First AKT Workshop on Semantic Web Services

AKT-SWS04

KMi, The Open University, Milton Keynes, UK

December 8, 2004

John Domingue, Liliana Cabral and Enrico Motta

Preface

Welcome to AKT-SWS04 the First AKT Sponsored Workshop on Semantic Web Services. The purpose of this one day workshop is to bring together relevant members of the AKT (Advanced Knowledge Technologies) project and the wider research community associated with semantic web services. AKT is a six year UK, EPSRC funded collaborative project between the Universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University. Over the last four years the AKT community has carried out research on the complete life-cycle related to the creation, deployment and maintenance of knowledge services - services which support the creation and sharing of knowledge. During the course of this work a significant number of technologies have been created – one of the highlights being the winning of the Semantic Web Challenge at the International Semantic Web Conference in 2003.

Web services are reusable program components which can be invoked over the internet through an XML based interface and, as such, have attracted considerable interest from both academia and industry. Industrial interest has focused on using web services for enterprise wide application integration and as a baseline for implementing business processes. Research in semantic web services aims to apply semantic web technology to automate or semi-automate the tasks associated with the development and deployment of web service based applications. More specifically, to support the tasks of discovery, mediation, composition and invocation.

Following our overall aim of bringing research communities together we have structured this workshop to be relatively informal and to stimulate discussion as far as possible. The morning of the workshop consists of ten short paper presentations. The first two papers *Treating 'Shimantic Web' Syndrome with Ontologies* and *Agent-based Mediation in Semantic Web Service Framework* look at how mismatches between web services in terms of data representation and protocols can be resolved. These are followed by two papers *Orchestration of Semantic Web Services in IRS-III* and *Interactive Composition of WSMO-based Semantic Web Services in IRS-III* which describe how semantic web services can be composed within The Open University's IRS III platform.

The final paper in the first session *SWSDesigner: The Graphical Interface of ODESWS* outlines a semantic web services development environment based on the UPML framework of tasks and problem solving methods. Tasks are also a key component of the first paper after the break - *Developing a Service-Oriented Architecture to Harvest Information for the Semantic Web*. Within this paper tasks are used to support reusable workflow templates. Issues related to the explicit represention of user's requests to facilitate web service discovery are covered by *Integrating Preferences into Service Requests to Automate Service Usage*.

At the boundaries of semantic web service based platforms XML based representations have to be converted into and out of semantic representations. These activities are called respectively *lifting* and *lowering*. Lowering in the context of IRS III is described in *OCML Ontologies to XML Schema Lowering*.

The final two workshop papers are related to OWL-S which last month was the subject of a member submission to W3C. The first *Using OWL-S to Annotate Services with Ancillary Behaviour* proposes extensions to OWL-S to allow non core functionalities and attributes of web services to be described. The second *Integration of OWL-S into IRS III* outlines a tool which automatically translates OWL-S descriptions into the WSMO based descriptions used in IRS III.

The afternoon starts with a panel *Which Semantic Web Service Standards?* In this session panellists will describe the strengths and weaknesses of approaches based on activities related to OWL-S, WSMO, SWSI and the IRS. After the panel session the majority of the afternoon is dedicated to focused discussion groups. The workshop closes with a demo session where participants will be able to examine a number of semantic web service related tools in detail.

We are pleased that this workshop has attracted considerable interest outside of the AKT consortium. In addition to our AKT partners, we are happy to welcome participants from British Telecom, De Montfort University, the Digital Enterprise Research Institute at Innsbruck, Essex County Council, Hewlett-Packard, the University of Karlsruhe, the University of Leicester, the University of Manchester, and Universidad Politécnica de Madrid.

We would like to thank all of our reviewer's for carrying out their reviews within a very short time scale. We would also like to thank Ben Hawkridge for arranging for the workshop to be webcast. Finally, we would like to thank Jane Whild for the enormous help that she has provided with the logistical aspects of setting up the workshop.

John Domingue, Liliana Cabral, and Enrico Motta

Programme

9:15 - 9:30	Welcome by John Domingue
9:30 - 9:45	1. Treating “shimantic web” syndrome with ontologies Duncan Hull, Robert Stevens, Phillip Lord, Chris Wroe, and Carole Goble
9:45 - 10:00	2. Agent-based Mediation in Semantic Web Service Framework Renato de Freitas Bulcao Neto, Yathiraj Bhat Udupi, and Steve Battle
10:00 - 10:15	3.Orchestration of Semantic Web Services in IRS-III Roberto Confalonieri, John Domingue and Enrico Motta
10:15 - 10:30	4. Interactive Composition of WSMO-based Semantic Web Services in IRS-III Denilson Sell, Farshad Hakimpour, John Domingue, Enrico Motta and Roberto C. S. Pacheco
10:30 - 10:45	5. SWSDesigner: The Graphical Interface of ODESWS Asunción Gómez-Pérez, Rafael González-Cabero and Manuel Lama
10:45 - 11:15	Coffee break
11:15 - 11:30	6. Developing a Service-Oriented Architecture to Harvest Information for the Semantic Web Barry Norton, Sam Chapman and Fabio Ciravegna
11:30 - 11:45	7. Integrating Preferences into Service Requests to Automate Service Usage Michael Klein and Birgitta Konig-Ries
11:45 - 12:00	8. OCML Ontologies to XML Schema Lowering Vlad Tanasescu, John Domingue and Liliana Cabral
12:00 - 12:15	9. Using OWL-S to annotate services with ancillary behaviour Roxana Belecheanu, Mariusz Jacyno and Terry Payne
12:15 - 12:30	10. Integration of OWL-S into IRS-III Farshad Hakimpour, John Domingue, Enrico Motta, Liliana Cabral and Yuangui Lei
12:30 - 13:30	Lunch and coffee
13:30 - 14:45	**Panel**: "Which Semantic Web Service standards?" Terry Payne, Michael Stollberg, John Domingue and Steve Battle
14:45 - 15:45	**Discussion Group 1** - MIAKT Position Paper: MIAKT position paper, MIAKT component services
14:45 - 15:45	**Discussion Group 2** - Applications Position Paper: Web Service Support for Scientific Data Analysis Other application papers: 1.

14:45 - 15:45 **Discussion Group 3** - SWS Composition/Orchestration/Planning
Position Paper: Proposed Functional-Style Extensions for Semantic Web Service Composition
Other Papers: 3, 4

15:45 - 16:00 Coffee break

16:00 - 17:00 **Discussion Group 4** - MIAKT (cont.)

16:00 - 17:00 **Discussion Group 5** - SWS Description
Position Paper: DIANE Service Description
Other Papers: 7, 9

16:00 - 17:00 **Discussion Group 6** - SWS Mediation/Lifting/Lowering
Related Papers: 1, 2, 8

17:00 - 17:20 Feedback and Wrap up

17:20 - 18:00 Demo session with Cream Tea

Demo 1: OntoSearch
Demo 2: SWSDesigner
Demo 3: IRS-OWL-S Translator
Demo 4: Composition Interface for IRS-III
Demo 5: OCML-XSD lowering

Organisation

Steering Committee

John Domingue, KMi, The Open University - Chair
Enrico Motta, KMi, The Open University

Workshop Organisers

Liliana Cabral, KMi, The Open University

Program Committee

Steve Battle, HP Labs, UK
Fabio Ciravegna, University of Sheffield
Oscar Cocho, ISOCO, Spain
John Davies, BT, UK
Asuncion Gomez-Perez, UPM, Spain
Farshad Hakimpour, KMi, The Open University
Martin Kollingbaum, University of Aberdeen
Ruben Lara, DERI, Innsbruck, Austria
Mathew Moran, DERI, Galway, Ireland
Barry Norton, University of Sheffield
Terry Payne, University of Southampton
Dave Robertson, University of Edinburgh
Nigel Shadbolt, University of Southampton
Derek Sleeman, University of Aberdeen
Monica Solanki, De Monfort University, UK
Michael Stollberg, DERI, Innsbruck, Austria

Attendee name	Affiliation
Marc Richardson	British Telecom
Monika Solanki	De Montfort University
Michael Stollberg	DERI
Leticia Gutierrez	Essex County Council
Rob Davies	Essex County Council
Steve Battle	Hewlett Packard Labs
Denilson Sell	OU
Dileep Damle	OU
Dnyanesh Rajpathak	OU
Enrico Motta	OU
Farshad Hakimpour	OU
Gaston Burek	OU
Jianhan Zhu	OU
John Domingue	OU
Liliana Cabral	OU
Maria Varga-Veras	OU
Michele Pasin	OU
Neil Benn	OU
Roberto Confalonieri	OU
Tom Heath	OU
Vanessa Lopez	OU
Vladimir Tanasescu	OU
Yuangui Lei	OU
Michael Klein	Universität Karlsruhe
Derek Sleeman	University of Aberdeen
Edward Thomas	University of Aberdeen
Martin Kollingbaum	University of Aberdeen
Chris Walton	University of Edinburgh
Dave Robertson	University of Edinburgh
Stephen Potter	University of Edinburgh
José Luiz Fiadeiro	University of Leicester
Duncan Hull	University of Manchester
Robert Stevens	University of Manchester
Sean Bechhofer	University of Manchester
Barry Norton	University of Sheffield
Fabio Ciravegna	University of Sheffield
Simon Foster	University of Sheffield
Ayomi Bandara	University of Southampton
David Dupplaw	University of Southampton
Gary Wills	University of Southampton
Hugh Glaser	University of Southampton
Mariusz Jacyno	University of Southampton
Mischa M Tuffield	University of Southampton
Nick Gibbins	University of Southampton
Paul H. Lewis	University of Southampton
Roxana Belecheanu	University of Southampton
Sebastian Stein	University of Southampton
Srinandan Dasmahapatra	University of Southampton
Stephen W Ball	University of Southampton
Steve Harris	University of Southampton
Terry Payne	University of Southampton
Y.David Liang	University of Southampton
Asunción Gómez-Pérez	UPM
Rafael Gonzalez-Cabero	UPM

Panelists Bios

Dr. Terry R. Payne is a lecturer at the University of Southampton, UK. He received a BSc in Computer Systems Engineering from the University of Kent at Canterbury, UK, and an MSc and PhD in Artificial Intelligence from the University of Aberdeen, Scotland. He spent four years at the Robotics Institute at Carnegie Mellon University, where he became involved in the DAML program. He is a co-author of the DAML-S / OWL-S Service Description Language, and a member of the Semantic Web Services Language Committee (SWSL). Contact him at the School of ECS, University of Southampton, Highfield, Southampton, SO17 1BJ; trp@ecs.soton.ac.uk; http://www.ecs.soton.ac.uk/~trp/index.html.

Michael Stollberg, M.A., is a researcher at DERI – Digital Enterprise Research Institute and a PhD student at the University of Innsbruck, Austria. His main research fields are architectures for Semantic Web Services and mechanisms for discovery and contracting. Michael Stollberg is project manager of the Semantic Web Fred project, and is involved in DIP, an Integrated Project on the 6th framework concerned with Semantic Web Services. Michael Stollberg is a founding member of the WSMO working group, wherein he is responsible for dissemination and exploitation of WSMO. Michael Stollberg is member of the DERI Business Development effort, wherein he is responsible for developing a professional marketing and dissemination strategy for DERI technologies.

Dr. Steve Battle gained his PhD in the area of Constraint Satisfaction Problem solving, at the University of the West of England, Bristol, in 1996. Further research at UWE involved the development of innovative distributed systems and mobile agent technology within two EU projects; the FollowMe project (ESPRIT 25.338) and the Traffic Engineering Network Data System (TRENDS – ESPRIT 20.791). After joining Hewlett-Packard Labs in 1999 he continued this research in service-oriented computing, working on the development of e-services for print. He is currently engaged in the EU Semantic Web enabled Web Services Project, or SWWS (IST-2002-37134), and is responsible for capturing semantically enriched descriptions of web-services operated across HP

Dr. John Domingue is the Deputy Director of the Knowledge Media Institute at The Open University, UK. Since the late 90s he has been investigating how knowledge and internet technologies can support the creation and sharing of knowledge. His main current work is centred on KMi's framework and platform for creating and deploying semantic web services IRS III. He is the Scientific Director of the EU funded Integrated Project on semantic web services DIP and is a co-Principle Investigator on AKT. Dr. Domingue is also a chair of the WSMO working group. More details on his work can be found at http://kmi.open.ac.uk/people/domingue/.

Table of contents

Position Papers

Demo papers

Papers

Treating "shimantic web" syndrome with ontologies

Duncan Hull, Robert Stevens, Phillip Lord, Chris Wroe, and Carole Goble

School of Computer Science, University of Manchester, Oxford Rd, Manchester, UK.

Abstract. This paper describes shimantic web syndrome, the use of "shims" to align or mediate mismatching third party Web Services that have closely related, but incompatible, inputs and outputs. The syndrome is illustrated using services from ^{my}Grid bioinformatics analyses. An ontology driven treatment for managing this syndrome through semi-automated service discovery and invocation is outlined. This treatment is likely to be applicable to other domains.

1 Introduction

In the vision of Semantic Web Services, machine understandable descriptions of data and Web Services facilitate their automated use. Yet, as ever, the devil is in the detail. We observe the need to use *shims* to align Web Services to make them useful or even to work at all. Shims align or mediate data that is syntactically or semantically closely related, but not directly compatible. As an example, we use Services from ^{my}Grid workflows that perform bioinformatics analyses [1]. A significant proportion of the Web Services in these workflows are shims, not directly relevant to the biological purpose of the workflow but required to mediate between outputs and inputs of consecutive services.

Shims are an important part of this workflow because bioinformatics data and services are autonomously created by different groups around the world. Consequently, there is no universally accepted data model for describing the data inputs and outputs that services operate upon. Standards like XML Schema, if used at all, rarely describe more than primitive types like `xsd:string` [2]. These are of limited use when mediating between services that operate on complex structured types like GenBank[1] and UniProt records[2] or BLAST reports[3].

Superficially, services that produce and consume `xsd:string` all match and are compatible with each other. However, because `xsd:string` hides many different complex data types, the degree of matching actually falls into three categories

1. Exact match: output and input are equivalent and compatible
2. Close match: some kind of shim required to align services and bridge gap
3. No match: either syntactic or semantic, services are incompatible

[1] See `http://www.ncbi.nlm.nih.gov/Sitemap/samplerecord.html`
[2] The UNIversal PROTein resource `http://www.uniprot.org`
[3] see `http://www.ncbi.nlm.nih.gov/Education/BLASTinfo/information3.html`

The second scenario, close matching, is common in bioinformatics and exemplifies shimantic web syndrome, where many services are *nearly* compatible. For example, a service producing a UniProt record (which contains a protein sequence) and a BLAST service consuming protein sequences are nearly compatible and require shimming. An `accessor` shim (see Table 1) is required to access the protein sequence from the UniProt record so that it can be passed to BLAST for further analysis. Given the open nature of the Web and the autonomy of the data and service providers [3] this syndrome is unlikely to be cured in the foreseeable future. What distinguishes our work from related projects integrating biological data on the semantic web is, firstly we aim to accommodate autonomous Web Services in an *open* world. Secondly, some of these services do not, and may never, use XML for their data, but pass around strings encapsulating many legacy proprietary file formats. Finally, our motivating examples are taken from real live scenarios using complex data where automation can be dangerous or impossible.

In this paper, we characterise the shimming problem and introduce the potential role of semantic description of services that aim to plug the gap left by the absence of effective, domain specific type systems in aligning third party services.

2 The shimming problem

Shims are symptomatic of integrating implicitly typed bioinformatics data. They are analogous to the shims in the physical world – thin strips of metal used to align pipes or rails. In bioinformatics, shims align data by performing some of the operations normally facilitated by a type system. Some example shims are shown in Table 1. Our shims are subclasses of WSMO[4] mediators, probably `wwMediators` [4]. However, an important difference is that the deployment and invocation of some bioinformatics shims may be difficult to fully automate, because their safe use can be context dependent. The `semantic translator` is one example of this.

We feel the shim metaphor is a useful one for describing and understanding the software components currently required to integrate and understand bioinformatics data. Our definition for a shim is software that transforms between closely related data (either syntactically or semantically) in order to join outputs and inputs of two components, in this case Web Services. As shims are hard to precisely define, we think of shims as a set of symptoms of integrating weakly or implicitly typed (both semantically and syntactically) data. Characterising shims is the first step in working towards novel treatments for "shimantic web" syndrome; which is not peculiar to bioinformatics. In an open world such as the Web, it is likely that the majority of services will need some kind of shim in order to align them with the user's needs.

[4] `http://www.wsmo.org`

Shim type	Input	Operation	Output	Description
Dereferencer	Identifier or Pointer	Dereference	The dereferenced resource	GenBank ID replaced with GenBank record
Syntax translator	Data represented in concrete representation, x	Translate	Data represented in an alternate concrete representation, y	SeqRet translates between representations of sequence data.
Semantic translator	DNA sequence	Translate	Protein sequence	Translate DNA into protein
Mapper	Unique identifier	Map	Unique Identifier	Maps between IDs. E.g. GenBank to EMBL
Parser	Record	Parse	Abstract syntax tree	Parse BLAST report.
Iterator	A set	Iterate	A member of a set	Iterate over members of a given set
Comparer	Two or more sets	Diff	Set of differences	Comparing BLAST reports notifies of new sequences
Accessor	Record	Access	Subset of record	Access a subset

Table 1. Examples of shims taken from the ^{my}Grid project. This partial classification is based on inputs, outputs and the task or operation performed.

3 Semantic approaches to type systems

The capabilities of a type system would alleviate some problems that shims currently address. Being able to coerce, access, infer, reflect and cast data would all be easier if bioinformatics data were more explicitly typed. However, creating a type system for a distributed, dynamic and complex domain like bioinformatics is a non-trivial task fraught with many technical and social pitfalls. Registries of Web Services such as BioMOBY [5] have succeeded because they have allowed data providers to register services quickly with only minimal typing. We cannot expect all bioinformatics data to be typed both semantically and syntactically in order to allow smoother mediation. Some systems [6] impose an internal type system on top of third party services via XML. While providing a working solution, we feel that such an approach is less scalable and restricts the number of services available. In the ^{my}Grid project, we have adopted an approach of using third party services in their native form.

In the ^{my}Grid project we have created an ontology to describe bioinformatics data and services [7] and we are currently extending this model to specifically tackle this syndrome by describing the shim services, the data they process and the main bioinformatics services that the shim services mediate between. The ontology describes the existence of data types without the detailed description of structure, and can map to XML schemas when and where they exist. It describes services semantically so that we know that the output "UniProt record" and input "protein sequence" are closely related and can be mapped between using an `accessor` shim, (see Table 1) to access the protein sequence contained in the UniProt record. This process currently requires human intervention; however,

with the help of an ontology, these sorts of transformations can be supported and the user guided to make biologically sensible workflows.

We intend to use the ontology when constructing workflows to recognise service mismatches and identify what kind of shim is required. With this information, a suitable shim or shims could be automatically retrieved from a library. Our goal is to make shim management more transparent to the user. By describing the transformations shims perform, the user composing the workflow can be abstracted away from the details of mediating the underlying services.

4 Conclusion

We have used bioinformatics as our example of shimantic web syndrome, but we suggest this problem will be found throughout the development of applications using autonomous Web Services. There is a high probability that third party services will not *exactly* match the user's needs but will be closely related; this is when a shim will be needed. We propose to treat the syndrome as follows

1. Describe and classify shims using an ontological model
2. Create a library or factory of shim services
3. Use model to identify closely related, but mismatching services
4. Semi-automate discovery and invocation of shims, using the above.

Whichever semantic web architecture is used, successful adoption by the bioinformatics community will require mechanisms for describing, discovering and composing shim services that currently make mediation possible.

References

1. Robert D. Stevens, Hannah J. Tipney, Chris Wroe, Tom Oinn, Martin Senger, Phillip W Lord, Carole A. Goble, Andy Brass, and May Tassabehji. Exploring Williams-Beuren Syndrome Using myGrid. In *Intelligent Systems for Molecular Biology, Glasgow, UK.*, volume 20, 2004. ISSN 1367-4803.
2. Phillip Lord, Sean Bechhofer, Mark D. Wilkinson, Gary Schiltz, Damian Gessler, Duncan Hull, Carole Goble, and Lincoln Stein. Applying semantic web services to bioinformatics: Experiences gained, lessons learnt. 2004. Proceedings of the 3rd International Semantic Web Conference, Hiroshima, Japan.
3. Lincoln Stein. Creating a bioinformatics nation. *Nature*, (417):119–120, May 2002.
4. Massimo Paolucci, Naveen Srinivasan, and Katia Sycara. Expressing WSMO Mediators in OWL-S. *International Semantic Web Conference 2004*, W6:120–134, 2004. Workshop notes VI: Semantic Web Services.
5. M Wilkinson and M Links. BioMOBY: An Open Source Biological Web Services proposal. *Briefings in Bioinformatics*, 3(4):331–341, 2002.
6. Shawn Bowers and Bertram Ludäscher. An ontology-driven framework for data transformation in scientific workflows, 2004. Intl. Workshop on Data Integration in the Life Sciences (DILS'04), March 25-26, 2004 Leipzig, Germany, LNCS 2994.
7. Chris Wroe, Robert Stevens, Carole Goble, Angus Roberts, and Mark Greenwood. A Suite of DAML+OIL Ontologies to Describe Bioinformatics Web Services and Data. *International Journal of Cooperative Information Systems*, 12(4):197–224, June 2003.

Agent-based Mediation in Semantic Web Service Framework

Renato de Freitas Bulcão Neto[1], Yathiraj Bhat Udupi[2], and Steve Battle[3]

[1] University of São Paulo, São Carlos SP 13560-970, Brazil,
rbulcao@icmc.usp.br
[2] North Carolina State University, Raleigh NC 27695, USA,
ybudupi@csc.ncsu.edu
[3] Hewlett Packard Laboratories, Bristol BS34 8QZ, UK
steve.battle@hp.com

Abstract. In a semantic web service scenario clients and services should interoperate by allowing a service to be delivered via different protocols and data formats. This paper describes a novel solution to protocol and data mediation through a goal-driven, agent-mediated interaction with web services described by OWL-S ontologies. Our contributions include: (i) an OWL-S compiler which mediates between two OWL-S service description interfaces and outputs a script containing a set of executable Nuin plans, and (ii) an agent-based mediator built upon Nuin framework that executes these plans in a event-driven fashion.

1 Introduction

There is a need for richer knowledge-based product and service descriptions to enhance the existing business interactions over the Internet. Current approaches to web-service description (e.g. WSDL) are strongly tied to the message syntax and protocol. This paper describes how we enrich the current web-services model with semantic support for goal-driven, agent-mediated interaction with web-services described by OWL-S ontologies [1]. We present a case study about a software product marketplace for vendors who lack a comprehensive sales infrastructure. A service request made using SOAP-based interactions with the web service enables the client to place an order. In the existing application hard-coded java applets enable the user to communicate with the web-service. For a user with a browser it is simpler to have a web-friendly, resource-oriented interaction [2]. The information in the client request is encoded in the request URL query string. This is unlike posting an explicit request message. This poses a mediation problem, where we need to enable clients and services to interoperate by allowing a service to be delivered via different protocols and data formats. There are two perspectives on the mediation problem, first is protocol mediation: how do we describe one service in terms of another and ensure that it achieves the same goals. The second is data mediation: how do we achieve independence from the syntax of the specific messages allowing us to map from one message format to another.

Our approach is set out in the *web service modelling framework (WSMF)* [3] and it caters to the main objectives of WSMF. It supports rich, declarative service descriptions, which separates the design of the service functionality from its delivery and provides for a framework in which those descriptions are used. Mediation is achieved within

an agent-based framework moving from the syntactic domain of messages into a representational framework based on semantic web technologies (RDF and OWL). This agent-based mediator assists the client in achieving specific goals, which seen as key to identifying the tasks and actions to be performed by the service.

Contribution. Our first contribution is the development of an OWL-S compiler which mediates between two different OWL-S service descriptions derived from the *requester* (the client), and the *service provider* interface and outputs an executable Nuin script [4]. Nuin is an agent-framework with emphasis on the building of Semantic Web agents. Our second contribution is the development of the agent-based mediator built upon the Nuin framework that executes the script generated from the compiler in an event-driven fashion. These approaches to solving the mediation problem help us overcome the main barriers to *e-business process automation*.

2 Agent-Based Mediation

This section describes agent-based mediation and event-driven plans, and is demonstrated by an example scenario.

2.1 The Agent Framework in the BDI Architecture

The agent framework which animates the WSMF can be described in terms of a *belief-desire-intention* (BDI) architecture [4]. Various elements of the conceptual architecture are mapped into agent beliefs, desires and intentions. *Beliefs* correspond to the background knowledge of the agent held in its knowledge base (updated with message content at run-time) and its accompanying ontology. *Desires* include information about the client goals, comprising of the information in the service request which is based on the OWL-S profile (including important service parameters). The *intent* of the user is conveyed to the agent through individual requests at the user interface. This way the agent translates the desires and intents of the user into tasks and actions at the provider interface.

The agent executes the various Nuin plans that perform the required protocol and data mediation. These plans coordinate activities across the various plug-in components that support communication with the client and the service provider. Figure 1(a) describes the agent architecture. The *web plug-in* of the agent mediator functions as an adapter between a web server and the agent, lifting HTTP requests into RDF and mapping responses back into HTML. The *service plug-in* of the agent acts as an adapter to an invocation client for the SOAP web services. A *lift module* provides an interpretation of the message content and a translation to or from a common representational form, RDF model, based on XML schema [5]. A request message gets dropped from RDF into XML and conversely responses are lifted from XML back into RDF.

2.2 Protocol Mediation by Process Planning with the OWL-S Compiler

Compilation is an off-line process that generates the Nuin plans required to mediate between the requester and the provider interfaces, and is performed by the OWL-S compiler. Both interfaces include OWL-S service descriptions including descriptions of *inputs*, *outputs*, *preconditions*, *unconditional effects*, *service parameters*, etc. The

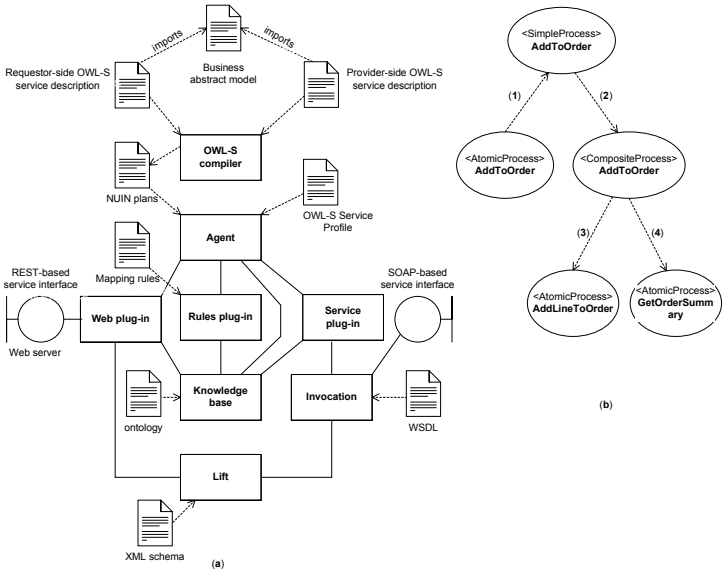


Fig. 1. (a) Agent mediation architecture. (b) An example of event-driven intent invocation.

service descriptions at both interfaces need not have a one-to-one mapping between them. Where the immediate effects of actions at the two interfaces do not correspond exactly, we define composite processes that have the required combined effect. Also, an *abstract business process model* represents the abstract view of the provider interface. This model is imported by the concrete processes of the two interfaces. The primitive parts of the abstract process are of type *OWL-S SimpleProcess* allowing us to describe a business process independently of its realization.

The OWL-S compiler reads the above descriptions and outputs a set of executable Nuin plans. The compiled output is modular in that each plan corresponds to an atomic, composite or simple process. Each atomic process corresponds to an invocation or receipt of a message (that may require a response). Each composite process corresponds to a breakdown of the plan into smaller tasks. Nuin supports backtracking enabling us to back out of a plan where the preconditions do not hold. Simple processes realized in different ways at the two interfaces, create the bridge necessary for protocol mediation. They are the point where the agent recognises the user intent and then forms its own intent to act.

2.3 Data Mediation using Mapping Rules

The agent-based mediator equipped with a *Rules plug-in* performs data mediation, which realizes the mapping between the incoming and outgoing message content and their common ontological conceptualizations. Mapping rules are applied to the content stored in the knowledge base representing previously received and lifted message content. The rules plug-in is based on the Jena rules engine and the mapping rules are expressed in the Jena rules language [6].

2.4 Event-driven Intent Invocation

The agent plans are designed to allow for event-driven triggering of plans. At the requester interface, the web plug-in extracts the query parameters and raises an event that signals the receipt of the message. The event triggers a plan corresponding to an atomic process which, in effect, recognises the event as a user action. In turn the atomic process plan signals the user action with another event. This event may trigger composite process plans that recognise more complex actions. At the point where the occurrence, or recognition, of a process on the user-side corresponds to an equivalent process available on the provider-side, the agent can form the intent to act. Once the invocation of a process against the provider has completed it it necessary to complete the user-side action by returning the appropriate HTTP response. This is viewed as a continuation of the action that raised the original event.

Figure 1(b) describes a scenario of the invocation of an atomic process at the requester-side interface being translated to a composite process with its component atomic processes in the provider-side interface. The user simply wants to add an item to his shopping basket; to *AddToOrder*. Step (1) is an event that alerts the agent to the user action to add an item to the order. It will include an input that identifies the required product. We see that the *AddToOrder* simple process is realized in different ways on the requester and provider sides. They are processes with equivalent effects. At step (2) the agent forms its own intent to *AddToOrder* at the provider interface. This happens to be a composite process plan, invoked from the simple process plan. This invokes the individual atomic process plans *addLineToOrder* and *getOrderSummary* in steps (3) and (4) respectively. Note that given the intention to *AddToOrder* in step (2) we drive the process in a top-down way. However, prior to recognising an intention we drive the process from the bottom-up.

3 Conclusion

This paper described an agent-based solution to protocol and data mediation following the major objectives set out by the WSMF: services, mediation, ontology, and goals. We have shown how OWL-S service descriptions may be used within an agent-based framework to support this ontology-based mediation.

References

[1] OWL-S Coalition. OWL-S 1.0 Release. At http://www.daml.org/services/owl-s/1.0/, 2003.
[2] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD dissertation, University of California, Irvine, USA, 2000.
[3] D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF.
At http://informatik.uibk.ac.at/users/c70385/wese/, 2002.
[4] I. Dickinson and M. Wooldridge. Towards Practical Reasoning Agents for the Semantic Web. In *International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 827–834, Melbourne, Australia, 2003.
[5] S. Battle. Round-tripping between XML and RDF. In *International Semantic Web Conference*, At http://iswc2004.semanticweb.org/posters/PID-BRRGVFRE-1090254811.pdf, 2004.
[6] HP Labs Semantic Web Team. Jena Semantic Web Framework. At http://jena.sourceforge.net/, 2003.

Orchestration of Semantic Web Services in IRS-III*

Roberto Confalonieri[1,2], John Domingue[1] and Enrico Motta[1]

[1]Knowledge Media Institute, The Open University, Milton Keynes, UK
{r.c.confalonieri, j.b.domingue, e.motta}@open.ac.uk
[2]Department of Computer Science, Università di Bologna, Bologna, Italy
confalon@cs.unibo.it

Abstract. In this paper we describe our orchestration model for IRS-III. IRS-III is a framework and platform for developing WSMO based semantic web services. Orchestration specifies how a complex web service calls subordinate web services. Our orchestration model is state-based: control and data flow are defined by and in states respectively; web services and goals are modeled as activities and their execution triggers state changes. The model is illustrated with a simple example.

1 Introduction

Web services are the new way of interacting with and using the web; users are expected to seek for appropriate web services that help them to achieve their goals. This process of manual search may be automated if web services are augmented with semantic descriptions and infrastructures for supporting them are developed [3]; IRS-III [2] is a framework and implemented infrastructure which supports the creation of semantic web services according to the WSMO ontology [6].

Web service interfaces play an important role in the composition and execution of web services. *Choreography* describes the interaction process between web services. *Orchestration* represents the workflow steps of a composite web service fulfilling its capability as its decomposition. Whilst choreography for IRS-III has been almost defined [1], orchestration has not.

In this paper we present an ontology for modeling the orchestration of a composite web service in IRS-III and an interpreter that executes it. The model is in OCML [4].

The paper is organized as follows: Section 2 briefly overviews orchestration. Section 3 describes our model and implementation issues through a simple example. Section 4 contains conclusions and describes future work.

2 Orchestration

Orchestration is a process-centric view of the interactions between the composite web service and the sub services that it relies upon. It includes complex process se-

*This work is supported by the DIP (Data, Information and Process Integration with Semantic Web Services) and AKT (Advanced Knowledge Technologies) projects. DIP (FP6 - 507483) is an Integrated Project funded under the European Union's IST programme. AKT is an Interdisciplinary Research Collaboration (IRC), which is sponsored by the UK Engineering and Physical Sciences Research Council under grant number GR/N15764/01. The AKT IRC comprises the Universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University.

mantics (loops, conditions, fork...) and/or workflow steps that are outsourced to external services. In a nutshell orchestration describes how the service works from the provider's perspective, i.e. how a service makes use of other services represented by activities in order to achieve its capability [5, 7]. This can be done in two ways: the specific sub services are fixed in the activities at design-time (static composition); activities are dynamically bounded by declaring them as goal descriptions on the basis of any goal-service discovery mechanism (automatic composition). An activity is expected to use a particular interface for the sub services it binds at run-time; the interface contains details about services it uses to solve the goal currently being processed. Heterogeneity mismatches between the used interface and the one needed have to be resolved through mediation.

3 State-based orchestration in IRS-III

We choose a state machine representation for the orchestration of semantic web services in IRS-III. In our approach states define the control flow, transitions represent *activities* and activity execution triggers state change. An activity can be a web service (simple or composite) or a goal as IRS-III supports capability-driven invocation of web services.

According to the WSMO standard model, a web service interface description is composed of choreography and orchestration; an orchestration has a *problem solving pattern* (fig.1).

```
currency-converter-orchestration (orchestration)
  has-problem-solving-pattern :value currency-converter-psp

currency-converter-psp (problem-solving-pattern)
  has-start :value currency-converter-psp-start
  has-end :value currency-converter-psp-end
```

Fig. 1 Orchestration definition of the currency converter composite web service

In our *orchestration ontology* a problem solving pattern consists of a set of classes modeling states and activities, with a *start-state* and *end-state* classes connected by control construct state classes. Start-state and end-state represent respectively the entry and exit data flow points for the data of the composite web service being orchestrated (fig.2).

```
currency-converter-psp-start (start-state)
  has-input-role :value has_source_currency
                 :value has_target_currency
                 :value has_amount
  has_source_currency :value(wsmo-orchestration-role-value
                 currency-converter-web-service 'has_source_currency)
  has_target_currency :value (wsmo-orchestration-role-value
                 currency-converter-web-service 'has_target_currency)
  has_amount :value (wsmo-orchestration-role-value
                        currency-converter-web-service 'has_amount)
  has-later-state :value exchange-rate-sequence-state
```

```
currency-converter-psp-end (end-state)
  has-output-role :value has-currency-conversion
  has-currency-conversion :value (wsmo-orchestration-role-value
                                   multiply-activity 'multiply-output)
```

Fig. 2 Start-state and end-state definitions of the currency converter orchestration; input and output-role value slots reflect input and output-role of the currency converter web service

Control states are wrappers for activities (fig.3) and they represent the control flow as an execution path in the model. We've defined three control construct states:

- *sequence-state*: the sequence construct is the elementary unit of orchestration as web services and goals are represented by activity in sequence-states;

```
exchange-rate-sequence-state (sequence-state)
  has-activity :value exchange-rate-activity
  has-later-state :value multiply-sequence-state

multiply-sequence-state (sequence-state)
  has-activity :value multiply-activity
  has-later-state :value currency-converter-psp-end
```

Fig. 3 Sequence state definitions of the currency converter orchestration: the currency-converter web service is composed by two activities to be executed in sequence; the interpreter selects the next state through the `has-later-slot`

- *conditional-state*: the conditional construct checks if a certain condition is true or false and selects the appropriate execution branch; the condition is an OCML relation, namely a *kappa-expression*, which ranges on either outputs of previous activities or inputs of the composite web service orchestrated; loops as *do-while* and *repeat-until* can be represented in terms of the conditional state;
- *fork+join-state*: the fork+join construct consists of concurrent execution of a bag of activities whose results have to be joined.

The data flow, i.e. how the data are used before and after the execution of activities, is defined in the activity classes by means of `has-input-role` and `has-output-role` value slots (fig.4).

```
multiply-activity (activity)
  activity-type :value multiply-goal
  activity-ontology :value wsmo-multiply
  has-input-role :value multiply-input1
                 :value multiply-input2
  has-output-role :value multiply-output
  multiply-input1 :value (wsmo-orchestration-role-value
                           currency-converter-psp-start 'has_amount)
  multiply-input2 :value (wsmo-orchestration-role-value
                           exchange-rate-activity 'has_exchange_rate)
  multiply-output :type number
```

Fig. 4 Multiply activity definition as a goal: the multiply-output slot is a relation for the output data flow in the model; exchange-rate activity definition is similar

We have adopted a consumer-pull convention for data. The data are stored locally and the consumer later retrieves the data needed through a *wsmo-orchestration-role-value* OCML function (fig.4). Web services may have heterogeneous input and output. For their composition either to the binding a mediation mechanism for the data is required. The aforementioned function therefore plays a double role in the model:

- *data binding*: the first argument of the function specifies which activity or state class the current activity relies upon,
- *data mapping*: the second argument of the function specifies the output-role needed and the evaluation of the function assigns a value to the input-role of the current activity.

A composite web service invocation results in instances of the appropriate orchestration classes being created at runtime. The OCML model is interpreted by an orchestration engine written in Common Lisp. The interpreter in a straightforward way reads state by state, instantiates the state it is currently processing and invokes the appropriate handler. At instantiation time the input-roles needed for the current activity are retrieved; each handler is responsible for executing, monitoring the activity and storing its result properly before selecting the next state. The "storage" is represented by an OCML relation of the form `(output-role instance-class-name output-role-value)` asserted as a fact to the orchestration ontology at runtime (fig.4).

4 Conclusions and Future Work

The orchestration engine is stateless as the web services in IRS-III; our model does not satisfy the orchestration requirements outlined in [5] as our intention was first to be able to run a composite web service in IRS-III. Our plan is to investigate more on semantic aspects related on the automatic composition following an approach based on parametric design; to add *fork* and *join* control construct, to make the engine statefull and to integrate orchestration with the choreography model presented in [1].

References

1. Domingue, J., and Galizia, S. Towards a Choreography in IRS-III. Proc. of the Workshop on WSMO Implementations (WIW 2004) Frankfurt, Germany, September 29-30, 2004.
2. Domingue, J., et al. IRS-III: A Platform and Infrastructure for Creating WSMO-based Semantic Web Services. Proc. of the Workshop on WSMO Implementations (WIW 2004) Frankfurt, Germany, September 29-30, 2004.
3. McIlraith, S., Son, T. C., and Zeng, H. Semantic Web Services. IEEE Intelligent Systems, Mar/Apr. 2001, pp.46-53.
4. Motta, E. An Overview of the OCML Modelling Language. KEML98.
5. Peltz, C. Web Service Orchestration and Choreography. IEEE Journal, Computer 36 (10). 46-52 October, 2003.
6. Web Service Modeling Ontology – Standard, available at: http://www.wsmo.org/2004/d2/
7. Orchestration in WSMO, available at: http://www.wsmo.org/temp/d15/v0.1/20040529/

Interactive Composition of WSMO-based Semantic Web Services in IRS-III

Denilson Sell[1,2], Farshad Hakimpour[2], John Domingue[2], Enrico Motta[2] and Roberto C. S. Pacheco[1]

[1]Stela Group and INE, Universidade Federal de Santa Catarina, Brazil
{denilson, pacheco} @stela.ufsc.br
[2]Knowledge Media Institute, The Open University, Milton Keynes, UK
{d.sell, f.hakimpour, j.b.domingue, e.motta} @open.ac.uk

Abstract. The discovery and integration of services in a composition are challenging tasks due to the lack of semantic in the Web services' description. WSMO community is working on developing ontologies and infrastructures to support Semantic Web Services. In this paper, we present a tool that takes into account WSMO descriptions to support a user-guided, interactive composition approach whereby Web services are discovered and recommended to the users according to the composition context. The generated composition is orchestrated in IRS-III by our Java API for dataflow orchestration.

1 Introduction

Research on Web services composition is gaining a considerable attention motivated by the need to support business interoperation and re-use or extension of available services. Challenges related to the Web service composition include constant changes in the business rules, high diversity and heterogeneity of Web services and the ad-hoc character of each composition.

Semantic Web technology can support this complex task, whereby semantic descriptions associated with each Web service can be used to filter and match the services according to the users needs. In particular, IRS-III following the WSMO framework [6], provides at the semantic level a distinction between goals (i.e. abstract definition of tasks to be accomplished) and Web services (i.e. description of services that can achieve a goal) and as a result support capability-driven service matching and invocation [1]. Moreover, the clean distinction between goals and Web services in IRS-III enables the specification of flexible n:m mapping between problems and methods and a dynamic, knowledge-based service selection.

According to [2], the problem of composing Web service can be reduced to three fundamental problems: 1) to prepare a plan dividing a complex task in sub-tasks; 2) discover Web services that achieve the sub-tasks identified in the plan; and 3) monitor and manage the execution and the interaction with the discovered Web Services. The full automation of the Web service composition is still the objective of many ongoing research activities [3], but supporting the user in the definition of the compo-

sition process can achieve accomplishing this objective in a semi-automatic fashion, taking into account user's non-functional expectations on a service composition.

In this paper, we introduce a graphical tool developed in Java that supports users on the definition of dynamic compositions in IRS-III by recommending goals according to the context at each step of a composition. The generated composition is performed by our Java API for orchestration. Our approach is similar to those described in [3], [4] and [5] in the sense that human holds the control of the definition of the composition, but laborious work such as discovery of services according to the users needs is assumed by the machine. However, our approach introduces additional features such as dynamic invocation of Web services in the orchestration, control operator and mediation.

2 Defining a Composition

The Fig. 1 depicts the composition tool and some of its functionalities. The tool guides users in a step-by-step composition process by selecting goals, mediators and control flow operators. The composition starts with the selection of the first goal, when the user receives a list containing all the goals defined in the IRS-III Server. The user can select a goal scrolling the list or use the discovery functionality to search for goals by defining some search criteria, as illustrated in the Fig. 2.

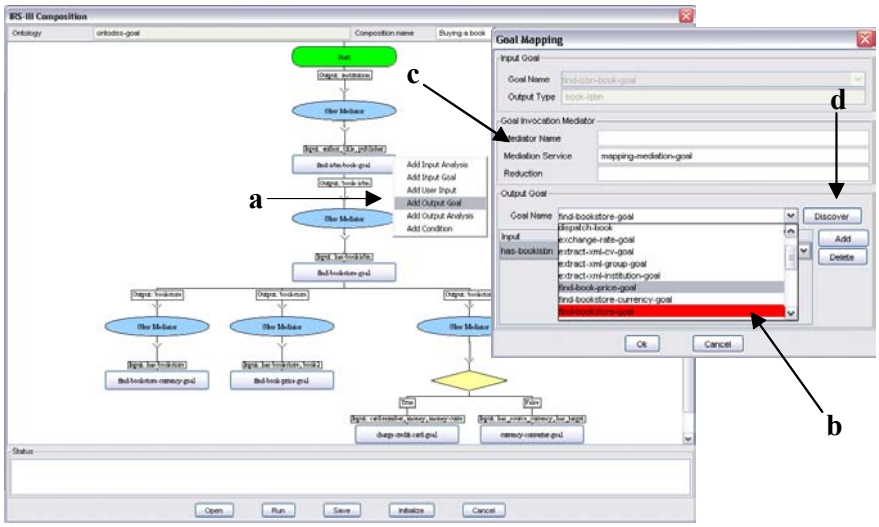


Fig. 1. An example of composition defined in our tool. Users right click over a component and select the desired action (a). In each step of the composition, users receive recommendations of services according to the automatic match of inputs and outputs of goals (b). Users also can define mediators (c) or call the discovery functionality (d).

In the subsequent steps, users can define whether they want to add goals that will receive the result or feed input to the previously selected goals. Each goal can have

more than one feeding source, for instance, a goal that have three inputs can have one input fed by the main flow of the composition and the remaining inputs fed by other goals. Users can also define the values for the inputs of the selected goals in design or orchestration time. Finally, users can add If-Then-Else control operators to the composition. This interactive process is supported by the tool, which in each step recommends goals by matching the inputs and outputs of the goals that were previously selected considering also the subsumption of the input and output types

One important characteristic of our approach is that the tool enables users to select mediators to map and perform transformations between goals. Those mediators (i.e. WSMO Mediators [6]) can solve mismatches between different parties in the data, protocol and process levels. In addition, users can select a Goal invocation mediator (GInv Mediator) that can bind and handle any other transformation required between the inputs and outputs of goals. The GInv mediator is not part of the WSMO specification but specific added to IRS-III to support flexible mappings in our composition model (see [1] for a complete description of the extensions implemented in IRS-III).

The adoption of mediators gives more flexibility to users, since it is inevitable to select services defined and implemented by different parties while building a composition (in fact, this is a basic requirement to support business interoperation). Therefore, we do not restrict the list of goals that the user can select in each step of the composition, allowing users to define mediators that could perform required transformations between goals.

Fig. 2. The Goal Discovery functionality. Users can define search criteria using a logical operator and identifying properties and correspondent values. The search criteria is translated to an OCML expression, processed against the IRS-III Server and presented to the user.

3 Orchestration of Composition

Once a composite service has been defined, the composition tool instantiate the workflow using our Java API for orchestration. In this process, the tool instantiates the service components and control operators defined in the composition according to their data dependencies using the constructors defined in our API for orchestration. The API offers necessary features to build, validate and write a composite service to

IRS Server, as well as, loading a composition from the server and editing it. The saved descriptions can be executed by the orchestration engine included in the API.

The API offers three categories of components to support compositions, namely service components, control components and mediators. A service component is actually a wrapper that keeps the necessary information about the goal to be achieved and its binding mediators. The control components provide the capability to define the control flow through the If-Then-Else operator. The mediator components bind the service components and point to WSMO mediators described in IRS-III Server for any data transformation required between service components.

The order of the execution will depends on the data provided to a service component at the execution time and it will not be defined at the design time. A service starts to execute when the necessary data is provided for its inputs. For a stateless service, that means, if all inputs to the service are provided it will be executed. The necessary means to define mediators are provided just as described above.

During the orchestration, the user is requested to enter values to feed input to the goals where the inputs were not specified in design time and that are not fed by other goals. The orchestration API relies on the IRS-III Server to achieve each goal defined in the composition, which in turn, dynamically discovers the most appropriated Web services that should be invoked according to their applicability conditions. Users can monitor the status of the orchestration by examining the status bar provided in the composition tool.

Acknowledgement

This research is supported in part by CNPq, Brazil, in the form of a scholarship held by Mr. Sell. This work is also supported by the DIP (Data, Information and Process Integration with Semantic Web Services) and AKT (Advanced Knowledge Technologies) projects.

References

1. Domingue, J., Cabral, L., Hakimpour, F., Sell, D. and Motta, E. IRS-III: A Platform and Infrastructure for Creating WSMO-based Semantic Web Services. In: Proceedings of the Workshop on WSMO Implementations (WIW 2004) Frankfurt, Germany, September 29-30 (2004).
2. Sycara, K., Paolucci, M., Ankolekar, A.; Srinivasan, N.: Automated Discovery, Interaction and Composition of Semantic Web Services. In: Journal of Web Semantics, Vol. 1, Issue 1 (2003).
3. Sirin, E., Parsia, B., Hendler, J. Filtering and selecting semantic Web services with interactive composition techniques. In: IEEE Intelligent Systems, Vol. 19, Issue 4 (2004) 42-49.
4. Kim, J., Spraragen, M., Gil, Y. An Intelligent Assistant for Interactive Workflow Composition. In: Proceedings of the International Conference on Intelligent User Interfaces (IUI-2004) Madeira, Portugal, 2004.
5. Ponnekanti, S. R., Fox, A. SWORD: A developer toolkit for web service composition. In: The Eleventh World Wide Web Conference, Honolulu, HI, USA, (2002).
6. WSMO Web Service Modeling Ontology – Standard, http://www.wsmo.org/2004/d2/

SWSDesigner: The Graphical Interface of ODESWS

Asunción Gómez-Pérez[1], Rafael González-Cabero[1], Manuel Lama[2]

[1]Departamento de Inteligencia Artificial, Facultad de Informática.
Campus de Montegancedo s/n, Universidad Politécnica de Madrid, 28660 Boadilla del Monte, Madrid. Spain.
asun@fi.upm.es, rgonza@delicias.dia.fi.upm.es

[2]Departamento de Electrónica e Computación, Facultad de Física.
Campus Sur s/n, Universidade de Santiago de Compostela, 15782 Santiago de Compostela, A Coruña.
lama@dec.usc.es

Abstract. ODESWS is a development environment to design Semantic Web Services (SWS) at the knowledge level. ODESWS describe the service following a problem-solving approach in which the SWS are modelled using tasks, to represent the SWS functional features, and methods, to describe the SWS internal structure. In this paper, we describe the ODESWS graphical interface (called SWSDesinger). This interface enables users to design SWS independently of the semantic markup language in which the service will be implemented, and once the design has been export the service to an SWS implementation language.

Introduction

Currently, there are some proposals to edit/design SWS, but the main drawback of these available editing tools is that they work at the representation level. Cosecuently, these tools are language-dependent, like the WSMO Editor [1], this means that: (1) SWS designed with these tools are less reusable; (2) the designs are more prone to inconsistencies or errors; (3) the design can be constrained with the chosen language characteristics. Also, many tools that claim to be SWS editors are no more than ontology editors, like the widely used option of OWL-S development with the OWL plug-in for Protegé-2000 [2]. This option not only suffers from all the problems enumerated above, but also adds the problem of working with an ontology instantiation, not with a SWS-like structure.

To solve these problems, we have proposed a framework, called ODESWS[3], for the design of SWS at the knowledge level, which is language-independent. This framework is based on: (1) a stack of ontologies that describe explicitly the different features of SWS; (2) a set of axioms used to check the consistency and correctness of the service descriptions; and (3) the assumption that a SWS are modeled as a problem-solving method (PSM) that describes how the service is decomposed into its components, and which is the reasoning process that describes the service execution.

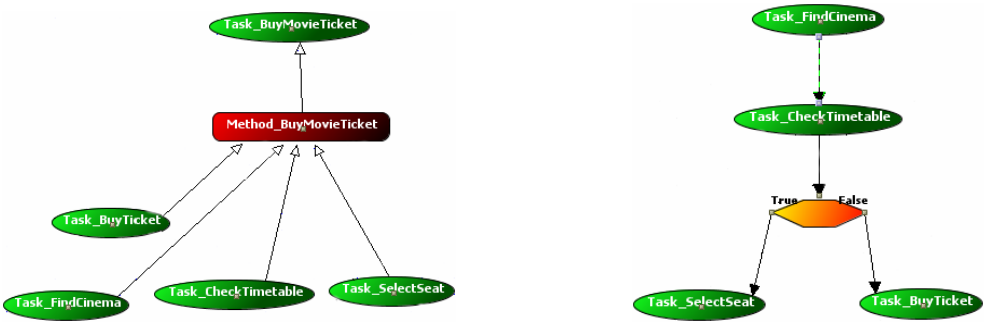


Fig. 1. A method (a) is composed of a set of sub-tasks, which are solved by other methods; and (b) defines the coordination of the execution of the sub-tasks (usually with workflows).

We also have implemented this framework, creating the ODESWS environment [3][4].

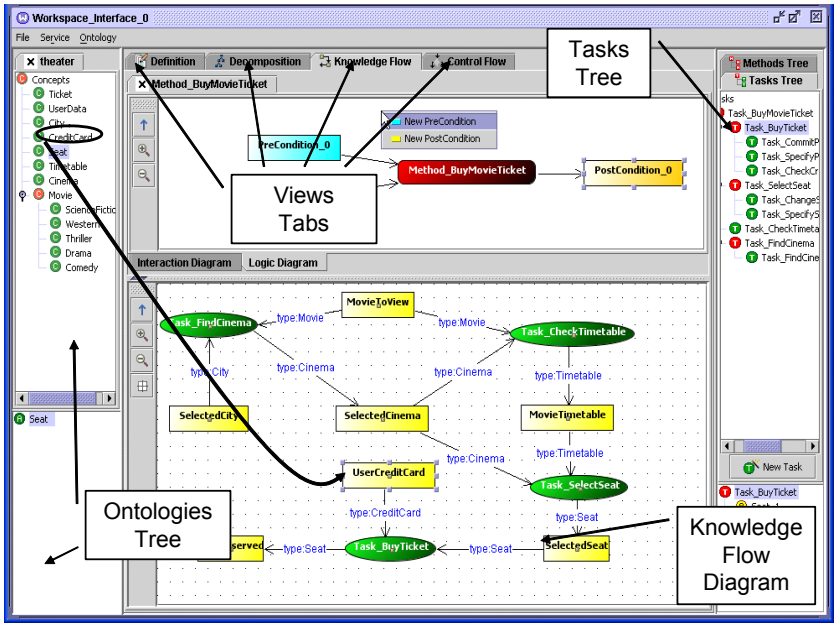
In this paper we describe SWSDesigner, the user interface of the ODESWS environment.

SWSDesigner, the ODESWS graphical interface

The design of SWSDesigner has been inspired in the classical modelling of the problem-solving methods, so it contains hierarchical trees of tasks-methods, input/output interaction diagrams among the sub-tasks that compose a method, and diagrams to specify the control flow that describes the coordination of the execution of the sub-tasks. Taking this into account, in the SWSDesigner we distinguish the following general components:

- *Trees* show the hierarchy of the knowledge components needed to define the service, such as tasks, methods, and ontologies.
  - *Tasks and methods trees* (right part of Figure 2) allow users to just create the tasks and methods associated with a service, describing its functional features in the case of tasks, and internal structure in the case of methods. Once tasks and methods have been created, from these trees the user could drag the icons representing a task or method and drop them in the diagrams as needed.
  - *Ontology trees* (left part of Figure 2) show the concepts and attributes of the ontology (or ontologies) used to specify the service input/output roles. Then, the user could drag the icons representing a concept/attribute and drop them in the diagrams that enable the specification of the input/output roles of both tasks and methods.
- *Views* allow users to specify all the features of a service, and they are represented as *tabs* (see the upper part of Figure 2):
  - *Service definition View* is used to specify the functional and non-functional features of a service, containing its input/output roles (interaction diagram), pre/post-conditions (logical diagram), providers, commercial classification, geographical location, and quality rating parameter.

Fig. 2. Knowledge Flow View of a method in SWSDesginer.



- *Decomposition View* (Figure 1 a) allows users to specify the decomposition of the method (that solves the task associated with the service) into its sub-tasks, which will be solved by other methods, and so forth. The user carries out this specification by dragging the icons of the tasks and methods from the related trees and dropping such icons into the view.
- *Knowledge Flow View* allows users to define the input/output interactions among the sub-tasks of a method: the user, when requires, connects the output of a sub-task to the input of other sub-task, and establish the *mappings* between the roles used in the definition of a sub-task (carried out in the service definition view) and the roles named when such sub-task is used as an internal component of a method. For example, *City* could be an input role of the task *Task_FindCinema* and when that task is used as part of the method *Method_BuyMovieTicket*, its input role could be named as *selectedCity*. In this view, the user also defines the *mappings* between the roles of a method and the roles of the task solved by that method: the role names of methods and tasks could be different.
- *Control Flow View* (Figure 1 b) enables users to describe the control flow of the method. The elements of this view are the sub-tasks of the method, which are dragged-and-dropped from the task tree, and the workflow constructions (if-then, while-until, split and join), which are introduced through a contextual menu.

Once the user has designed the service following all the complementary views provided by the SWSDesigner, it is necessary to translate that service from the graphical representation into a semantic-oriented language such as OWL-S or WSMO. To enable this translation, the SWSDesigner invokes the SWSInstanceCreator execution, which uses the graphical model of SWS to create the instances of the

SWS description ontologies. Then, SWSTranslator is invoked to translate these instances into the language selected by the user (currently OWL-S).

Conclusions

The tools that currently enable users to design SWS depend on the capabilities of representation and reasoning of a specific SWS-oriented language. Those tools are constrained by the language expressiveness; the service must be designed using the capabilities provided by the language in which will be expressed. Furthermore, users usually introduce both errors and inconsistencies, which could be minimized using tools that operate at the knowledge level.

To solve these problems we provide tools to facilitate the design SWS in a language-independent manner. For it, we have developed the ODESWS conceptual framework and the environment that supports such framework. Once the service is completely designed using SWSDesigner it will be checked to detect inconsistencies and/or errors that could be present in the user design. If they are not detected, user will select the language in which the SWS will be expressed, and it will be automatically created.

Acknowledgements

This work has been partially financed by the Esperonto project (IST-2001-34373) and by a grant provided by the Community Autonomous of Madrid.

References

[1] Lausen, H., Felderer, M., and Roman, D., eds. (2004), "Web Service Modeling Ontology (WSMO) Editor", Available: http://www.wsmo.org/2004/d9/v01

[2] Holger Knublauch, Ray W. Fergerson, Natalya F. Noy, Mark A. Musen (2004) The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications Third International Semantic Web Conference - ISWC 2004

[3] A. Gómez-Pérez, R. González-Cabero and M. Lama (2004): ODE SWS: A framework for designing and Composing Semantic Web Services. *IEEE Intelligent Systems*, 19(4):24-31.

[4] O. Corcho, M. Fernández-López, A. Gómez-Pérez, and M. Lama (2003): An environment for Development of Semantic Web Services. Proceedings of the *IJCAI-Workshop on Ontologies and Distributed Systems*, Acapulco, Mexico.

[5] A. Gómez-Pérez, R. González-Cabero, and M. Lama (2004): Development of Semantic Web Services at the Knowledge Level. *ECOWS*. Erfurt, Germany.

Developing a Service-Oriented Architecture to Harvest Information for the Semantic Web

Barry Norton, Sam Chapman and Fabio Ciravegna

Department of Computer Science, University of Sheffield
Regent Court, 211 Portobello Street, Sheffield S1 4DP, UK
{B.Norton, S.Chapman, F.Ciravegna}@dcs.shef.ac.uk

Abstract. Armadillo is a tool that provides automatic annotation for the Semantic Web using unannotated resources like the existing Web for *information harvesting*, that is: combining a *crawling* mechanism with an extensible architecture for *ontology population*. The latter is achieved via largely unsupervised machine learning, boot-strapped from oracles, such as web-site wrappers, and backed up by an 'evidential reasoning', allowing evidence to be gained from the redundancy in the Web and allowing the inaccuracies in information, also characteristic of today's Web, to be circumvented. In this paper we sketch how the Armadillo architecture has been reinterpreted as workflow templates that compose semantic web services and show how the porting of Armadillo to new domains, and the application of new tools, has thus been simplified.

1 Introduction

The Semantic Web needs semantically-based document annotation to both enable better document retrieval and empower semantically-aware software agents. Most of the current technology is based on human centred annotation, very often completely manual which can be incorrect and incomplete, deliberately or through lack of skill, or can become obsolete. A major advantage of Armadillo [2] is its ability to annotate large repositories in a largely unsupervised manner and thereby make available to the Semantic Web the huge amount of information, available only in human-oriented form, on the current Web.

Armadillo annotates by extracting information from different sources, boot-strapped by 'oracles' i.e. relatively infallible authorities. 'Evidential reasoning' is used to validate the classifications of, and relations between, instances. This evidence is then integrated and the knowledge entered into a repository.

Our aim in using a semantic web service (SWS)-based architecture is to allow porting to a new domain by providing workflow templates, *i.e.* where some subtasks are left as parameters, expressed in BPEL [5], where SWS's achieving these subtasks are described in OWL-S [4]. We claim that this allows the process to be understood abstractly, and represented graphically rather than in code, and that those points where services must be located and 'plugged in' are more easily understood in terms of the OWL [8] concepts they relate. Furthermore direct reuse of domain-specific and externally-authored functionality is facilitated.

2 Architecture

The act of porting Armadillo to a new ontology population task begins by providing a domain-specific ontology. The given ontology includes structural relationships which are transformed into a plan for population. The plan details an order in which the concepts and relations will be explored by attaching a direction to each relation. For each relation to be followed, from a concept A to a concept B, the workflow shown in Figure 1 organises the following subtasks: Crawling, Instance Recognition, Evidential Reasoning and finally Combination and Storage. Crawling will systemically retrieve documents associated with an instance of concept A; Instance Recognition will find candidate instances of B with an implicit relation to that instance; Evidential Reasoning will find support for both the classification and the relation discovered, which will be combined and, if sufficient, cause both to be stored.

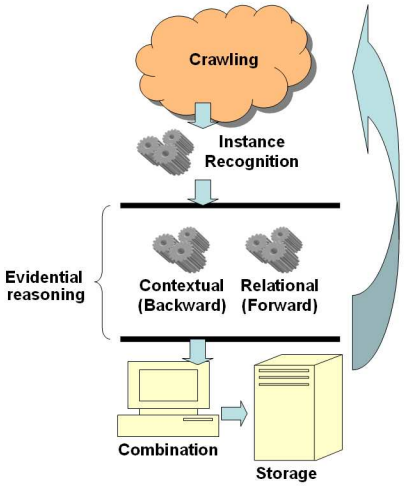


Fig. 1. Architecture

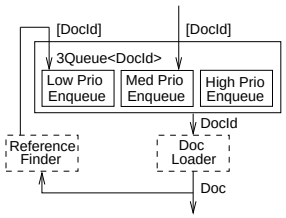


Fig. 2. *Crawling* Task

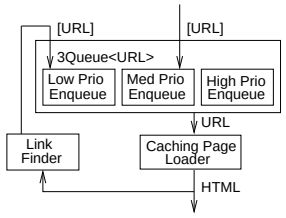


Fig. 3. Example *Crawling* Instantiation

In the remainder of the paper we detail the template workflows which achieve each of these subtasks in turn, composing generic services with others that a developer must locate to accomplish clear strategies. For these the choice, aided by *semantic discovery* [7], is between:

- *generic* services: wholly independent of domain and context;
- *context-dependent* services: where reuse depends on the context but not the domain (*e.g.* the type of documents or repository used);
- *domain-tailored* services: parameterised to be tailored to a given domain;
- *domain-trained* services: encapsulate machine-learning which can adapt semi-automatically to a given domain;
- *domain-specific* services: encapsulate techniques which are hard-coded for a given domain (but might still be re-used across applications in that domain).

In the following we shall use a familiar example of the academic domain [2], wherein we view instances of the 'University' concept as being provided by an oracle and populate the 'Academic' concept, via the 'employedBy' relationship. We use square brackets to represent lists over the contained type, and so $[DocId]$ is the type of a list of elements of type $DocId$. Angle brackets mean instantiation of some generic (parameterised) service, so that $3Queue$ represents some generic queue service that stores instances of some type notated $DocId$, at three different priority levels, and supplies them to the consequent workflow one at a time.

2.1 Crawling

The general form of the 'crawling' task is shown in Figure 2. There are two levels at which this abstract workflow is parameterised: the $DocId$ and Doc labels are actually type variables that must be instantiated at concrete types; the $DocLoader$ and $ReferenceFinder$ tasks are service variables to be instantiated.

Figure 3 shows how we could achieve a concrete instantiation of this task. First we choose instances for the type variables consistent with Web-oriented technology, i.e., in this example, $Doc = HTML$ and $DocId = URL$. We then choose *context-dependent* services that meet the resulting signatures, i.e. producing a page from a URL, and a list of URLs from a page, respectively.

2.2 Instance Recognition

The 'instance recognition' task can be realised by both *domain-tailored*, and *domain-trained* services to find candidate instances of a given type B with an implicit relation to the instance of A being investigated. This can be seen in Figure 4, where the $B-Recogniser$ outline implies that multiple parallel implementations may be used. For instance in the example in Figure 5 we see that both *domain-tailored* regular expression matching and *domain-trained* Amilcare [3] will be used side-by-side, as could other Information Extraction tools.

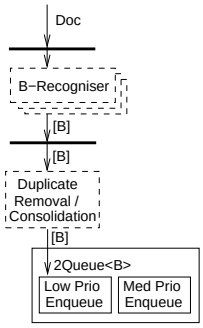


Fig. 4. *Instance Recognition* Task

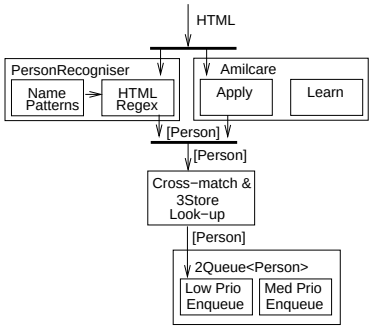


Fig. 5. Example *Recognition* Instantiation

The subsequent consolidation stage is typically domain-specific Information Integration but reuse can be made; for instance from the 'similarity metrics' library, *SimMetrics*, which we have recently released [1]. The consolidated list of candidate instances is then queued to be validated by 'evidential reasoning'.

2.3 Evidential Reasoning

For each potential instance of the concept B from the queue this task will attempt to find evidence to confirm its classification and relation to the original instance. This may consist of several parallel services, each of which implements a strategy that falls into one of two classes, as follow.

Contextual Reasoning considers each potential B instance in the context of the A instance via which it was discovered. As seen in Figure 6, two services will be used to find occurrences of the B instance in general and co-located with the A instance respectively. A third service then produces a list of potential triples relating these instances, with evidence supporting the relation. Figure 7 shows a simple instance of this strategy where we 'promote' the candidate instance to being an academic employed by the university based on co-located references on the web, obtained by a Google wrapper which is domain-independent.

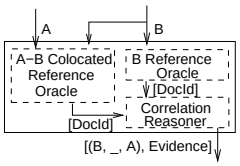


Fig. 6. *Contextual Reasoning* Task

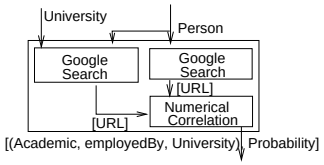


Fig. 7. Example *Contextual* Instantiation

Relational Reasoning provides evidence for the candidate B instance being correctly classified as such, based on other relations an oracle may find. In this subtask, as shown in Figure 9, we may apply *domain-tailored* or *domain-specific* technologies such as gazetteers and site wrappers, as well as *domain-trained* relation extraction, as we are developing in the tool *T-Rex*.

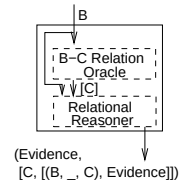


Fig. 8. *Relational Reasoning* Task

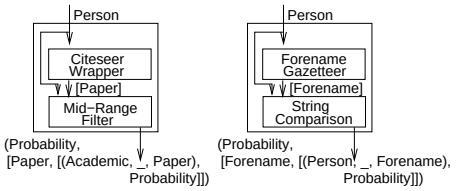


Fig. 9. Example *Relational* instantiations

2.4 Combination and Storage

For the sake of brevity we do not consider, in this paper, the 'combination' and 'storage' tasks except to say that we have instantiated the *Evidence* type variable as a probability and will use a statistical combination to provide a higher-confidence result; other solutions are also accommodated.

3 Conclusions

We have illustrated how it is possible to make use of SWS's in harvesting the Web, and other corpora, to provide annotations for the Semantic Web. The architecture presented is based on workflow and follows an IE-oriented strategy. Initial approximations to both classification and implicit relation extraction are followed by evidential reasoning based on both context and further relations. In this way a wide variety of semantic web services may be accommodated and porting is eased since, in many cases, users can avoid coding altogether, merely using the workflow templates to guide semantic discovery and composition.

This architecture also provides many other benefits associated with service-oriented architectures, such as speed-up from concurrency and distribution, an automatic means to reuse of any code that does have to be written specifically for a new domain, and the ability to provide services remotely to users with little infrastructure. A more complete picture of the workflow template described here, and the means by which it is currently implemented, is provided in the related position paper [6].

References

1. Sam Chapman. SimMetrics. http://sourceforge.net/projects/simmetrics/.
2. Fabio Ciravegna, Sam Chapman, Alexiei Dingli, and Yorick Wilks. Learning to harvest information for the semantic web. In *Proceedings of the First European Semantic Web Symposium*, May 2004.
3. Fabio Ciravegna and Yorick Wilks. *Annotation for the Semantic Web*. Series Frontiers in Artificial Intelligence and Applications. IOS Press, 2003.
4. Anupriya Ankolekar *et al.* DAML-S: Web service description for the semantic web. In *Proc. 1st International Semantic Web Conference (ISWC)*, 2002.
5. IBM *et al.* Business process execution language for web services version 1.1. http://www-128.ibm.com/developerworks/library/ws-bpel, 2003.
6. Barry Norton. Proposed functional-style extensions for semantic web service composition. In *Proc. 1st AKT Workshop on Semantic Web Services*, 2004.
7. Naveen Srinivasan, Massimo Paolucci, and Katia Sycara. Adding OWL-S to UDDI: implementation and throughput. In *Proc. 1st Intl. Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, pages 6–9, 2004.
8. W3C. OWL web ontology language overview. http://www.w3c.org/TR/owl-features, 2004.

Acknowledgements

This work was carried out within the AKT project (http://www.aktors.org), sponsored by the UK Engineering and Physical Sciences Research Council (grant GR/N15764/01), and the Dot.Kom project, sponsored by the EU IST asp part of Framework V (grant IST-2001-34038).

Integrating Preferences into Service Requests to Automate Service Usage*

Michael Klein[1] and Birgitta König-Ries[2]

[1] Institute for Program Structures and Data Organization, Universität Karlsruhe, 76128 Karlsruhe, Germany, kleinm@ipd.uni-karlsruhe.de
[2] Institute of Computer Science, Friedrich-Schiller-Universität Jena, 07743 Jena, Germany, koenig@informatik.uni-jena.de

1 Introduction

Today, web services are often used as a technology to integrate functionality of different entities. However, one important potential of service oriented computing is not exploited: the ability to form agile networks. Here, service requestors and service providers are not fixedly tied together, rather, bindings to inefficient or unavailable service providers are transparently replaced by bindings to more appropriate providers at runtime. In such an architecture, the robustness and efficiency would be increased dramatically.

The main reason why these networks are not a reality today is that current technologies do not allow for automatic service selection and invocation; rather, they require human interaction to decide on an appropriate service provider. Obviously, this approach is not feasible in a system where service selection needs to be carried out repeatedly at run time.

The most challenging prerequisite for automatic service description is an appropriate service description language. This language needs to be able to capture service offers and requests in sufficient detail to allow for automatic matchmaking. In this paper, we argue that such a language needs to explicitly incorporate user preferences into service requests.

2 Problems with the State of the Art

Many existing languages for service description use the same technique for describing requests and offers: the requesting application describes its desired functionality by specifying an instance of the "perfect" service. A generic matchmaker compares this request to the published offer descriptions and calculates a similarity as a value from the interval$[0.0, 1.0]$ by using heuristical structural and/or semantical similarity metrics. The service offer with the highest similarity is invoked directly by the service requestor.

Much effort has been put into the development of intelligent similarity calculators. The most important approaches perform a comparison of the functional parameters, perform a structural comparison of the description graphs, use logic subsumption or use combined approaches [2–5].

* This work is partially funded by the Deutsche Forschungsgemeinschaft (DFG) within SPP 1140. Some of the ideas in this paper have been published in [1].

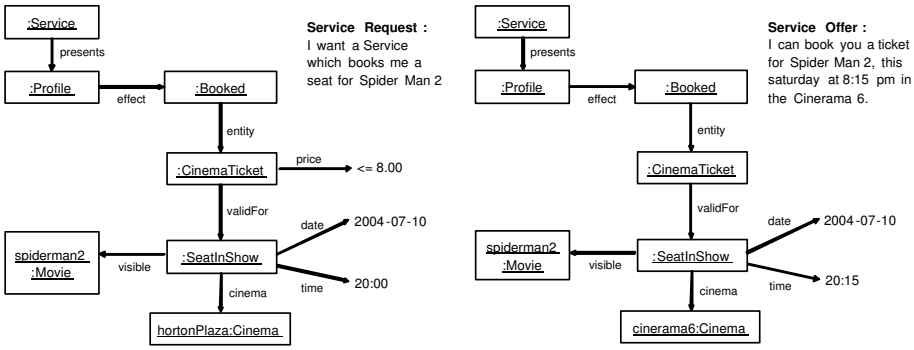


Fig. 1. Request description as perfect service (left) and offer description (right).

However, these approaches only work well, if offer and request description are exactly equal, so the matcher returns 1.0, or obviously different, so the matcher returns 0.0. However, in intermediate situations, in which the offer differs somewhat from the request, it becomes very difficult for the matcher to assign the value from $(0, 1)$ that is appropriate, i.e., that reflects the requestor's perception of the usefulness of the service offered.

An example shall illustrate this. The request on the left of Figure 1 is specified as one single instance[3], which represents the requestor's ideal service. He wants to invoke a service reserving a seat for Spiderman 2. He also gives some information about his perfect reservation: It should be in the cinema Horton Plaza, at a given date and time and the ticket's price should be 8 Euro or less.

On the other hand, we have a service provider which offers a nearly matching service offer (see Figure 1 (right)). He offers to book a ticket for Spiderman 2, but differs in some of the requested attributes[4]. The matcher now has to decide:

- The requestor wanted 20:00 as starting time, but the offered service can only reserve a ticket for 20:15. Is this still a match or only a 90% match?
- The requestor wanted the Horton Plaza cinema, but the offer is a about the cinema Cinerama 6. Is this still ok because they are in the same city[5]?
- The requestor wanted a price below 8 Euro, but the offer didn't mention the price. Is this a matching value of 0.0, or should some other value be assigned?
- What is more important for the offerer: A good price, a good time, a near cinema? The matcher has to decide whether to take the average of the individual matching values, their minimum or another function.

As we can see, the main problem lies in the fact that the preferences of the requestor are not clear as they are not explicitly specified anywhere. This matchmaker has to either use general, domain- and user-independent deviation heuristics or simply perform a very strict, conservative matching. Each of these

[3] We use a graphical, UML-like notation of the description.

[4] Realistically, the provider would offer a more generic service like the booking of arbitrary movie tickets. Our approach can handle this by introducing variables [1].

[5] This information could be provided in the underlying ontology.

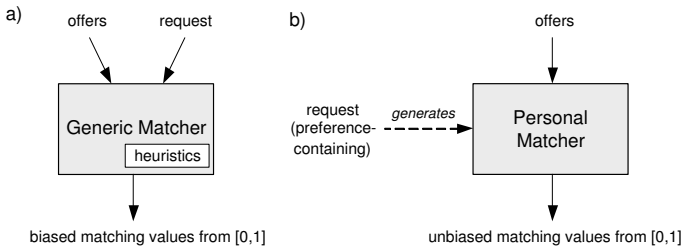


Fig. 2. Generic vs. personal matcher.

approaches leads to a biased matching process, i.e. the result of the match depends on the matchmaker used. Thus, the requestor typically will not blindly rely on its result but will want to choose one of the proposed services manually. Thus, automatic service invocation is prevented.

3 Approach: Preference Integration

As shown in Section 2, only an unbiased matching process could be accepted within an automatic service usage process. Such an unbiased matcher can only become a reality, if the service request contains enough information for the matcher to decide in deviation cases. This means, that the service request has to include the client's preferences.

Figure 2 illustrates this idea. We have to overcome the approach of a generic, all-purpose matcher (left side). Such a matcher is biased. It is not able to calculate reasonable matching results for service descriptions from arbitrary application domains that can be used to automatically choose an appropriate service for the requestor. Instead, we need *preference-containing* request descriptions that can be used to generate a highly specialized, personal matcher (right side). Such a matcher would be unbiased so that the requestor would agree to automatically invoke the best matching service.

To achieve this goal, we propose to express requests as fuzzy declarative sets of suitable services rather than as one specific instance representing the perfect service. The degree of membership of a service offer to the fuzzy set described in the service request expresses the requestor's preference for this offer and also its matching value. Requests are build up by using a limited set of well-defined constructors, which leads to structured and computationally feasible service requests. Consider as an example the request shown in Figure 3. Here, sets are depicted as rectangles with a small cross line in the left upper corner.

Again, the requestor is looking for a ticket for Spiderman 2. In her request, she specifies that she is not willing to see another movie or to see the movie on another than the specified date. However, she is willing to accept a slightly later or earlier time than her preferred starting time (expressed by the `~==[15min]20:00` condition for the time property of SeatInShow) and is also willing to attend a show in a theater close to the one that is her first choice (expressed by the similarity function `near`. Here, either a predefined function from the ontology

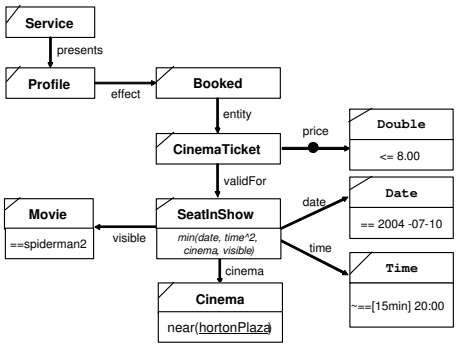


Fig. 3. Preference-containing request.

can be used or the user specifies her own function). While she would prefer not to pay more than 8 Euros, offers that do not specify a price should also be considered as possible matches (based on the real world experience of the user that cinema tickets seldom cost more than eight Euros). This is expressed by the *missing strategy* `assume_fulfilled`, which is depicted by the circle. The *connecting strategy* `min(cinema,visible,date,time^2)` expresses how the individual matching values should be combined. Here, the requestor has stated that the result has to be calculated by minimizing the results from the single conditions where the time attribute is emphasized by the exponent 2. As a result, this is a conjunctive connection. Given all this information, it is straightforward to generate a personalized matcher that will be able to determine exactly how well a service offer fulfills the requestor's needs [6].

Our service description language, DIANE Service Description (DSD) [1, 7], offers the means to express such requests. DSD and the corresponding matcher have been implemented.

References

1. Klein, M., König-Ries, B.: Combining query and preference - an approach to fully automatize dynamic service binding. In: Short paper at IEEE International Conference on Web Services (ICWS 2004), San Diego, CA, USA (2004)
2. Paolucci, M., Kawmura, T., Payne, T., Sycara, K.: Semantic matching of web services capabilities. In: Proc. of the Semantic Web Conf., Sardinia, Italy (2002)
3. Trastour, D., Bartolini, C., Gonzalez-Castillo, J.: A semantic web approach to service description for matchmaking of services. In: Proc. of the Intl. Semantic Web Working Symposium (SWWS), Stanford, CA, USA (2001)
4. Li, L., Horrocks, I.: A software framework for matchmaking based on semantic web technology. In: Proc. of the Intl. WWW Conference, Budapest, Hungary (2003)
5. Sycara, K., Widoff, S., Klusch, M., Lu, J.: Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. Autonomous Agents and Multi-Agent Systems **5** (2002) 173–203
6. Klein, M., König-Ries, B.: Coupled signature and specification matching for automatic service binding. In: Proc. of ECOWS 2004, Erfurt, Germany (2004)
7. Klein, M., König-Ries, B., Müssig, M.: What is needed for semantic service descriptions? Intl. Journal on Web and Grid Services (2005). Submitted for publication.

OCML Ontologies to XML Schema Lowering *,**

Vlad Tanasescu[1,2], John Domingue[2], Liliana Cabral[2]

[1] Swiss Federal Institute of Technology (EPFL), 1015 Lausanne, Switzerland
vlad.tanasescu@epfl.ch
[2] Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK
{v.tanasescu, j.b.domingue, l.s.cabral}@open.ac.uk

Abstract. Ontologies are explicit specifications of a conceptualization intended for logical processing. They are used to express meaning and to apply it on otherwise less structured data. Nevertheless other models of organization and structure of knowledge, like database models and XML document definition standards, are widely used and valuable. In order to extend ontology usage to multiple layers of an application, or to integrate ontologies with pre-existing software, the use of these various means to structure data could be necessary. We are therfore using metadata in order to adapt an ontology to these use cases and provide the example of automatically generating user interfaces for goal descriptions in IRS-III [1] by extracting an XML schema from an ontology.

1 Introduction

The semantic web vision foresees the advent of automatic machine operable services through the use of knowledge based technologies. An interesting class of standardized services are *web services*, which are widely accepted if not yet massively used by the industry. Current web service implementations are, however, relatively inflexible, and not powerful enough to support automatic discovery, mediation and composition. Therefore ongoing research is investigating how semantic web technology can alleviate this. The IRS-III (Internet Reasoning Service) framework [1] supports the creation of semantic web services according to the WSMO ontology [2], and extends it. Users of IRS-III directly invoke web services via *goals* i.e. IRS-III supports *capability-driven* service execution. A goal is described as a class in OCML (Operational Conceptual Modelling Language) [3] and related to one or more web services through mediators, which provide

* This research was partially supported by the Advanced Knowledge Technologies (AKT) project. AKT is an Interdisciplinary Research Collaboration (IRC), which is sponsored by the UK Engineering and Physical Sciences Research Council under grant number GR/N15764/01. The AKT IRC comprises the Universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University.

** This work was also partially supported by the DIP (Data, Information and Process Integration with Semantic Web Services) project. DIP (FP6 - 507483) is an Integrated Project funded under the European Unions IST programme.

the required flexibility by allowing to specify mapping mechanisms between goal input roles, output roles, and matching web services.

Ontologies, like the ones used in a goal description to define a capability, are intended for logical inference, and are often loosely related to applications based on them; therefore one has to provide some *glue* to allow diverse aspects of an application to conform to an ontology; the problem of data consistency, of user input for example, usually occurs and is hard to tackle in a generic way.

Our motivating example is to provide a consistent interface to IRS-III goals. The input to these goals, i.e. the data one has to provide in order to expect achievement of the goal, is defined in an OCML ontology, and uses data types provided by it. For example an *exchange rate* goal will have three input roles: *has-source-currency*, *has-target-currency* and *has-amount*, which represents the amount which has to be converted from source to target currency.

The difficulty is that OCML types are used to describe the roles, i.e. *pound* or *euro* are instances of a *currency* class. The enumeration of available currencies is therefore well defined in the ontology and this constraint has to be reflected in the actual web or Java goal access interfaces, task which is actually achieved by the developer, in a disconnected and hence highly error-prone way.

2 Translating OCML ontologies to XML schema

The relation between ontologies and XML schemas (XSD [5]) has already been described (for example in [4]): ontology languages are a means to specify domain theories while XML schemas provide integrity constraints for information sources, but both provide vocabularies and structure for describing information sources that are intended for exchange. However ontology modelling languages like OCML do not offer rich collections of built-in data types for two main reasons:

1. Providing clear semantics and reasoning support for a large collection of complex data types is difficult.
2. The precise representation of a data type is often superfluous in a knowledge modeling context, i.e. a *date* may be an important aspect of a domain but various representations of it are not.

The XML schema type system has been inspired by language independent data types as well as actual query and object-oriented programming idioms like SQL or JAVA [5]. OCML is a frame-based, i.e. object-centered, knowledge representation system, which also provides a relational view. These systems therefore present similarities that we can exploit and a quite intuitive mapping appears between the two (see table 1).

Unfortunately the XSD type system does not support multiple inheritance, and even simple derived types can be only created *either* by *restriction* or by *extension* but not both at the same time. However this kind of modelling is mostly useful for knowledge representation and have usually little role to play when interacting with other software architectures. Moreover we are only interested

Table 1. OCML to XSD data types

OCML	XSD
class	complexType
slot	element
string, float, ...	simpleType

by the *constraints* allowed by an XSD definition, not by the reuse of the derived XSD types since we are able to generate them on the fly. Therefore we chose to treat these cases by defining a new type for every OCML class, inherited or not, containing all the slots (local and inherited) as elements. Also there is actually no planned support in our approach for classes defined in OCML *intensionally* by logical expressions.

Still the result of this simple mapping is not very helpful from an applicative point of view since we do not have any validation constraints yet, i.e. we do not know, because it is not specified in the ontology, that for our purposes the slot/element called *has-amount* must be smaller than a given value (otherwise the program may crash, or the database will not be able to store the value, etc).

3 Constraint metadata applied to ontologies

Constraint information could simply be added to the ontology, however the applicative environment (Java program, web interface, database) may change while the ontology is already shared in multiple contexts, therefore it is more appropriate to only *attach* this information to the ontology, without any modification, hereby allowing for multiple *views* or *aspects* of it depending of the applicative context. Without doing so constraints on a goal could be superfluous or even conflicting depending on the context, e.g. the maximum value of a number could be greater for a Java application than for a Web interface, while entirely meaningless for the knowledge domain.

During the translation process we therefore use a metadata system to store the constraints we chose to add to the ontology XSD representation for our specific context dependent purpose. Our metadata system is composed of:

1. A *naming convention* mimicking the *has-a* tree-like structure of the ontology and allowing to request information about a particular element of it (class, slot or relation). For example:

   ```
   '(transform classes EXCHANGE-RATE-GOAL HAS-AMOUNT)
   ```

2. A *repository* which can be anything from a DBMS to simple association lists, able to store the information regarding a node according to the naming structure, for example the required length of a string, as well as other relevant information like OCML basic types mappings to XSD:

```
'((transform ((classes ((EXCHANGE-RATE-GOAL
                          ((HAS-AMOUNT
                            ((maxInclusive ((1000000)))))))))
              (types ((float (("xs:decimal")))))))
```

3. An *access interface* to the metadata repository that has to provide necessary access functions as well as commodity ones required by the generation program.

By using the metadata convenience (through the repository called *mddb*) we can now easily map an OCML ontology to a constrained XSD schema:

Algorithm *OCML2XSD*(*ontology*, *mddb*)
1. **for** $c \in classes(ontology)$
2. **do** Create element *complexType* with attribute $name = name(c)$
3. Create *sequence* element
4. **for** $s \in slots(c)$
5. **if** s is a type referenced as *basic* in *mddb*
6. **then** create element *simpleType*
7. add *base* attribute with value specified in *mddb*
8. **if** s has attached constraints in *mddb*
9. **then** add the constraints as *restriction* elements
10. **else** create element with *type* attribute set to $class(s)$

It is straightforward to restrict this algorithm to the slots representing input roles of an IRS-III goal. Then, by applying an XSL transformation to the generated schema, we obtain a web interface with embedded validation.

4 Conclusion

We demonstrated the use of metadata applied to ontologies in order to adapt them to practical application contexts, and used this technique to generate XML schema from an OCML goal description in order to obtain a consistent validating web interface. Metadata can be used to produce many applicative *views* of an ontology, like SQL Data Definition Language statements for persisting ontology instances, or JavaBeans template code to ease ontology driven enterprise application building. In further work we aim to generalise this process by providing a language for building generative mappings.

References

1. J. Domingue, L. Cabral, F. Hakimpour, D. Sell and E. Motta (2004). IRS-III: A Platform and Infrastructure for Creating WSMO-based Semantic Web Services. (WIW 2004)
2. Web Service Modeling Ontology Standard, http://www.wsmo.org/2004/d2/
3. E. Motta, An Overview of the OCML Modelling Language (KEML '98)
4. M. Klein, D. Fensel, F. van Harmelen, and I. Horrocks, 'The Relation between Ontologies and XML Schemas' (2001)
5. XML Schema - W3C Recommendations 28 October 2004, http://www.w3.org/TR/xmlschema-0/ and http://www.w3.org/TR/xmlschema-2/

Using OWL-S to annotate services with ancillary behaviour

Roxana Belecheanu, Mariusz Jacyno, Terry Payne

School of Electronics and Computer Science
University of Southampton
Highfield, Southampton, SO17 1BJ
{rab2, mj04r, trp}@ecs.soton.ac.uk

Abstract: This paper introduces the concept of services with ancillary behaviour and illustrates the use of OWL-S to semantically describe them. The OWL-S syntax used reflects the dynamic and core-function independent nature of ancillary behaviour. The approach is illustrated on the case of a ubiquitous computing system designed to offer care in the home of a cardiac patient. Here one of the challenges is to ensure service availability, team awareness and transaction atomicity. The concept of commitment is discussed as an example of ancillary behaviour that can achieve these requirements.

1 Introduction

Service oriented applications often require that services which implement certain core functions are accompanied by supporting functionality, like monitoring behaviour, commitment, authentication, encryption/decryption. This supporting functionality is usually relevant only to the context of the core function, but does not always play a direct role in the invocation of the core function. This paper introduces the concept of *ancillary behaviour* to describe functionality that is additional to the core service typically annotated in an OWL-S description, and which has the role of augmenting the service's capability, or enhancing the QoS achieved by the core service.

The ancillary functionality of a web service can be either mandatory or optional. Mandatory ancillary operations (e.g. authentication) *must* be executed in addition to the service's core functionality and the service cannot be invoked without the execution of these operations. Ancillary functionality contributes to the delivery of a service either by providing a *supporting role* (enable the core service) or an *enhancing role* (increase the value of the core service) [1].

A related but distinct concept that can also be introduced here is that of a *higher-order service*. A higher-order service is one which determines a particular aspect of the invocation of a core service without invoking the core service. For example, in order to find out the cost of the invocation of a service for certain input parameters, a costing service can be attached to the core service, whereby the invocation of the cost function would not require the invocation of the core service.

2 An application example

An application example for the concept of services with ancillary behaviour is an ubiquitous computing [4] system whereby a set of heterogeneous devices and their applications establish connections between each other in a dynamic manner (i.e. devices leaving and joining the system unexpectedly), in order to achieve certain tasks. Due to the high variability of the Quality of Service of wireless data communications (e.g. line rate, throughput, error rate) in such a system, there is also a need for dynamically discovering and composing the applications running on these connected devices. Abstracting devices (physical) functionality as services (logical) functionality and using OWL-S to semantically annotate these services can help achieve this flexibility. Thus building interoperable service descriptions similar to the Semantic Web Services is needed to accomplish tasks like service discovery, management, invocation and monitoring.

For example, in the case of a ubiquitous system designed to offer care in the home of a patient, there may be the issue of having a limited number of devices, each providing services that take a finite time to execute, and that may have to be shared by several contexts. Hence, no guarantees exist that any single service is available for invocation at any given time. Therefore, a service centric application must ensure:

1. *service availability*: services which are critical for the patient care (e.g. heart rate monitoring) must be available for execution at the desired time.
2. *team awareness*: services must be aware of the status of other services they depend on.
3. *atomic transactions*: due to high power consumption of message transmission, it is necessary to reduce the number of service requests, or to support a mechanism whereby both core and supporting activities can be requested via a single call (i.e. as one transaction).

3 Commitment based services

Commitment is an example of ancillary behaviour that can help achieve this type of service interactions. The concept is used to enforce that the provider of a commitment supporting service commits to perform an action for a requester. This has been formalised within the theory of local and social agent behaviour [2], using concepts from the Beliefs-Desires-Intentions model. Commitment has also been used to model coordination protocols for business transactions [3], e.g. in an “atomic transaction”, several services committed to one requester will succeed or fail as an atomic unit. A common point of the two interpretations of commitment is that a protocol (e.g. operations like *RequestCommitment*, *DischargeCommitment*, *ReleaseCommitment*, etc.) can be designed to allow services to check each other’s availability to perform a joint task, thus building *team awareness*. The difference between the two commitment models is that a committed service in agent theory will eventually perform the task, but may accept other requests in the meantime, while a committed service in a transactions model must be available to the requester immediately after commitment is granted (a reservation-like interaction).

4 Describing ancillary service behaviour in OWL-S

Service ancillary behaviour has two salient characteristics:

1. It is *dynamic*, i.e. it involves communication with other services of which instances can only be discovered at runtime; (hence it cannot be included in the static description of the service's process).
2. It is *core function independent,* i.e. it is common to a range of services with different core functions.

For these reasons, ancillary behaviour has to be abstracted and described separately from a given service's core functionality, thus facilitating several different services sharing the same ancillary behaviour. Ancillary behaviour description should be loosely linked to the core service specification. The resulting workflow for both the core and ancillary functionality can then be realized (through entailment) when an agent reasons about the service, with respect to a usage context.

To illustrate how core services can be augmented by ancillary services, we take the example of commitment as ancillary behaviour and an example of a core service from the healthcare setting. Suppose *HRMonitor_with_Commitment* is a heart rate monitoring service which supports commitment. To annotate it in OWL-S, we use the core service instance *GetHeartRate* and augment it by: i) defining a *CommitmentService* instance and ii) by linking the core service description with the ancillary service description. *CommitmentService* is the representation of a commitment protocol that ensures that the interaction with the *HRMonitor_with_Commitment* service occurs as one transaction. The resulting workflow for this service must therefore be the sequence of the processes: *RequestCommitment, GetHeartRate* and *DischargeCommitment*. Suppose that *GetHeartRate* is an atomic process:

```
<process:AtomicProcess rdf:ID="GetHeartRate">
 <hasOutput rdf:resource="ansConcepts#HeartRate"/>
</process:AtomicProcess>
```

To combine the core and ancillary workflows, we use the OWL-S *Simple Process* class as an unbound service abstraction that can be dynamically linked to the *GetHeartRate* service instance. The *CommitmentService* workflow can be written as:

```
<process:CompositeProcess rdf:ID="Commitment_Process">
 <process:composedOf>
  <process:Sequence>
   <process:Components>
    <process:AtomicProcess rdf:resource="#RequestCommitment"/>
    <process:SimpleProcess rdf:resource="#Core_Function"/>
    <process:AtomicProcess rdf:resource="#DischargeCommitment"/>
   </process:Components>
  </process:Sequence>
 </process:composedOf>
</process:CompositeProcess>,
```

whereby the Simple Process represents a step that must be replaced by a core process before execution, thus allowing the annotation of the commitment as a function that is common and can be attached to several core services. The *CommitmentService* thus acts as a wrapper around the core function of any service that supports commitment.

Additional annotations are then added to the core *GetHeartRate* service to link it with the ancillary commitment service definition: 1. the *GetHeartRate* process ***real-***

izes the abstract Simple Process, thus stating that the ancillary service definition is effectively unbound, until reasoned about in context with the *GetHeartRate* service:

```
<process:AtomicProcess rdf:ID="GetHeartRate">
 <process:realizes rdf:resource="ansAncillary.owl#Core_Function"/>
</process:AtomicProcess>
```

2. In order to distinguish between a committed and non-committed service during discovery, (as typically a service is requested according to the functional parameters of the core service), *ANSAncillaryFunctionality* is defined as a *serviceParameter* type of property, in the profile of any ANS service:

```
<ANSServiceProfile rdf:ID="HRMonitor_with_Commitment_Profile">
 <profile:has_process rdf:resource="#GetHeartRate"/>
 <ansProfile:ANSAncillaryFunctionality>
  <ansProfile:ANSAncillary>
   <profile:sParameter rdf:resource="ansAncillary#Commitment_Profile"/>
  </ansProfile:ANSAncillary>
 </ansProfile:ANSAncillaryFunctionality>
</ANSServiceProfile>
```

5 Further work

Further work aims to address issues which result from using this type of annotation: first, discovering services with ancillary behaviour requires searching by core functionality parameters, as well as by non-functional properties (i.e. *sParameter*), to determine what ancillary services are also provided as part of the service description. Secondly, the runtime composition of the workflows describing the core and ancillary sub-processes into one executable workflow is necessary. The possibility to annotate services with more than one type of ancillary behaviour must also be addressed.

Acknowledgements

This research is funded by DTI (UK) as part of the ANS (Autonomous Networked System) project; other research partners are Imperial College and Lancaster Univ.

References

1. Baida, Z., Akkermans, H. and Bernaras, A., 2003., The configurable nature of real-world services: analysis and demonstration, ICEC-Workshop.
2. P. R. Cohen and H. J. Levesque, Intention is choice with commitment, Artificial Intelligence, 42 (1990), pp. 213-261
3. Papazoglou, M.P., 2003. “Web Services and Business Transactions”, WWW: Internet and Web Information Systems, 6, pp. 49-91.
4. Weiser, M, "The Computer for the Twenty-First Century," Scientific American, pp. 94-10, September 1991

Integration of OWL-S into IRS-III

Farshad Hakimpour, John Domingue, Enrico Motta,
Liliana Cabral and Yuangui Lei

Knowledge Media Institute, The Open University, Milton Keynes, UK
{f.hakimpour, y.lei, j.b.domingue, e.motta, l.s.cabral}@open.ac.uk

Abstract. IRS-III is the first WSMO compliant system for supporting the Semantic Web Services technologies and it is based on the IRS-II [3]. This paper presents how we integrated the OWL-S [5] service description ontology to IRS-III. We describe how the underlying model of IRS-III supports OWL-S.

1 Introduction

As Web Service technologies are evolving, the need for semantic description of the services has also been increasing. There is a great deal of work on developing specifications for describing Web Services, such as in OWL-S [5] and WSMO [1]. The description of Web Services facilitates automatic matching of services with a service request, automatic service composition, controlling and monitoring the execution of a Web Service. IRS-III [3] is one of the few existing systems to support the Semantic Web Services technologies. IRS-III complies with the WSMO ontology of Goal, Web Services and Mediators [1]. WSMO is a developing specification mainly supported and developed by European partners.[1] IRS-III uses a version of WSMO ontology defined in OCML (a knowledge representation language [4]) which also provides the corresponding reasoning system. OWL-S provides a specification for describing semantics of Web Services developed as part of DAML program. Supporting OWL-S will extend the potential of the IRS-III and would also help us to explore the similarities and the differences between OWL-S and WSMO ontology.

In this paper, we explain how ontologies describing a service in OWL-S specification are mapped to the WSMO ontology and translated to OCML which is in turn suitable to be used by IRS-III. The rest of the paper is organized as follows. Section 2 introduces the IRS-III, WSMO ontology and the OWL-S specification. In Section 3, we describe how major features of the OWL-S Process are translated to Web Service descriptions. Section 4 is dedicated to the technical details of OWL-S to WSMO translation, as well as OWL to OCML translation. Finally, in Section 5 we present a summary.

2 Background

IRS (Internet Reasoning Service) is a framework to support Semantic Web Services, in which services can be described by their semantics, discovered, invoked and monitored. IRS-III consists of three main components, IRS Server, IRS Publisher and IRS Client. IRS Server stores and reasons with Web Service and Goal descriptions. IRS Publisher generates wrappers for programs as Web Services. Invocation of Web

1.http://www.wsmo.org/

Services via Goal descriptions is supported by the IRS Client.

The notions of Goal and Mediator are particular characteristic of IRS-III and WSMO ontology. While a Web Service is a description of a method and concerned with the specification of mechanisms and execution, a Goal is a general description of a problem and concerned with describing a problem rather than mechanisms. As a result, Goals are suitable for describing a service for a user whose concern is not the technical details of the solution.

OWL-S [5] is based on a process ontology and benefits from developments in workflow technologies. OWL-S model is based on three major components ServiceModel, ServiceProfile and ServiceGrounding. The most important of the three for IRS-III is the ServiceModel which describes how a service works by describing its Inputs, Outputs, Preconditions and Effects (IOPEs). It also specifies the component Processes of composite services and their execution order. ServiceProfile foresees information that may be required to search for a service, such as, ContactInformation, QualityRating, ServiceCategory and other optional ServiceParameters. Furthermore, a Profile contains pointers to the IOPEs of a Process. ServiceGrounding specifies the details of accessing a service, such as communication protocol and message format. ServiceGrounding is not discussed in this paper, as we concentrate on the semantic description of services here.

3 OWL-S Process and Web Service

The core functional description of services in OWL-S appears in Process descriptions. A Process is described mainly in terms of its functional parameters: Inputs, Outputs, Preconditions and Effects (IOPEs). OWL-S also divides Processes into two types AtomicProcess and CompositeProcess in the ProcessModel. CompositeProcesses are further described by ControlStructures. We map Atomic Process and Composite Process to a Web Services description in IRS-III. In case of a composite web service the composition will be translated to the orchestration part of the Web Service description.

Functional Parameters. Web Service description in IRS-III includes: has-input-role and has-output-role that are also present in the OWL-S process description, as has-Input and hasOutput. The Capability of a Web Service in IRS-III includes: has-precondition, has-assumption, has-effect and has-postcondition. These roles are similar to hasPrecondition and hasEffect in defined in OWL-S 1.0. Since OWL-S 1.1 (Beta version) Effects are part of a new property called hasResult.

The distinction between preconditions and assumptions (as well as postconditions and effects) are result of the distinction between the state in the information space and state of the world. That is, preconditions and postcondions are conditions related to the information space while assumptions and effects are related to the state of the world. At the moment, we map the OWL-S preconditions to IRS assumptions.

On the other hand, OWL-S 1.1 provides extra parameters, namely, Local and ResVar. These parameters are used respectively in the Preconditions and Results. These parameters can be thought of as environment variables that represents states of the world. By this interpretation, OWL-S 1.1 also distinguishes between the state of the information space and the state of the world. However, this distinction is done at the level of

parameters rather than the conditions.

Goals and Web Services. The IRS Web Service is suitable for representing a service description as described by Process in OWL-S. However in IRS-III, the notion of Goal refers to a general description of a problem and can be solved by different Web Services. A Goal describes a problem to be solved and represents the knowledge required for matching the problem to a set of Web Service descriptions presented by providers. During the translation a user may decide whether a Goal based on the OWL-S description should also be generated by the translator. While generating the Goal translator will also generate the necessary mediator to mediate between the Goal and the Web Service. After the translation one can modify the generated Goal and mediator by the IRS browser. Generation of a Goal allows service discovery based on the Goal description. If a user does not require to generate a Goal she may associate a Web Service to an existing Goal, later by the IRS browser. That is, a translated OWL-S description can be associated to any existing Goal.

There is a conceptual difference between the IRS-III and OWL-S. OWL-S views a process (e.g. Amazon-book-selling-service) as an instance of class Process, while in the IRS-III the translated result is defined as a subclass of Web Service. However, this difference has side effect on translation or other IRS-III functionality.[2]

Composition. Describing Service compositions is an essential advantage of the reusability of Web Services. The IRS-III composition model is currently under development and the translator will soon be able to support translation of the composite processes. However, we developed a translator from OWL-S 1.0 to the previous version of IRS (i.e. IRS-II). OWL-S describes a composite process by means of Control Structures. OWL-S Control Constructs are built as a set of Processes, such as a Sequence of processes or an iteration of a list of processes. The same is supported in IRS-II by means of body of a PSM definition in OCML. As a result all control construct in OWL-S can be

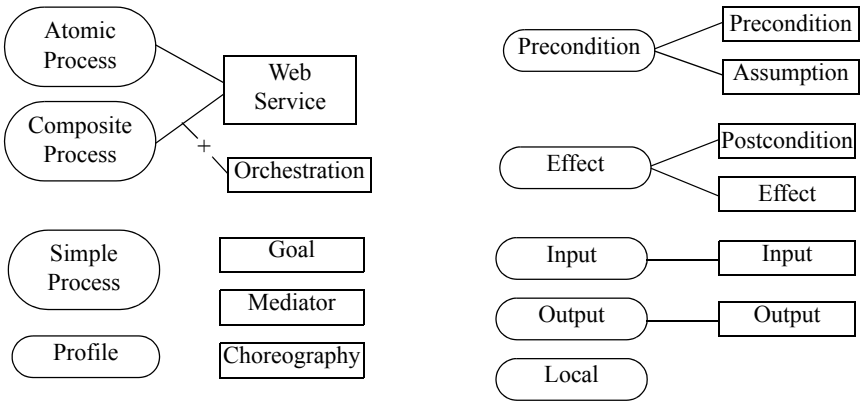


Fig. 1. Similarity of the basic elements of OWL-S (in ovals) and WSMO (in boxes).

2.This difference has started as of OWL-S version 1.0. In OWL-S 0.9 and before a Process (e.g. BravoAir_Process) was defined as a subclass of Process rather than its instance.

easily translated to OCML and used by IRS-II.

In General, IRS allows building a composition of Goals. This feature provides a certain level of dynamism in composing Web Services. Such dynamism can only be achieved by SimpleProcesses in OWL-S. That means, Web Services executed in a composition can be discovered during the execution and should not be necessarily specified during composing process.

4 Translation

The translator contains two components. Firstly, translation of OWL-S to the WSMO ontology and secondly, translation of OWL to OCML. The common concepts in both OWL-S and WSMO ontology are translated by the former component. However, OWL-S service descriptions are in OWL language and contain many definitions in OWL (e.g., concepts defining the input types of a Process). These definitions are translated by the later component. The translator also validates the consistency of the OWL-S service descriptions. For example, Profile is associated to Process descriptions by means of the has_process property. We also validate the consistency of this association with the hasProcess property of the ServiceModel for the Service. There are a number of such validation that are performed on the service descriptions.

5 Summary

This paper describes how we mapped the OWL-S descriptions to WSMO. We explained similarities and differences between OWL-S and the WSMO ontologies. OWL-S uses the ProcessModel for modelling and describing Web Services, which is similar to Web Service in the WSMO ontology. The functional parameters of OWL-S Processes (IOPE's) have similar concepts in the WSMO ontology. The paper discusses how the separation of the Goal and the Web Services can add to the flexibility in defining a composition of tasks.

Acknowledgement

This work is partly supported by and the AKT (GR/N15764/01) project sponsored by the UK Engineering and Physical Sciences Research Council and the DIP (FP6 - 507483) project funded under the European Union's IST programme.

References

1. Fensel, D. and Bussler, C.: The Web Service Modeling Framework WSMF. In Electronic Commerce: Research and Applications, Vol. 1, No. 2, (2002) 113-137
2. McGuinness D. L. and Harmelen F. V. (eds.): OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features/ (2004)
3. Motta, E., Domingue, J., Cabral, L., Gaspari, M. IRS-II: A Framework and Infra-structure for Semantic Web Services. In the Proceedings of The 2nd Int'l. Semantic Web Conf. (ISWC2003), Fensel, D. et al. eds., Springer-Verlag, LNCS Vol. 2870, (2003) pp. 306-318
4. Motta, E.: Reusable Components for Knowledge Modelling: Case Studies in Parametric Design Problem Solving, IOS Press (1999)
5. OWL Services Coalition, OWL-S: Semantic Markup for Web Services, white paper, www.daml.org/services/owl-s/1.0, (Nov. 2003)

Position Papers

MIAKT

David Dupplaw, Srinandan Dasmahapatra, Bo Hu, Paul Lewis, Nigel Shadbolt

IAM Group, University of Southampton, Southampton, SO17 1BJ, UK
[dpd|sd|bh|phl|nrs]@ecs.soton.ac.uk,
WWW home page: http://www.aktors.org/miakt

Abstract. This paper briefly describes the work we have undertaken in the MIAKT project to provide a generic architecture and user interface for distributed multimedia knowledge management with the application of supporting diagnosis of breast-cancer. Most of the domain-specific functionality is provided by web-services and how these are made functionally and practically accessible in a general way is a main concern to the ongoing work in MIAKT, and this is the main focus of this position paper.

The Medical Imaging and Advanced Knowledge Technologies (MIAKT) project is a collaboration of a subset of the partners from the Advanced Knowledge Technologies (AKT) and Medical Imaging and Signals (MIAS) interdisciplinary research collaborations (IRCs). The project is concerned with the management of the knowledge that is produced during breast cancer screening in an attempt to support the collaborative meetings that occur during breast cancer diagnosis. Medical staff from different disciplines come together at a Multi-Disciplinary Meeting (MDM) to discuss cases where symptoms of cancer have already been identified (symptomatic cases). These symptoms are detected using imaging, such as x-rays, magnetic resonance imaging, ultra-sound or microscopic views of the results of a biopsy. Together with historical patient records, interpretation of these provides a diagnosis and therefore identifies the further treatment for the patient.

The MIAKT project currently provides information management in a semantically principled way, accessing knowledge bases through a generic architecture loosely controlled by a generic client application. Functionality of the system is disjoint from the client to provide flexibility and it provided through web-service interfaces which are made available to the client through an enterprise server.

Currently, we have over 20 services available, all providing useful functionality to the MIAKT application of medical imaging. Very briefly, these include retrieval services accessed through the Internet Reasoning Service (IRS) provided by the Open University, and, through a SOAP interface, natural language generation and medical term lookup provided by The University of Sheffield, image analysis provided by The Universities of Southampton and Oxford and King's College London, and Image Registration also provided by King's College London. Further details of these services will be made available at the workshop.

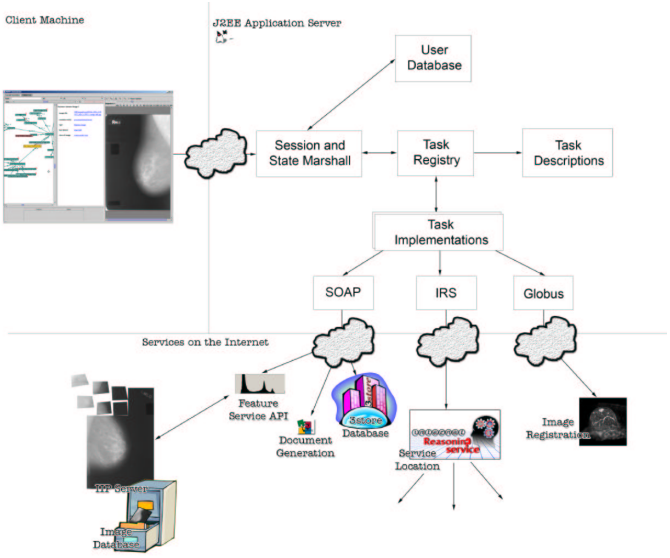


Fig. 1. The MIAKT framework

Trust and security issues are a major concern to medical practitioners who may wish to employ systems such as these, so facing the challenge of providing secure transactions with provenance, with some quality of service on trusted services is a very necessary issue to tackle. Currently, access to the client user interface is governed by username and password databases which means services are available through the enterprise server to the client only if a user has logged in. However, bypassing of both the client and the enterprise server is still possible because access to most of these services is currently unrestricted. The only exception to this are the services provided by King's College London whose firewall is limited to accept connections only from our enterprise server. However, this rather limits flexibility and the general use of the services and was also difficult to negotiate and setup. Both technical and logistical problems made it near impossible to use GRID services directly. Possible ways to tackle this might be with WS-Security [1], as used by the Artemis Project [3], or the extension to that WS-Trust [2]

Once the framework is in place for secure, trusted services to be provided on an ad-hoc basis to different applications, it becomes necessary for these to be described and published in a way that makes it possible for these applications to use them sensibly. Clearly the IRS has some of this functionality already, although having a single point of access to services (that is not the administrating client or server) can be considered a disadvantage. The service's publishing method should ensure that some higher level semantics are provided that in-

dicate a service's role with higher-level tasks (rather than input/output level semantics).

The services in place in MIAKT are all stateless; that is, they all provide a single output from a set of inputs as a black-box. This is important, as the design of the architecture does not currently store state for services. Support for asynchronous web-services currently does not exist, and so there is scope for investigating how these could be implemented and subsequently integrated into the architecture. Currently, expensive GRID services are initiated and return immediately. They are then pinged by the client to retrieve their status once they have started execution. WSGrid [4] has shown that with the addition of client-side services to the architecture, asynchronous services can notify the client on completion, and within a trusted architecture this could be accomplished securely, while retaining programmatic control at the client.

Composition of services may provide a very useful tool for simplifying access to complex services. Assuming the existence of a well described set of atomic services, composition could take place automatically. For example, in the medical domain an automatic suggested diagnosis could be generated from the composition of image segmentation, image analysis and lesion classification algorithms, and the composition could automatically take account of the best-of-breed implementations that are available in each case. It may be necessary for more atomic, less domain-dependent services to be published for this to be actually realised.

Although MIAKT is clearly a flexible architecture, the way in which areas of the system are constructed needs to be addressed in order to realise the architecture as a test-bed for service development; in particular the user interface which currently requires extending to afford the integration of new services (with the exception of image analysis or simple retrieval services). With the further generalisation of this architecture it is possible it could be useful as a general test-bed for the evaluation of service-based solutions.

References

1. WS-Security, IBM
   `http://www-106.ibm.com/developerworks/webservices/library/ws-secure/`.
2. WS-Trust, IBM
   `http://www-106.ibm.com/developerworks/library/specification/ws-trust/`.
3. ARTEMIS Homepage, IT Innovation
   `http://www.it-innovation.co.uk/research/grid/artemis.shtml`.
4. WSGrid, Peter Henderson, 2004
   `http://www.omii.ac.uk/news/wsgrid.ppt`.

Web Service Support for Scientific Data Analysis

Martin J Kollingbaum*, Kun Cai**, Timothy J Norman, Derek Sleeman, and Wamberto Vasconcelos

Department of Computing Science, University of Aberdeen, AB24 3UE
{mkolling,kcai,tnorman,sleeman,wvasconc}@csd.abdn.ac.uk

1 Automated Scientific Analysis

The analysis of large data sets requires extensive computing resources and detailed knowledge about the analysis process itself. E-Science and Grid computing is an effort to provide computing resources for the scientific community and support collaborative research efforts. With the availability of integrated platforms that allow seamless communication and collaboration between researchers and access to distributed computing facilities, there is a strong case to add intelligent applications on top of these infrastructures that support and inform research activities and allow the automation of scientific workflows [1].

Providing an automated advice for scientists in their data analysis is not a new concern. Previous work has been conducted in the machine learning domain. Kodatoff et al. [3] and Sleeman et al. [4] describe an advisory system for this domain. The advisor uses a knowledge base that captures selection criteria for various machine learning algorithms. The central element of this system is a classification expert system, called Consultant, which interrogates the user about features of the learning task and recommends algorithms based on this information. Consultant shows the essential elements of an automated advisor in the machine-learning domain: information about the learning task (including characteristics of the data) is fed into a central knowledge base, which returns specific advice.

The Consultant expert system is a typical "single-vendor" monolithic tool. What we need today is an environment that accounts for the collaborative nature of research. The Web can be used to establish an advisory system that can be maintained by the research community. Consequently, the functionality of such a system is not established by a single group (organisation, company) but emerges from a collaborative effort of many people. This can be described as a "multi-vendor" scenario.

2 Web Services

Web service technology has been used to design a web-based scientific advisor. The move towards web services is driven by the requirement that it is main-

* Funded by the EPSRC IRC in Advanced Knowledge Technologies GR/NI 5764
** Partially funded by the EPSRC Master Training Program in E-Commerce Technology GR/N29709

tainable by the research community. In its essence, it shows the same elements as the original implementation: domain knowledge is captured as a set of rules comprising the knowledge base, which is used to advise on the selection of a specific algorithm. But it is not an advisor for any specific domain — the system represents a platform to which domain specifications can be submitted by users via the Web. Given such heuristics, a user in the role of a knowledge engineer can submit such a specification to the advisory system. The knowledge engineer engages in a dialogue with an administrator tool to specify algorithm selection heuristics in a convenient manner. This tool ultimately generates an OWL description of the engineer's specification, but in generating this description the tool may give guidance to the engineer by asking critical questions such as: "is decision criteria x derivable from the data set or must it be user-specified?". Once this design is complete, the administrator tool automatically generates a set of Jess[1] rules that provide a procedural encoding of the domain heuristics. This domain knowledge is subsequently available for any user seeking advice. Figure 1 shows the essential elements of this system with the two basic roles for users. The so-called "administrator" is responsible for the deployment of new domain-specific knowledge bases. These are wrapped by so-called "broker" services, which provide the advisory service.

Future research has to take into account the open nature of such a web-based advisory infrastructure. Because of its openness, its functionality emerges from the concerted effort of the research community. It also offers possibilities for commercial vendors to contribute descriptions and knowledge about products implementing scientific analysis or to access various sources that can provide data. With scientists using third-party data for their analysis or with commercial interests involved, problems of privacy, charging models, trust in and reputation of services have to be tackled [2]. Using agent technology in this advisory scenario can provide the necessary capability to negotiate contracts for the usage of specific services or to perform analysis tasks without disclosing privacy issues of data sets.

References

1. D. De Roure, N.R. Jennings, and N.R. Shadbolt. The Semantic Grid: Past, Present and Future. In *to appear*, 2005.
2. I. Foster, N.R. Jennings, and C. Kesselman. Brain Meets Brawn: Why Grid and Agents Need Each Other. *Proc. 3rd international Conference on Autonomous Agents and Multi-Agent Systems AAMAS 2004*, pages 8–15, 2004.
3. Y. Kodratoff, D. Sleeman, M. Uszynski, K. Causse, and S. Craw. Building a Machine Learning Toolbox. In L. Steels and B. Lepape, editors, *Enhancing the Knowledge Engineering Process*, pages 81–108. Elsevier Science Publishers, 1992.
4. D. Sleeman, M. Rissakis, S. Craw, N. Graner, and S. Sharma. Consultant-2: pre- and post-processing of Machine Learning Applications. *International Journal of Human-Computer Studies*, 43:43–63, 1995.

[1] http://herzberg.ca.sandia.gov/jess

Proposed Functional-Style Extensions for Semantic Web Service Composition

Barry Norton

Department of Computer Science, University of Sheffield
Regent Court, 211 Portobello Street, S1 4DP Sheffield, UK
B.Norton@dcs.shef.ac.uk

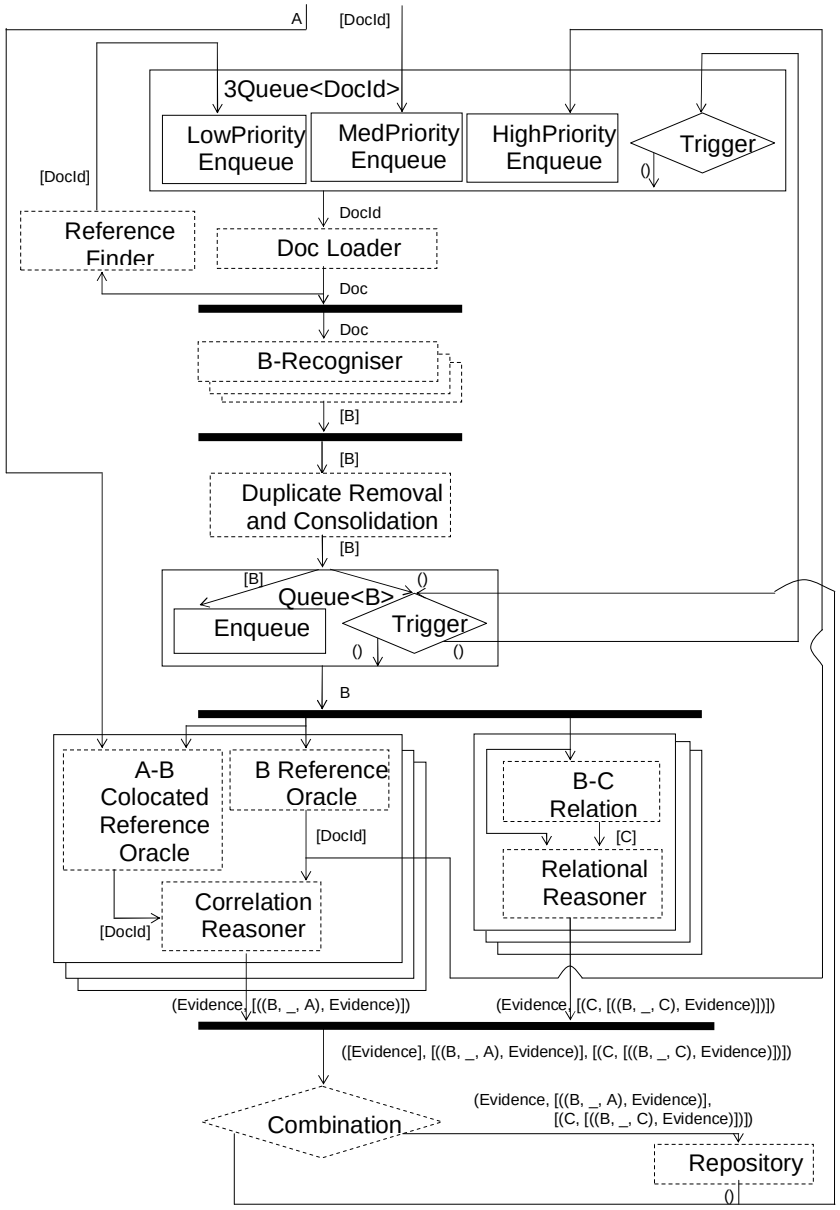
In a related paper [9] we set out how various parts of a semantic web service-based architecture for Armadillo[2], a harvesting tool for semantic annotation, can be instantiated with information extraction and related language services. We have constructed this as a workflow, illustrated as a whole on the following page, in BPEL4WS [3], reasoning, as have several other authors [7], that even while we should like to take advantage of semantic web service technology there exist few, if any, generally available choreography solutions for OWL-S. As a result we plan to take the lessons learned as input to an effort to implement and extend a 'coordination engine' for OWL-S in the CASheW-s project[1].

The broad goals are to implement an engine in the programming language Haskell [6] to which a workflow, expressed in an extended XML-encoded version of OWL-S, can be communicated (over SOAP). This workflow will then be converted to a process-algebraic representation in CaSE [10], a qualitatively-timed CCS derivative, from which coordination engines have already been formed for Microsoft COM via H/Direct [4]. Our plan is to use the GXS module of the HAIFA framework [5] to allow the same kind of binding of SOAP services.

Our existing generalised dataflow model in CaSE [10] allows a semantics for dataflows with loops, two loops can be seen over, and non-deterministic agents, shown over as flow-graph style diamonds. Both of these features are disallowed from the 'Flow' construct in BPEL (to construct the flow shown over it is necessary to use more than one workflow, BPEL being insufficiently algebraic), and the former syntactically restricted out of OWL-S, but very useful.

Our other major observation from BPEL was the frustrating need for use of its extended XPATH language together with mutable variables - anathema to the kind of analyses we should like to allow over designs. We have previously proposed parametrically polymorphic functional language-like features as a more suitable means to carry out transformation of XML-encoded data [8].

As well as allowing the binding of pure-functional Haskell operations at the 'mediation' level between services (as shown for instance in the second queue with $[B]$, a list of elements typed B, being transformed into $()$, the singleton type), we should like to go further in adding operations to workflow primitives. Firstly, we should like to bind Haskell functions to 'If-Then-Else' constructs, allowing us to form an operation like 'Trigger' (in the two Queue services) directly from a WSDL-described service without non-deterministic outputs. Furthermore, we should like to bind functions to the 'Split' and 'Join' primitives so that information can be propagated forward and consolidated through these features without the use of global variables, as shown in the diagram.



Acknowledgements

This work was carried out within the AKT project (http://www.aktors.org), sponsored by the UK Engineering and Physical Sciences Research Council (grant GR/N15764/01), and the Dot.Kom project, sponsored by the EU IST asp part of Framework V (grant IST-2001-34038). Also to be acknowledged are the students working on the CASheW-s project: Ravish Bhagdev, Xian Liu, Atheesh Sanka, Andrew Hughes and Simon Foster whose HAIFA project is being continued here.

References

1. CASheW-s engine project. http://savannah.nongnu.org/projects/CASheW-s-engine.
2. Fabio Ciravegna, Sam Chapman, Alexiei Dingli, and Yorick Wilks. Learning to harvest information for the semantic web. In *Proceedings of the First European Semantic Web Symposium*, May 2004.
3. IBM *et al.* Business process execution language for web services version 1.1. http://www-128.ibm.com/developerworks/library/ws-bpel, 2003.
4. Sigbjorn Finne, Daan Leijen, Erik Meijer, and Simon Peyton Jones. H/Direct: A binary foreign language interface for Haskell. In *Proc. 3rd ACM SIGPLAN International Conference on Functional Programming (ICFP-98)*, ACM SIGPLAN Notices. ACM Press, 1998.
5. Simon Foster. HAIFA : An interoperability framework for Haskell. MEng Dissertation, Department of Computer Science, University of Sheffield, 2004. http://www.dcs.shef.ac.uk/teaching/eproj/ug2004/abs/u1sf.htm.
6. Simon Peyton Jones. *Haskell 98 Language and Libraries*. Cambridge University Press, 2003.
7. David J. Mandell and Sheila A. McIlraith. Adapting BPEL4WS for the Semantic Web: The bottom-up approach to web service interoperation. In *Proc. 2nd Intl. Semantic Web Conference (ISWC2003)*, 2003.
8. Barry Norton. Eclipse as a development platform for semantic web services. Eclipse Technology Exchange (eTX04), 18th European Conference on Object-Oriented Programming (ECOOP-2004), 2004.
   http://www.dcs.shef.ac.uk/∼barry/CASheW-s/Norton04.pdf.
9. Barry Norton, Sam Chapman, and Fabio Ciravegna. Developing a service-oriented architecture to harvest information for the semantic web. In *Proc. 1st AKT Workshop on Semantic Web Services*.
10. Barry Norton, Gerald Luettgen, and Michael Mendler. A compositional semantic theory for synchronous component-based design. In *Proc. 14th Intl. Conference on Concurrency Theory (CONCUR'03)*, volume 2761 of *LNCS*, 2003.

Position Paper for the First AKT Workshop on Semantic Web Services

Birgitta König-Ries[1] and Michael Klein[2]

[1] Institut of Computer Science, Friedrich-Schiller-Universität Jena, D-07743 Jena, Germany, koenig@informatik.uni-jena.de
[2] Institute for Program Structures and Data Organization, Universität Karlsruhe, D-76128 Karlsruhe, Germany, kleinm@ipd.uni-karlsruhe.de

1 The DIANE Service Description

The goal of the DIANE project[1] is it to enable automatic resource sharing in dynamic environments, more precisely in ad hoc networks. The work is based on the paradigm of service oriented computing. One major prerequisite to allow for *automatic* resource sharing in such a system is the ability to automatically discover and bind services. This in turn requires semantic service descriptions and appropriate methods to match service offers and requests.

Since existing service description languages, in particular OWL-S, did not provide all that is necessary for automation, we have developed our own description language, the DIANE Service Description (DSD). In our opinion, this language prototypically realizes all the functionality that is needed from a semantic service description.

A major enhancement of DSD over OWL-S is the explicit distinction between service offers and requests. While in OWL-S (and most other proposals), service requests are formulated as descriptions of the ideal service, DSD allows for a more flexible, yet at the same time more precise description of what is needed. Instead of specifying one instance (namely the ideal one) and leaving it to the matcher to determine how close any given offer is to that request, in DSD, a requestor will specify a fuzzy set of acceptable services. This specification explicitly encodes the user's preferences so that the matcher is able to unambiguously decide how well a service offer matches the request.

A second major difference of DSD from other approaches is its pure state-orientation. Instead of modeling message flow and state change separately as OWL-S and WSMO do, DSD describes services exclusively by their state change. The message flow is encoded in these effects by the introduction of input and output variables. These variables are bound during the matching process. This approach has two advantages: First, the semantics of the service are captured more clearly, since the influence of input variables on the effect is made explicit. Second, it allows to invoke services that offer the desired functionality but use a different interface then the one envisioned by the requestor.

[1] http://www.ipd.uka.de/DIANE/en

2 Tools from the DIANE Project

Within the DIANE project, we are developing not only the language itself, but also a number of accompanying tools.

- A Microsoft VISIO template is available that allows even unexperienced users to graphically develop DSD descriptions. These are then automatically translated into a formal representation.
- Transformation tools that transform the formal representation for example in a java based one. These java classes are then used by the other components of the system.
- A matcher that takes full advantage of the features of DSD is currently being developed. A preliminary version with limited capabilities is available, the full matcher will be realized by the beginning of 2005.
- A simulation environment, DIANEmu, that allows for extensive testing of service discovery and invocation in a dynamic environment, is available, too.
- Finally, we offer an execution framework, i.e. a middleware platform.

3 Questions to be Addressed at the Workshop and Challenges for Semantic Web Services

- What are the key features each description language for semantic web services should possess? Which of these features are still lacking from current approaches?
- Do we really need powerful reasoning mechanisms for semantic web services? Can't matching be done without them?
- A unified world ontology is certainly not realistic. How can we handle a multitude of small (and possibly overlapping) ontologies?
- Will there be "the one" description language? If not: How can co-existing solutions be used in a unified way (or: Given a request expressed in OWL-S, will I be able to find a service described in WSMO?)

About the Authors

Birgitta König-Ries got her PhD from the University of Karlsruhe in 1999. After 3 years as a postdoctoral research assistant at the University of Louisiana at Lafayette and Florida International University, she returned to Germany. After a few years as head of the mobile information systems group at Karlsruhe University, she has recently joined the Computer Science Department of the Friedrich-Schiller-University in Jena. Her research is focused on resource sharing in dynamic, in particular mobile environments.

Michael Klein studied computer science at the University of Karlsruhe in Germany. Since 2001, he is working towards his PhD at the Institute for Program Structures and Data Organization at the University of Karlsruhe. His main research area is semantic service descriptions.

Demo Papers

OntoSearch: a Semantic Web Service to Support the Reuse of Ontologies[1]

Edward Thomas, Yi Zhang, Joe Wright, Craig McKenzie, Alun Preece, Derek Sleeman
Department of Computing Science, University of Aberdeen
Aberdeen, AB24 3UE
{ethomas, yzhang, jwright, cmckenzi, apreece, dsleeman} @ csd.abdn.ac.uk

This document provides an overview of the development of OntoSearch, an ontological search engine designed to help users find RDF based ontological information on the Semantic Web. It uses the Google API to search several million documents on the Semantic Web with supporting servlets to provide summary information & various visualizations of the documents found.

1. Overview and Motivation

Finding a suitable ontology from the Internet is a hard task because of the difficulty of separating ontological data from the mass of instance data on the Semantic Web and quickly evaluating its suitability. There is still no good tool to handle this problem. Google offers a powerful web search engine. However, with regard to ontology searching, it has its own problems, such as a lack of visualisation facilities. Using the Google API[2] give us a chance to develop our own tool (OntoSearch*[2]*) to search the relevant ontology files to meet the user's requirements.

2. Conceptual design of OntoSearch

The concept of OntoSearch is to facilitate knowledge reuse by allowing the huge number of ontology files available on the Semantic Web to be quickly and effectively searched using an online tool. The search results contain both ontologies and other SWDs to allow knowledge engineers to browse relevant documents to find supplementary information on a subject if a complete ontology is not available.

To allow fast development of a system it was decided to use the Google search engine through its API to search the available Ontologies in the RDF(S)[3], OWL[4] and DAML (+OIL)[5] representational formalisms. Once the Google results have been returned, each document listed is examined and summary information identifying where the terms matched the returned documents and statistical data about the size of

1 This work is part of the Advanced Knowledge Technology (AKT) [1] project, which is funded by EPSRC
2 Google APIs: http://www.google.com/apis
3 RDF(S): http://www.w3.org/TR/rdf-schema/
4 |OWL: http://www.w3.org/TR/owl-features/
5 DAML (+OIL): http://www.w3.org/TR/daml+oil-reference

the ontology is presented to the user to allow quick evaluation on the suitability of a large number of potential ontologies and other Semantic Web Documents (SWDs).

3. Outline of implementation

OntoSearch is implemented as a number of specialised servlets which work together to provide the functionality of OntoSearch. This architecture allows additional features to be plugged into the basic system quickly and easily.

The SearchServlet is responsible for querying the Google database using the Google API object, it takes the users query either through a web form or as an HTTP GET request and returns a list of matching ontology files from Google, either as an HTML file or an RDF file (depending on a variable set in the user's request).

If an HTML file is returned, this contains several embedded iframe elements which each reference the DetailsServlet. This accesses each document and examines the ontology returned and displays a list of where the search terms were found in the ontology, gives general statistics about the size of the ontology and lists the namespaces used in the ontology.

Once a file has been found which appears to suit the purpose, a variety of visualisations are available through links on the results page, either in a format provided by Notation 3 (N3) or as a hyperbolic tree showing the structure of the ontology. The visualisation is particularly useful for examining an entire ontology quickly, browsing through multiple classes and linkages to quickly see the underlying structure.

4. Discussion

The facilities provided by the current version of OntoSearch have been used by several people within AKT, and over 2000 searches have been performed. However there are several areas of the system which we intend to develop further.

To address these points, a new version of OntoSearch is currently being designed which will preserve the unique functionality of OntoSearch and to address the short-comings identified to provide a more valuable tool. It will allow the data to be searched for a far wider set of parameters (including some aspects of structure), and the results returned will be more focused as an enhanced scoring mechanism will reflect the important aspects of the user's request.

The current HTTP API will be extended to allow the new features to be made available to other applications as a web service, and the basic RDF results which can be accessed at OntoSearch will be extended to include the full range of information available through the web interface.

References

1. Advanced Knowledge Technology (AKT project) http://www.aktors.org
2. Zhang Y, Vasconcelos W, and Sleeman D. OntoSearch: An Ontology Search Engine (AI-2004). The Twenty-fourth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence, Cambridge.