather and 45 for Geography. The service might be considered "incorrectly classified" in the results of Figure 2. The results shown in the graph include these situations where a service is wrongly classified but by numbers indicating that it belongs to both domains.

Figure 2 corresponds to a test run on the Geography and Weather domains for comparison with the MWSAF results, but tests were performed for additional domains. The averages drop if the WSDL files in a particular domain are not related enough to have common frequencies. For example, the test bed contains a business domain where the WSDL files and the words used within them are unrelated. The classifier performs poorly when categorizing these business services. Figure 3 illustrates the performance when testing with several domains but excluding business.

The backbone of this approach is finding common words and frequencies of words between WSDL files; therefore, if the terms used for the description of different domains show a high overlap, the classifier cannot identify distinct features and thus may predict the wrong domain. Future work will be to investigate a solution to this problem.

Another limitation of this approach occurs when Web services do not have meaningful names. It is common for a Web service to have attributes with irrelevant and meaningless names. For example, one input might be named “in1”. In a situation like this it is impossible for an automatic matcher of any type to successfully match this attribute with a concept from an ontology. Likewise, a machine learning approach that relies on similar words within the same domain would be hindered by a service containing too many irrelevant words. One solution to this could be to match the extracted word stems to a dictionary such as WordNet and only use terms for classification that appear in the dictionary and are longer than some predefined threshold.

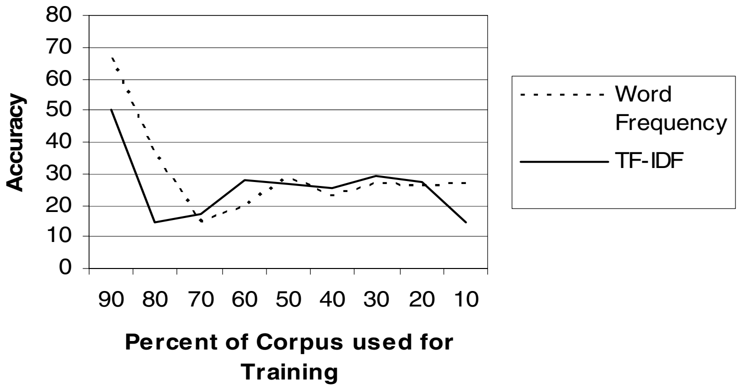


Fig. 3. Classification Statistics for Several Domains

6.2 Performance

As mentioned in section 4, the machine learning approach is significantly faster than the classification approach used by MWSAF because it is not necessary to match every concept of WSDL to every concept of each ontology. This approach does not require the use of an ontology for classification at all. This eliminates the risk of having wrongly categorized services because the ontologies are not comprehensive enough to contain all of the words for matches. The most timely part of our approach is the training phase, however, the training of the classifier itself does not take time. The time used for word extraction and the calculation of word frequencies is minimal. The learner is trained once and the actual categorization of the Web service takes only milliseconds.

6.3 Comparison of MWSAF Classification and Machine Learning Classification

Since the MWSAF schema matching technique has not been evaluated with more than 2 domain ontologies, it is hard to do a comparison in terms of accuracy. In our test case for 2 domains, the classifier's accuracy was on average comparable to the schema matching. Training on a large corpus of documents increased the accuracy radically above the schema matching. In the more realistic case of 10 domains, training with a large corpus produced good results around 68% correctly identified domains, decreasing the training set size let the accuracy drop but stabilize at around 28%. Two conclusions can be drawn from this: a) The more web services are classified, the more accurate will the classification of new services be. b) the approach is scalable. Even with a low percentage of training data the classification is stable.

Overall, the machine learning approach for classification has the potential to be a major enhancement for MWSAF. It is always much faster because it does not require any matching to an ontology. It relies only on the training data obtained from classified WSDL files. Domain prediction is achieved almost instantly. We believe that this approach has the potential to be a much more accurate classifier, and plan to investigate this with future research.

7 Related and Future Work

There is work currently being done in the area of machine learning for the classification and annotation of Web services. Assam [4] is a tool for semi-automatically annotating Web services. The tool provides the user with assistance and suggestions for matches based on the results of machine learning techniques. Assam implements an ensemble learning approach to improves the results [3]. They also researched and tested the approach of using previously annotated WSDL files as training data. Not only did this dramatically improve the results for classification, but it also enabled them to annotate operations, inputs and outputs [3]. The approach presented here improves the first step of MWSAF's schema matching – finding the appropriate ontologies to annotate the Web services. MWSAF's next step is to also annotate operations, inputs, and outputs, for which it still uses a schema matching technique. This is much faster now, because MWSAF only has to compare concepts of a single ontology with method and attribute descriptions in the WSDL file. Extending our machine learning technique to replace MWSAF's matching algorithm for specific concepts to be used for annotation would further enhance the performance of the tool.

Future work includes improving the accuracy of our current machine learning approach so that the averages are higher with less training data. We will investigate ensemble learning techniques. We must also provide a solution to situations where there is large overlap between the significant terms of multiple domains, as explained in section 5.1. Since the approach was very successful for two domains, we will investigate the potential for it to perform better for many domains.

8 Conclusion

As Web services become more widely used it is imperative that semantic metadata is added to Web services. This will facilitate the discovery, composition and interoperation of these services. Since manual annotation is far too expensive and time consuming to be an option, there is a great need for the ability to annotate them automatically or semi-automatically. MWSAF is an effective tool for annotation but is limited by its slow exhaustive schema matching technique. We proposed a machine learning approach that uses a Bayesian classifier to predict the domain of a Web service based on previous training data. This approach is significantly faster than matching every concept of the WSDL to every concept of each ontology in the store. Replacing the classification technique currently used by MWSAF with the naïve Bayesian classifier described in this paper significantly enhances the speed and performance of the tool. How successful machine learning techniques in our annotation framework can be will be shown by our future work of also matching individual classes and properties to methods and attributes.

References

1. Patil, A., Oundhakar, S., Sheth, A., Verma, K.: METEOR-S Web service Annotation Framework., Proceeding of the World Wide Web Conference, (2004)
2. Witten, I., Frank, E.: Data Mining: Practical machine learning tools with Java implementations. Morgan Kaufmann, San Francisco
3. Heß A., Kushmerick N., Machine Learning for Annotating Semantic Web Services. University College Dublin, Ireland.
4. Heß, A., Johnston, E., Kushmerick, N. ASSAM: A Tool for Semi-Automatically Annotating Semantic Web Services, Computer Science Department, University College Dublin, Ireland, ISWC04, (2004)
5. Sivashanmugam, K., Verma, K., Sheth, A., Miller, J., Adding Semantics to Web Services Standards. LSDIS Lab, University of Georgia. ICWS03 (2003)
6. Sheth, A., Ramakrishnan, C., Semantic (Web) Technology In Action Ontology Driven Information Systems for Search, Integration and Analysis. Data Engineering special issue on the Semantic Web, (2003)
7. Porter, M., An algorithm for Suffix Stripping, Program – Automated Library and Information Systems, 14(3):130-137, (1980)
8. Salton, G., Buckley, C., Term Weighting Approaches in Automatic Text Retrieval, Information Processing and Management, Vol. 24, No.5, P513, (1998)

Author Index