RA 220 July 16, 2003
Computer Science

IBM Research Report

Proceedings of the First European Workshop on Object Orientation and Web Services

Editors:

Giacomo Piccinelli
Department of Computer Science
University College London
United Kingdom
G.Piccinelli@cs.ucl.ac.uk

Sanjiva Weerawarana
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
and
University of Moratuwa
Sri Lanka
sanjiva@us.ibm.com

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

First European Workshop on Object Orientation and Web Services

Held at ECOOP '03, Darmstadt, Germany
July 21, 2003

Organizers

Anthony Finkelstein[1], Winfried Lamerdorf[2], Frank Leyman[3], Giacomo Piccinelli[1], and Sanjiva Weerawarana[4]

[1]Department of Computer Science, University College London, United Kingdom
{A.Finkelstein, G.Piccinelli}@cs.ucl.ac.uk
[2]Department of Computer Science, University of Hamburg, Germany
Lamersd@informatik.uni-hamburg.de
[3]IBM Software Group Germany and University of Stuttgart, Germany
LEY1@de.ibm.com
[4]IBM T. J. Watson Research Centre and University of Moratuwa, Sri Lanka
Sanjiva@us.ibm.com

Themes and Objectives

Web Services are evolving beyond their SOAP, WSDL, and UDDI roots toward being able to solve significant real-world integration problems. Developers of Web Services systems are currently working on new generations systems that incorporate security, transactions, orchestration and choreography, grid computing capabilities, business documents and processes, and simplified integration with existing middle-ware systems. Current economic issues continue to force consolidation and reduction in enterprise computing resources, which is resulting in developers discovering that Web Services can provide the foundation engineering and realisation of complex computing systems.

The question of how Web Services could and should change system and solution development is very much open. Are Web Services just about standards, or do they

imply a new conceptual framework for engineering and development? Similarly open is the question of how requirements coming from system and solution development could and should make Web Services evolve. In particular, methodologies as well as technologies based on the object-oriented conceptual framework are an established reality. How do Web Services and object-orientation relate? How can Web Services leverage the experience built into current object-oriented practices?

The overall theme of the workshop is the relation between Web Services and object orientation. Such relation can be explored from different perspectives, ranging from system modelling and engineering to system development, management, maintenance, and evolution. Aspects of particular interest are the modularisation of a system into components and the (possibly cross-domain) composition and orchestration of different modules. Components and composition are closely connected with the issue of reuse, and an important thread of discussion within the workshop will address the way in which Web Services impact reuse.

The objective of the workshop is twofold: assessing the current work on Web Services, and discussing lines of development and possible cooperation. Current work includes research activities as well as practical experiences. The assessment covers an analysis of driving factors and a retrospective on lessons learned. The identification and prioritisation of new lines of research and activity is a key outcome of the workshop. In particular, the intention is to foster future cooperation among the participants.

Table of Contents

Web Services and Seamless Interoperability

João Paulo A. Almeida, Luís Ferreira Pires, Marten J. van Sinderen

Centre for Telematics and Information Technology, University of Twente
PO Box 217, 7500 AE Enschede, The Netherlands
almeida@cs.utwente.nl, pires@cs.utwente.nl,
sinderen@ctit.utwente.nl

Abstract. Web Services technologies are often proposed as a means to integrate applications that are developed in different middleware platforms and implementation environments. Ideally, application developers and integrators should be shielded from the existence of different middleware platforms and programming language abstractions. This characterizes seamless interoperability, in which a set of consistent constructs is manipulated to integrate both the applications or services that are located both in the same and in different technology domains. In this paper, we argue that Web Services are not sufficient to facilitate seamless interoperability. We also outline some developments that may be used in a systematic approach to seamless interoperability within the context of the Model-Driven Architecture.

1 Introduction

The generalized term *Web Services* does not currently describe a coherent or necessarily consistent set of technologies, architectures, or even visions [18]. It is often used loosely to denote a collection of related technologies, which include: SOAP [17], Web Services Description Language (WSDL) [21] and Universal Description, Discovery and Integration (UDDI) [16].

Web Services technologies are built upon widely supported Internet standards, including XML standards, HTTP, SMTP, FTP, etc. and stem from the Internet community. These technologies have gained strong industry momentum and are supported by a large number of organizations, such as IBM, Microsoft and Sun Microsystems.

Web Services technologies are based on concepts that include strict separation between interface and implementation and adequate level of coupling (often loose coupling for application integration). With respect to these concepts, Web Services do not introduce significant novelties or enhancements. These concepts are derived from and largely identical to the ones adopted in more mature middleware or integration technologies, such as CORBA, Java RMI, DCOM and Enterprise Application Integration in general [11].

Nevertheless, with respect to standardization, Web Services only require agreement with respect to the protocols used to realize interactions between application parts. This leads to a significant difference between Web Services and traditional middleware, such as, e.g., CORBA/CCM and EJB, in which interfaces to

access the run-time infrastructure are also standardized. In the case of Web Services, these interfaces are, in general, proprietary or defined within the scope of a particular technology domain, i.e., implementation environment and/or middleware platforms such as, e.g., J2EE [12], .NET [1] or CORBA/CCM [4].

In this paper, we do not intend to criticize Web Services standards or consider specific technical issues related to Web Services implementation support. We rather aim at questioning Web Services in its merits as an architecture to support seamless interoperability of applications developed in different technology domains. Ideally, an application developer should manipulate a set of consistent constructs to integrate both the applications that are located within the same technology domain and applications or services that are implemented in other technology domains. We outline some developments that may be used in a systematic approach to seamless interoperability within the context of the Model-Driven Architecture (MDA) [7].

2 Web Services Abstractions

There is no consensus yet on a precise vocabulary and conceptual model for Web Services [18]. Both a "Web Services Reference Architecture" and a new version WSDL (WSDL 1.2) ([18, 22]) are work-in-progress within the context of the World Wide Web Consortium (W3C). Therefore, we provide some concepts and definitions for the purpose of precision and clarity within the scope of this paper.

A *web service provider* is a software entity that offers web services. A *web service* is a set of *endpoints* that operate on SOAP messages conveyed by Internet protocols, such as HTTP, FTP and SMTP. Each endpoint is identified by a Uniform Resource Identifier (URI). A web service and its endpoints may be described in WSDL. WSDL allows one to define the message types and message exchange patterns manipulated by web service endpoints, as well as the concrete means to interact with the web service endpoints, entailing concrete protocols for message exchange and the URIs that identify the web service endpoints. While WSDL descriptions are recommended for interoperability of web services descriptions, WSDL is not the only means to describe a web service. Descriptions in WSDL may be augmented with descriptions in other languages, such as Web Service Choreography Interface (WSCI) [19] and Business Process Execution Language for Web Services (BPEL4WS) [14].

Figure 1 shows a service requester and a web service provider that interact through the exchange of SOAP messages. A web service provider may also assume the role of service requester with respect to another web service provider.

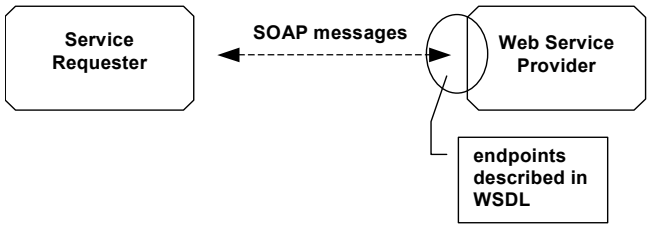

Fig. 1. Service requester and a web service provider interact through SOAP messages

In order to interact with a web service provider, a service requester must be able to find descriptions of the web service that define the concrete means to interact with the web service endpoints. A web service description does not prescribe a particular means to find the web service. A web service description may be found through a local file system, an FTP site, a standardized service registry such as UDDI registries [16], etc.

3 Middleware Platforms and Implementation Environments

Web services are not implemented in a green-field situation. This means developers of web services requesters and providers have to cope with the re-use of legacy applications and infrastructures that have been deployed and that are still being deployed successfully. Examples of these (legacy) implementation infrastructures on top of which web services requesters and providers are implemented are: middleware platforms, such as DCOM, CORBA, Java RMI and JMS; and programming languages such as Java, COBOL, Visual Basic and the .NET languages. Figure 2 shows the resulting structure of the integration of applications implemented in different technology domains with web services technologies.

Legacy implementation infrastructures are specified and implemented with abstractions that differ from the abstractions manipulated for the specification and implementation of web services. Examples of divergences can be seen in the definition of data types (Java datatypes versus XML Schema Data Types [13]), the failure semantics of RPC invocations, the abstractions for object references, etc. Therefore, there must be some support to accommodate the differences in the abstractions manipulated, in order to (i) provide abstractions that are suitable and intuitive for application developers that develop and maintain applications in different technology domains, and in order to (ii) re-use a larger number of specifications and components defined in terms of the abstractions of particular technology domains.

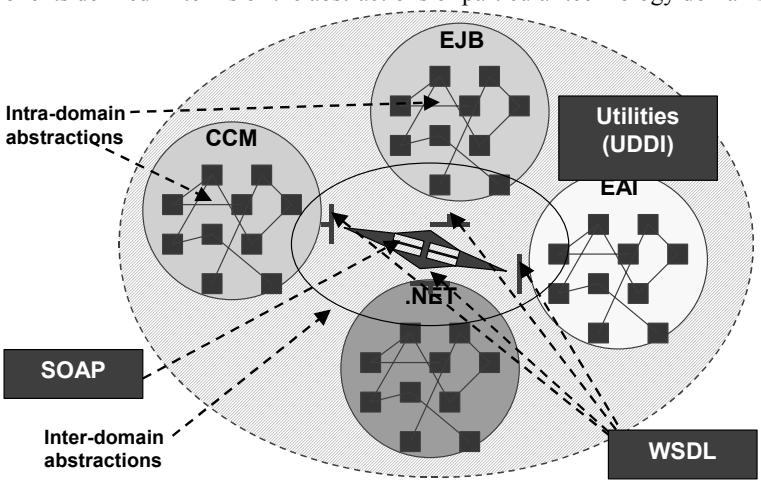

Fig. 1. Web services for inter-domain interoperation

4 Seamless Interoperability

In order to enable the cooperation of distributed applications, Web Services must accommodate the heterogeneity of middleware platforms, programming languages and other technologies in which these applications are realized. Not only interoperability may be hindered by the heterogeneity of platforms, but also application portability and the *provision of transparency* for the application developer. Ideally, application developers should be shielded from the existence of different middleware platforms and programming language abstractions, manipulating a set of consistent higher-level constructs to access both the services that are located within the same technology domain and services that are implemented in other technology domains.

In this sense, Web Services technologies can only offer a solution if they are adopted for all future intra-domain development. This would mean that the abstractions manipulated in Web Services languages and protocols should be used as a starting point for development of applications at the first place. Given the proposed use of Web Services as a technology for the integration of applications and services implemented on top of different middleware platforms, it is unlikely that Web Services will replace existing middleware platforms. This is corroborated with the fact that some of these platforms are flourishing now and have strong Web Services support such as the J2EE and .NET platforms. If Web Services are confined to inter-domain interoperation, abstractions manipulated by intra-domain middleware platforms will indeed diverge from abstractions manipulated across technology domains, and there will always be a "seam" between the abstractions manipulated in a technology domain and abstractions used in inter-domain interoperation. As a consequence, a large effort in the development of web services is concentrated on the (manual) coding of wrappers to existing applications.

The lack of seamless interoperation can be observed in different attempts to provide mappings between Web Services abstractions and abstractions supported by different middleware platforms, such as, e.g., the mappings from and to Java in the JAX-RPC specification [13], the mappings from and to .NET's Common Type System [2] and the upcoming mappings from and to CORBA IDL [5, 6]. These mappings are not sufficient to overcome the intrinsic conceptual differences of the abstractions adopted. For example, a Java developer that is used to passing remote object references as parameters in J2EE is not able to do so if an object is to be exposed as a web service endpoint [13]. This is because the concept of remote object references is not directly supported in a standardized way in SOAP and WSDL, and hence this abstraction has no direct counterpart. Several other examples of mismatch can be identified when considering these mappings, in terms of fault semantics, type mappings, etc. This is a recurring pattern that we have seen earlier in the development of mappings to and from OMG Interface Definition Language (IDL) to Java, C, C++, Ada, Smalltalk, etc. [3].

Abstractions of particular domains are not the only obstacles for seamless interoperation. For applications to achieve meaningful interaction, they must agree on the application protocols they use. These protocols have been called *application choreographies* [11] in the context of web services, and refer to the behavioural or dynamic aspects of an application or application parts that cooperate. Behaviour

complements static aspects of a system, such as interface signatures, data structures and deployment descriptors. Divergences in the behaviour of components of different technology domains offer challenges to transparent inter-domain interoperability. For example, the use of the Naming Service in a CORBA platform to retrieve object references requires clients to be able to locate the Root Naming Context and request the resolution of the names that refer to the objects they are interested in. Even if the mapping from SOAP/IIOP were transparent, web services requesters would be directly exposed to the use of the Naming Service, and would not be able to locate a service if they were not able to use the Naming Service properly. The rule of thumb often considered in this case is to avoid exposing such internal aspects of a technology domain in a web services definition.

This approach, however, is severely limited for non-trivial web services, since it is based on the assumption that the interface of a service can be simplified regardless of intrinsic complexities of service requester - service provider interactions. An example of potentially harmful simplification is the replacing of callback invocations to request/response polling invocations, such as in the Parlay Web Services standardization activities [15], implying in limitations to the scalability of the service.

5 Outlook

We expect that a more systematic approach to accommodate the divergences in abstractions may be defined in a model-driven approach to application development, such as proposed in the context of the Model-Driven Architecture by the Object Management Group (OMG) [7]. In such an approach, mappings between Web Services abstractions and abstractions of other implementation infrastructures would be facilitated through the use of platform-independent models, meta-modelling techniques and model transformation tools.

There is on-going standardization activity in mapping platform-independent models to Web Services artefacts: an OMG Request For Proposal (RFP) has been issued [9] to request for a mapping from the EDOC-Component Collaboration Architecture UML Profile to XML-Schema, WSDL 1.1 and SOAP. An initial submission [10] is available, and a revised submission is expected in August 2003. These efforts, however, should be revisited with the adoption of UML 2.0 [8].

With respect to the application choreographies, the behavioural aspects of a web service may be specified in Web Services specific languages, such as e.g., WSCI [19] and BPEL4WS [14]. These languages are being considered in the W3C Web Services Choreography Working Group [20] as an input for a W3C recommendation for a Web Services specific behaviour modelling language. We will work on the incorporation of these Web Services behavioural descriptions into a systematic model-driven approach, by defining transformations from behavioural descriptions in UML (or specialized UML profiles) to these languages and vice-versa. This would allow seamless interoperability to be considered at platform-independent level through platform-independent models that include the behavioural aspects of a system and its components. These platform-independent models are ultimately reflected at platform-specific level through model transformations.

Acknowledgements

We are currently working on these issues in the context of the MODA-TEL IST project (http://www.modatel.org), supported by the European Commission, and the WASP project (http://www.freeband.nl/projecten/wasp/ENindex.html), supported by the 'Telematica Instituut' in the Dutch Freeband Programme.

References

1. Microsoft Corporation. .NET Development. Available at http://msdn.microsoft.com/library/en-us/dnanchor/html/netdevanchor.asp
2. Microsoft Corporation. Data Types Supported by XML Web Services Created Using ASP.NET. Available at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpcondatatypessupportedbywebservices.asp
3. Object Management Group. Catalog of OMG IDL / Language Mappings Specifications. Available at http://www.omg.org/technology/documents/idl2x_spec_catalog.htm
4. Object Management Group. Common Object Request Broker Architecture: Core Specification, Version 3.0, formal/02-12-06, Dec. 2002.
5. Object Management Group. CORBA-WSDL/SOAP specification, ptc/03-01-14, Jan. 2003.
6. Object Management Group. Joint Revised Submission to the WSDL-SOAP to CORBA Interworking RFP, mars/03-03-03, March 2003.
7. Object Management Group. Model Driven Architecture, ormsc/01-07-01, July 2001.
8. Object Management Group. UML 2.0 Superstructure RFP, ad/00-09-02, Sept. 2000.
9. Object Management Group. Web Services for Enterprise Collaboration (WSEC) RFP, mars/2002-06-06, June 2002.
10. Object Management Group. Web Services for Enterprise Collaboration (WSEC), mars/02-10-11 October 2002.
11. Schmidt, D. and Vinoski, S. Object Interconnections: CORBA and XML – Part 3: SOAP and Web Services, C/C++ Users Journal C++ Experts Forum, Sept 2001.
12. Sun Microsystems. Java 2 Platform Enterprise Edition Specification, v1.4, April 15, 2003.
13. Sun Microsystems. Java API for XML-Based RPC Specification 1.0, June 2002.
14. Thatte, S (ed.). Business Process Execution Language for Web Services, Version 1.0, July 2002. Available at http://www.ibm.com/developerworks/library/ws-bpel/
15. The Parlay Group. Parlay Web Services Architecture Comparison, October 2002. Available at http://www.parlay.org/specs/ParlayWebServices-ArchitectureComparison1_0.pdf
16. Universal Description, Discovery and Integration (UDDI) project. UDDI: Specifications. Available at http://www.uddi.org/specification.html
17. World Wide Web Consortium. SOAP Version 1.2, May 2003. Available at http://www.w3.org/TR/soap12-part1/
18. World Wide Web Consortium. Web Services Architecture Working Draft, Nov. 2002. Available at http://www.w3.org/TR/ws-arch/
19. World Wide Web Consortium. Web Service Choreography Interface 1.0, August 2002. Available at http://www.w3.org/TR/wsci/
20. World Wide Web Consortium. Web Services Choreography Working Group Charter. Available at http://www.w3.org/2003/01/wscwg-charter
21. World Wide Web Consortium. Web Services Description Language (WSDL) 1.1, March 2001. Available at http://www.w3.org/TR/wsdl
22. World Wide Web Consortium. Web Services Description Language (WSDL) 1.2 Working Draft, March 2003. Available at http://www.w3.org/TR/wsdl12/

enTish: an Approach to Service Description and Composition

Stanislaw Ambroszkiewicz[1,2] *

[1] Institute of Informatics, University of Podlasie,
al. Sienkiewicza 51, PL-08-110 Siedlce, Poland
[2] Institute of Computer Science, Polish Academy of Sciences,
al. Ordona 21, PL-01-237 Warsaw,
sambrosz@ipipan.waw.pl; http://www.ipipan.waw.pl/mas/

Abstract. A technology for service description and composition in open and distributed environment is proposed. The technology consists of description language (called Entish) and composition protocol called entish 1.0. They are based on software agent paradigm. The description language is the contents language of the messages that are exchanged (between agents and services) according to the composition protocol. The syntax of the language as well as the message format are expressed in XML. The language and the protocol are merely specifications. To prove that the technology does work, the prototype implementation is provided available for use and evaluation via web interfaces starting with www.ipipan.waw.pl/mas/ . Related work was done by WSDL + BPEL4WS + (WS-Coordination) + (WS-Transactions), WSCI, BPML, DAML-S, SWORD, XSRL, and SELF-SERV. Our technology is based on similar principles as XSRL, however the proposed solution is different. The language Entish is fully declarative. A task (expressed in Entish) describes the desired static situation to be realized by the composition protocol.

1 Our approach to service composition

Generally, there are two approaches to service composition. The first one is based on the assumption that services are composed, orchestrated, or choreographed in order to create sophisticated business processes, whereas the second one assumes that services are composed (typically on the fly) in order to realize clients' requests. Most of the existing technologies realize the first approach. The second approach is followed by academic projects, e.g., SWORD, XSRL, and our own project enTish. It seems that the service architecture corresponding to SOAP and WSDL is appropriate for the first approach. However, in our opinion, a different service architecture is required for realizing the second approach. The reason is that clients' requests are expressed in a declarative way in a formal language, so that it is natural to propose a universal protocol for the request realization. However, also in this case the service architecture based on SOAP and WSDL may be applied as it is done in XSRL[4].

* The work was supported partially by KBN project No. 7 T11C 040 20

We follow the idea of layered view of service architecture introduced in [2, 3]. Our service architecture comprises the following three layers: Conversation layer, functionality layer, and database management (executive) layer. The database management layer is the same as in [3], it influences the real world. However, the next two layers have different meaning. The functionality layer has exactly two interrelated components: Raw application, and so called filter associated with the raw application. Raw application implements a single operation, i.e., given input resources, it produces the output resource according to the operation specification. Note, that operation has exactly one output, although it may have several inputs. The associated filter works as follows. Given constrains on the output resource, it produces the constrains on the input resources. That is, given a specification of the desired output, the filter replies with properties that must be satisfied by the input in order to produce the desired output by the raw application. It is clear that these constrains must be expressed in one common language. The conversation layer implements a conversation protocol to arrange raw application invocation, as well as input / output resource passing to / from the raw application. The conversation protocol specifies the order for message exchange. Message contents is expressed in the common language.

Since our service architecture is different than the one that corresponds to WSDL and UDDI, we must revise the concept of service description language as well as the concept of service registry. It is natural that service description language should describe the types of service input / output resources as well as attributes of these types to express constrains. Note, that the language is supposed merely to *describe* resource types in terms of theirs attributes, not to construct data structures as it is done in WSDL. It is also natural to describe *What service does* in the language, i.e., the type of the operation the service performs. This type is expressed in terms of abstract function implemented by the operation. Usually, *What service does* is described in UDDI. We include this in our description language.

Since service has additional functionality performed by filter (i.e., a service may be asked if it can produce output resources satisfying some properties), the description language should be augmented with a possibility to formulate such questions as well as answers. Moreover, the clients' requests (tasks) should be expressed in the language.

We also want to describe some static properties of service composition process such as intentions, and commitments; this corresponds to the functionality of WS-Coordination.

The final requirement is that the language must be open and of distributed use. It means that names for new resource types, their attributes, and names for new functions, as well as for new relations can be introduced to the language by any user, and these names are unique (e.g., URIs). This completes the requirements for the description language called Entish. Since our technology is supposed to realize the declarative approach, we need a universal protocol for realizing the requests (tasks) specified in our description language.

For simplicity (i.e., for avoiding reasoning) as well as for making the prototype implementation feasible, we assume that the requests are extremely simple; in fact they are expressed as formulas that represent abstract plans and initial situations. In the next step of our project a *distributed* reasoning for plan generation will be implemented, so

that the requests will have a form of arbitrary formulas. The plan realization is done by the protocol called entish 1.0.

To prove that the requirements for the service description language and composition protocol can be satisfied we provide the prototype implementation available from http://www.ipipan.waw.pl/mas/ .

2 Walk-through example

The working example presented below constitutes an intuitive introduction to the description language and the composition protocol. The services described in the example are implemented and are ready for testing via the www interfaces.

A client was going to book a flight from Warsaw to Geneva; the departure was scheduled on Nov. 31, 2002. It wanted to arrange its request (task) by Nov. 15, 2002. With the help of TaskManager (TM for short), the client expressed the task in a formal language; suppose that it was the following formula:
phi =
"invoice for ticket (flight from Warsaw to Geneva, departure is Nov. 31, 2002) is delivered to TM by Nov. 15, 2002"

Then, the task formula (i.e., *phi*) was delegated to a software agent, say agent0. The task became the goal of the agent0. The agent0 set the task formula as its first intention, and was looking for a service that could realize it. First of all, the agent0 sent the query: *"agent0's intention is phi"* to a service registry called infoService in our framework. Suppose that infoService replied that there was a travel agent called FirstClass that could realize agent0's intention. Then, the agent sent again the formula *"agent0's intention is phi"* however, this time to the FirstClass. Suppose that FirstClass replied with the following commitment:
"FirstClass commits to realize phi,
if (order is delivered to FirstClass by Nov. 15, 2002 and
the order specifies the flight (i.e., from Warsaw to Geneva, departure Nov. 31,2002)
and one of the following additional specification of the order is satisfied:
(airline is Lufthansa and the price is 300 euro)
or (airline is Swissair and the price is 330 euro)
or (airline is LOT and the price is 280 euro))"

Let *psi* denote, the formula after *"if"* inside (...) parentheses. The formula *psi* is the precondition of the commitment. Once the agent0 received the info about the commitment, the agent0 considered the intention *phi* as arranged to be realized by FirstClass, and then the agent0 put the formula *psi* as its current intention, and looked for a service that could realize it. Let us notice that the order specified in the formula *psi* could be created only by the client via its TM, that is, the client had to decide which airline (price) should be chosen, and the complete order was supposed to include details of a credit card of the client. Hence, the agent0 sent the following message to TM: *"agent0's intention is psi"* Suppose that TM replied to the agent: *"TM commits to realize psi, if true "* The agent0 considered the intention *psi* as arranged to be realized by TM. Since the precondition of the TM commitment was the formula *"true"*, a workflow for realizing agent0's task was already constructed. Once TM created the order and sent it to FirstClass, the FirstClass would produce the invoice and send it to TM. It was supposed

(in the protocol) that once a service realized a commitment, it sent the confirmation to the agent0. Once the agent0 received all confirmation, it got to know that the workflow was executed successfully. In order to complete this distributed transaction, the agent sent synchronously the final confirmation to the all services engaged in the workflow. This completes the example. The complete enTish documentation is available at the project web site http://www.ipipan.waw.pl/mas/

References

1. S. Ambroszkiewicz. Entish: a simple language for Web Service Description and Composition, In (eds.) W. Cellary and A. Iyengar. Internet Technologies, Applications and Societal Impact. Kluwer Academic Publishers. pp. 289- 306, 2002.
2. Santhosh Kumaran and Prabir Nandi. Conversational Support for Web Services: The next stage of Web services abstraction. http://www-106.ibm.com/developerworks/webservices/library/ws- conver/?dwzone=webservices
3. F. Leymann and D. Roller.Workfbw-based applications. IBM Systems Journal, Volume 36, Number 1, 1997 Application Development http://researchweb.watson.ibm.com/journal /sj/361/leymann.html
4. Mike Papazoglou, Marco Aiello, Marco Pistore, and Jian Yang. XSRL: A Request Language for Web Services. http://eprints.biblio.unitn.it/archive/00000232/

Modularizing Web Services Management with AOP

María Agustina Cibrán, Bart Verheecke

System and Software Engineering Lab
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussels, Europe

{Maria.Cibran, Bart.Verheecke}@vub.ac.be

Abstract. Web service technologies accelerate application development by allowing the selection and integration of third-party web services, achieving high modularity, flexibility and configurability. However, current approaches to integrate web services in client applications do not provide any management support, which is fundamental for achieving robustness. In this paper we show how *Aspect Oriented Programming* (AOP) can be used to modularize service management issues in service oriented applications. To deal with the dynamic nature of the service environment we suggest the use of a dynamic aspect-oriented programming language called JAsCo. We encapsulate the management code in *aspects* placed in an intermediate layer in between the application and the world of web services, called Web Services Management Layer (WSML).

1. Introduction

Web services (WS) are modular applications that are described, published, localised and invoked over a network. Web services technologies accelerate application development by allowing the selection and integration of third-party web services, achieving high modularity, flexibility and configurability. However, current approaches only allow this integration by hard wiring the references to concrete web services into the client applications. As stated in [1], this leads to unmanageable applications that cannot adapt to changes in the business environment (e.g. a service that is abandoned or changed, a new service that becomes available on the market, etc). Moreover these approaches do not provide any management support, which is fundamental for achieving robustness. To deal with these issues, code has to be written manually and repeated for each service, resulting scattered in the application. We observe the need for the application to be independent of specific services.

The focus of this paper is to show how the modularization of service management issues can be enhanced by using dynamic *Aspect Oriented Programming* (AOP) [2] [3]. To deal with the dynamic nature of the service environment we suggest the use of a dynamic aspect-oriented programming language called JAsCo [4] [5]. We encapsulate the management code in *aspects* placed in an intermediate layer in between the application and the world of web services, called **Web Services**

Management Layer (WSML) [6]. In the next section we motivate the need for AOP and introduce JAsCo. In section 3 we show how JAsCo is ideal to modularize the management functionality of the WSML and provide some code examples. Finally, we present our conclusions in section 4.

2. WS Integration and Management as Crosscutting Concerns

The web service architecture is the logical evolution of object-oriented principles in a distributed context. Just as in object oriented approaches, the fundamental concepts of web services are encapsulation, message passing, dynamic building, interface description and querying. However, the distributed nature of web service applications leads to the emergence of various management concerns that are difficult to modularize using traditional software engineering methodologies.

First of all, we want to avoid hard wiring references to concrete services in the applications achieve high flexibility in the selection of services. By decoupling web services from the client application the concept of *most suitable service* is introduced. With current approaches it would be the responsibility of the application to decide which the most appropriate services are. This way, code for implementing service selection would be written at each point where some service functionality is required, resulting tangled and scattered in different places in the application. Thus, we need support for encapsulating this crosscutting code separated from the application and plug it in and out in a non-invasive way.

Moreover the selection of services also involves other management issues to be considered at the moment the services are integrated in the applications. For instance, services might need to control security, accounting, billing concerns at the time their functionality is requested. This also results in crosscutting code since the application developer would need to include this management code each time a service is requested.

Therefore, to avoid tangling the application code with service related code we identify the need for AOP. AOP states that some concerns of a system, such as synchronisation and logging, cannot be cleanly modularized using current software engineering methodologies, which leads to code duplication. To this end, AOP approaches introduce a new concept that is able to modularize crosscutting concerns, called an *aspect*. An aspect defines a set of *join points* in the target application where the normal execution is altered.

Using aspects to express the selection and management concerns as part of the WSML allows the application to remain independent of the service selection infrastructure. Moreover, we also pursue dynamism in the management of services and therefore an AOP technology that provides support for dynamic inclusion and removal of aspects is required. For this reason we introduce an aspect-oriented implementation language called JAsCo. JAsCo combines the expressive power of

AspectJ [7] with the aspect independency idea of Aspectual Components [8]. Originally JAsCo was designed to integrate aspect-oriented ideas into Component-Based Software Development [9]. However, JAsCo has some characteristics that are also useful in an object-oriented context:

- Aspects are described independently of a concrete context, making them highly reusable.
- JAsCo allows easy application and removal of aspects at run time.
- JAsCo has extensive support for specifying aspect combinations.

JAsCo introduces two concepts:

- **Aspect Beans:** specify crosscutting behaviour by defining hooks which specify *when* the normal execution of a method should be intercepted and *what* extra behaviour should be executed.
- **Connectors:** apply the crosscutting behaviour of the Aspect Beans specifying *where* the crosscutting behaviour should be deployed.

JAsCo enables the run-time plug in and out of connectors. This high flexibility and configurability is exactly what is needed for the management of web services. For more information about JAsCo we refer to [4], [5].

3. JAsCo Aspects in the WSML

3.1 Introducing WSML

In [6] we present an abstraction layer, called **Web Services Management Layer (WSML)**, which is placed between the application and the world of web services. It realises the concept of *just-in-time integration of services*: multiple services or compositions of services can be used to provide the same functionality.

Figure 1 illustrates the general architecture of the WSML. On the left side the core application resides, and if necessary, web service requests are issued to the layer. The WSML is responsible for choosing the most appropriate service or composition in a completely transparent way. This is realised by the **Selection Module** by considering different service properties. The collaboration with the **Monitoring Module** is required for this purpose as several properties of services might need observation over time.

Additional management functionality resides in the layer like traffic optimisation, billing**,** accounting, security, transaction, etc. The WSML is reusable in new applications and is completely configurable to avoid unnecessary overhead.

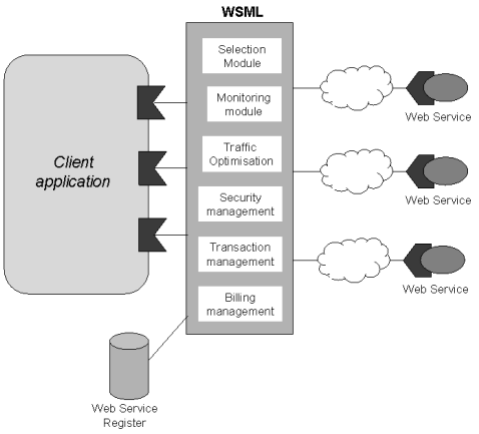

Fig. 1. General Architecture of WSML

The WSML has the following advantages:

- The application becomes more flexible as it can continuously adapt to the changing business environment and communicate with new services.
- Extracting all web service related code from the core application facilitates future maintenance of the code.
- Weakening the link between the application and the service enables hot swapping of services.

In the remainder of this section generic management aspects to deal with the crosscutting concerns will be presented.

3.2 Using Aspects for Service Redirection

Figure 2 shows how we implement the WSML using JAsCo aspect beans and dynamic connectors. A basic requirement is that hard-wiring services should be avoided. Therefore, service requests must be formulated in an abstract way at the left side of the layer and the WSML will be responsible for making the translation to a concrete service at the right side. The requests of the application are formulated in an abstract way as specified in an **Abstract Service Interface (ASI)**. This can be seen as a contract specified by the application towards the services. This way the syntactical differences between semantically equivalent services can be hidden. In order to enable this we introduce the concept of mapping schemas with sequence diagrams that unambiguously describe how the service or service composition maps to the ASI. An example of this mapping can be found in [6].

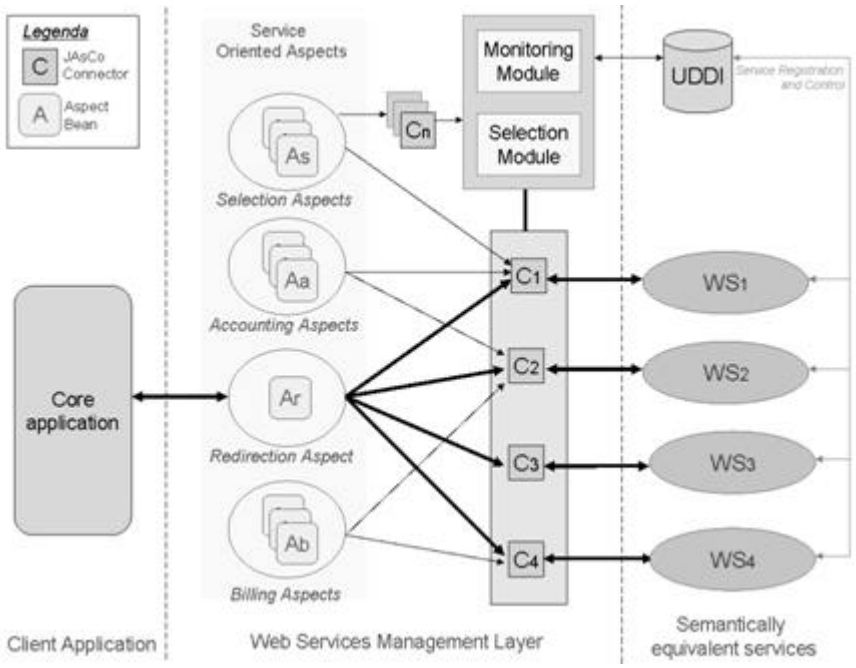

Fig. 2. Detailed Architecture of the WSML

To illustrate these ideas an example of a travel agency application is introduced. The application offers the functionality to book holidays online and customers can make reservations for both flights and hotels. To achieve this functionality the agency application integrates different web services. Suppose HotelServiceA and HotelServiceB are services that offer the same functionality for the online booking of hotels. Each hotel service returns exactly the same results.

Assume in the client-application a list of hotels needs to be shown to the customer. A HotelServiceInterface is defined with the following method for this purpose. `HotelList giveAvailableHotels(Date, Date, CityCode)`. At deployment time or at runtime the following two services are available: HotelServiceA provides the method: `giveHotels (CityCode, Date, Date,)`. HotelServiceB provides the method: `listHotels (Date, Date, CityName)`.

To make the mapping possible between the ASI and the concrete service interfaces, we make use of the aspect power of JAsCo and define an aspect in charge of redirecting the generic requests to the concrete services that will provide the functionality required. The redirection aspect defines the logic of intercepting the application requests and replacing them by a concrete invocation on a specific web service. Figure 3 shows the code for the redirection aspect. Note that this aspect is generic and does not refer to any concrete web service. The mapping to concrete web

services is specified in the connectors that deploy the redirection aspect. Several connectors can exist each in charge of deploying the redirection to a concrete web service. Figure 4 illustrates the deployment of the redirection aspect. The connector HotelServiceA specifies the mapping between the ASI `giveAvailableHotels (Date, Date, CityCode)` and the particular way to invoke that functionality on the web service HotelServiceA, that is invoking the method `giveHotels (CityCode, Date, Date,)`. To communicate with HotelServiceA the GLUE library is used [10].

```
class getAvailableHotelsRedirection {
 hook RedirectionHook {
  RedirectionHook(method (Date d1,Date d2,CityCode
cc)){
   call(method);
 }

 replace() {
  specificMethod(d1, d2, cc);
 }
 abstract public List specificMethod(
           Date d1,Date d2,CityCode cc);
 }}
```

Fig. 3. The Redirection Aspect Bean for hotel retrieval

```
static connector getAvailableHotelsOfServiceA {
 HotelServiceAStub hotelServiceA = null;
 try {
  hotelServiceA = HotelServiceAHelper.bind();
  // the stub is instantiated by analysing the WSDL-
  // file of hotelServiceA by using the GLUE library
  }
  catch(Exception e) {  }
 getAvailableHotelsRedirection.RedirectionHook rhook =
   new getAvailableHotelsRedirection.
      RedirectionHook(Application.
         giveAvailableHotels(Date, Date, CityCode){
 public List specificMethod(Date d1,Date d2,CityCode
cc){
  return hotelServiceA.giveHotels(cc, d1, d2));
 }
}}
```

Fig. 4. Connector that deploys redirection aspect

Each connector encapsulates the mapping between each generic request in the application and the concrete manner to solve that request in a specific service. Thus, there will be one connector for each different request that can be invoked by the

application. The WSML is responsible for the creation and management of these connectors. JAsCo allows the creation of connectors to be done dynamically. This characteristic enables the dynamic integration of new services. When the functionality of a new service has to be integrated in the application, a connector realizing the mapping for that service is created at run time. This is achieved transparently for the application.

3.3 Using Aspects for Service Management

As mentioned above, the layer can also deal with other management issues that need to be controlled at the application side. For instance, suppose the HotelServiceA describes a strategy for billing its use and the application wants to locally control this for auditing reasons. Suppose the service specifies that each time the method `giveHotels (CityCode, Date, Date)` is invoked, an amount of 2 euros has to be paid. We can achieve this in a non-invasive way by defining a new aspect that abstracts the logic for a "pay per use" billing strategy. Figure 5 shows the implementation of this aspect. Note that the redirection aspect is generic and can be deployed and customised for other services that adopt this billing policy. This deployment is specified as part of the connector shown in Figure 6. In this example, the billing is done when `getAvailableHotels` is invoked in the application. However, as connector `getAvailableHotelsOfServiceA` implements this method as a call to HotelServiceA, the billing is only done when this concrete service is used. Note that the hook can also be initialised with multiple functionalities provided by a web service.

```
class BillingPerUse {
  hook BillingHook {
    private int total = 0;
    private int cost = 0;

    public void setCost(int aCost){
      cost = aCost;
  }
  private void pay(){
    total = total + cost;
  }
  BillingHook(method (Date d1,Date d2,CityCode cc)) {
    call(method);
  }
after() {
    pay();
  }
}
}
```

Fig. 5. Billing Aspect

```
static connector getAvailableHotelsOfServiceA {
…
BillingPerUse.BillingHook billPerUse =
  new BillingPerUse.BillingHook(List
    Application.giveAvailableHotels(Date, Date,
CityCode));
  billPerUse.setCost(2);
  rhook.replace();
  billPerUse.after();
}
```

Fig. 6. Billing connector

The aspect BillingPerUse defines a billing template that can be reused by different services. Other more complex billing aspects can be formulated and implemented in a similar way.

This simple example illustrates that a generic library of aspects can be created to achieve high flexibility in the creation and manipulation aspects that implement other management issues.

4. Conclusion

In this paper we show how the use of AOP is needed to fully decouple service management concerns from the client applications. We propose to use a dynamic AOP implementation language JAsCo to enable hot-swapping and runtime management of services.

This approach has the advantage that applications become adaptable as they can easily integrate new services and dynamically accommodate to management requirements.

We are currently working on the definition of a library of reusable aspects that would allow the application developer to dynamically instantiate and configure the needed aspects to deal with different service management issues. We are also working on realising the hot swapping mechanism in a more intelligent way by considering service oriented rules. These rules are derived from the requirements the application specifies and are based on the non-functional properties of services.

5. References

[1] J. Malhotra, Ph.D., Co-Founder & CEO interKeel Inc., "Challenges in Developing Web Services-based e-Business Applications," Whitepaper, interKeel Inc., 2001

[2] Aspect-Oriented Software Development. http://www.aosd.net/

[3] Communications of the ACM. Aspect-Oriented Software Development, October 2001.

[4] D. Suvée and W. Vanderperren. "JAsCo: an Aspect-Oriented approach tailored for Component Based Software Development". Proc. of 2nd Int. Conf. on AOSD, Boston, USA, 2003.

[5] W. Vanderperren, D. Suvée, B. Wydaeghe and V. Jonckers. "PacoSuite & JAsCo: A visual component composition environment with advanced aspect separation features". Proc. of Int. Conf. on FASE, Warshaw, Poland, April 2003.

[6] B. Verheecke, M. A. Cibrán. "AOP for Dynamic Configuration and Management of Web services in Client-Applications". ICWS'03-Europe (submitted)

[7] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersen, J. Palm, and W. G. Griswold. "An overview of AspectJ". In Proceedings European Conference on Object-Oriented Programming, volume 2072 of Lecture Notes in Computer Science, pages 327--353, Berlin, Heidelberg, and New York, 2001. Springer-Verlag.

[8] Lieberherr, K., Lorenz, D. and Mezini, M. Programming with Aspectual Components. Technical Report, NU-CCS-99-01, March 1999. Available at: http://www.ccs.neu.edu/research/demeter/biblio/aspectual-comps.html.

[9] C. Szyperski. Component software: Beyond Object-oriented programming. Addison-Wesley, 1998.

[10] The Mind Electric, "The Glue Platform," 2003, http://www.themindelectric.com/glue/index.html

Services and Objects: Open issues

Vincenzo D'Andrea and Marco Aiello

Department of Information and Telecommunication Technologies
University of Trento
Via Sommarive, 14 38050 Trento
Italy
{dandrea,aiellom}@dit.unitn.it

Abstract. One of the common metaphors used in textbooks on Object-Oriented programming (OOP) is to view objects in terms of the services they provide, describing them in "service oriented" terms. This opens a number of interesting questions, moving away from the simple view of OOP as an implementation tool for Web Services. First of all: if an Object is a Service, can we also say that a Service is an Object?
While the short answers seems to be negative, there are several connections between the two concepts and it is possible to exploit the large repository of methodological tools available in OOP. What are the counterparts, in terms of services, of concepts like class or instance? Is it possible to apply techniques as containment or inheritance to services? What are interfaces, properties and methods for services? In this paper we try to start building some connections, underlining the open issues and the gray areas.

1 Introduction

One of the common metaphors used in textbooks on Object-Oriented programming (OOP) is to view objects in terms of the services they provide, describing them in "service oriented" terms (see for instance [3]). Building on abstraction and encapsulation, the key idea is to hide programming details that provide object functionalities. An interface describes these functionalities in terms of methods and properties, providing a logical boundary between operations invocations and their implementations. Then an object is just a "server" of its own methods. If on the one hand, this view is useful for educational purposes, on the other hand, it represents only a minor feature when compared to inheritance, polymorphism, code sharing, and so on.

If the object oriented paradigm is already 'service oriented' why is it then that we talk about a *new* computing paradigm with the advent of web services? Objects in OOP are already described as services, so is it because of the gaining momentum of web services that one describes this new trend as a shift in computing paradigm? To answer this let us consider more precisely what a web service is and what we mean by service orientation. In [6], Curberra et al. describe a web service in the following way:

> A Web service is a networked application that is able to interact using standard application-to-application Web protocols over well defined interfaces, and which is described using a standard functional description language.

The interfaces no longer hide units of code, but entire applications in a way closer to components [12]. In addition, the network plays a major role, with the consequences that web services have to deal with issues typical of distributed systems [5], such as: heterogeneity, openness, security, scalability, failure handling, concurrency, transparency. Web services are shifting perspective on programming and are now calling for a new term for programming. There seems to be consensus on the term *service oriented computing (SOC)*. A definition of SOC is in the "Service Oriented Computing Manifesto" [7].

> Services are autonomous platform-independent computational elements that can be described, published, discovered, orchestrated and programmed using XML artifacts for the purpose of developing massively distributed interoperable applications.

The SOC definition above generalizes the one of web services. One does not distinguish anymore among applications or components, but simply deals with computational elements. The *find-bind-use* model can summarize the idea of describing, publishing, discovering, orchestrating and programming the distributed computational entities. Standardization is explicitly mentioned and referred to XML-based languages.

In this position paper, we indicate some areas where web services may be contaminated by concepts and ideas from the object-oriented paradigm. We will base our analysis on abstract object-oriented concepts trying to avoid language peculiarities and tricks.

2 Similarities and differences

To justify a call for a paradigm shift, there must be some significant differences between object-oriented programming and service oriented computing. What we consider to be the key differences, among the many ones over which much hype has grown recently, are the following three:

OOP		SOC
invoke	vs.	*find-bind-use*
shared context	vs.	*multiple contexts*
synchronous method invocation	vs.	*asynchronous message passing*

Find-bind-use is the heart of service orientation. A software entity that needs a service from another entity first searches for available services, then decides

on the basis of some parameter among the available ones and only then binds it in order to use it. On the other hand, in object-oriented programming there is no search for service, but direct method invocation. The method must be provided by an object running at invocation time. The find-bind-use model allows for greater flexibility, especially in distributed environments, opening the road for the choice of services based on non-functional requirements, such as those ensuring quality of services.

In OOP the execution context is typically shared among all objects. Usually, objects are written in the same language, run on the same memory space and live for the execution span of the same program. Recent extensions allow for the objects to be distributed (e.g., Java RMI) and to be written in different languages (e.g., CORBA). These extensions go in the direction of service orientation, where everything is distributed and services live in heterogeneous multiple contexts. The operating systems in which web services live, the languages in which they are written, the middleware used for interoperation is completely transparent in the SOC model therefore we speak of *multiple contexts* of execution for interactive web service.

Finally, the interaction between objects through method invocation, which can be seen as a message passing mechanism, is synchronous. In open distributed environments a more flexible communication mechanism is often necessary, that is, the *asynchronous* communication among the software entities contributing to a computation. An example of asynchronicity is when one interacts with a web service by including an appropriate XML request inside an email.

If the above are the key mechanisms that differentiate between OOP and SOC, one may wonder at what is the different forms of abstractions that one considers when looking at SOC. In [3], Budd indicates how OOP realizes various forms of abstractions. Let us compare these with the SOC case.

Composition is a central issue in service oriented computing. A large amount of effort in research and industry is devoted to service composition. Some define ways to design the composition of service (e.g., [4, 13]) while others define how semantically annotated services can be automatically composed (e.g., [10]).

In Object Oriented systems, composition is a design activity and it is mainly a problem of statically designing the proper architecture of the system. The situation in Service Oriented computing is radically different: a service can build its functionalities upon others, for instance an e-commerce purchase service could include the actual purchase service, the shipping service and the insurance service. The composed services are not statically designed, the services and the supporting infrastructure are designed in terms of dynamically discovering the other services they need to include. In other words, the service paradigm provides the capabilities for dynamic, run-time composition rather than requesting a statically planned architecture.

The dynamic nature of composition has several consequences. Negotiation and contractual agreements cannot be accomplished off-line, they have to be dealt with at run-time. The role of catalogs and the discovery mechanism have no counterpart in the world of objects and components.

Services demand a transition from static binding between objects or components that are to be integrated to the dynamic binding of services. From the point of view of the design there is the need of a transition from designing an architecture to designing the *enabling medium*, that is, the infrastructure for runtime composition.

In object oriented systems, the term **inheritance** is used to describe the mechanism allowing the derivation of a class from another one. One may even distinguish between several forms of inheritance. The most common form is *specialization*; a class is defined in terms of specialization of a second one – this is expressed by the *is a* relationships (a TextWindow is a Window, i.e., the TextWindow has all the properties and behaviors of the Window). Specialization implies a semantic coherence between the two classes, one class is called a subtype of the other. Otherwise it is just a subclass, where the meanings attached to the interface can change. It is obvious that while a subclass must have at least some code differences with respect to the original class, a class that inherits in the sense of subtype can leave untouched the implementation details of the inherited class. In other words, the subclass requires a syntactical match, while the subtype implies also a semantical match between the involved classes.

The concept of subtyping is also related to a common distinction made between what is sometime referred to as "true" inheritance versus interface inheritance. The former is used when a class presents the same external interface *and* has access to the code of the inherited class, that is, the subclass is a subtype unless it overrides the behavior of the inherited one. The term interface is used when a class has the same external interface of the inherited one, but it has no direct access to its code. In this case, it became a subtype only when the behavior of the inherited class is reproduced with the same semantics.

In terms of implementation, a simplifying model is to view inheritance as a special form of composition. Composition generally implies wrapping the interface of the included classes, and filtering the communication between these classes and the external world. Inheritance can be described as if the inheriting class incorporates (composes with) the inherited one, but without filtering the communication; the inherited class can be accessed directly. An object of the inheriting class responds to the same invocations as an object of the inherited class. If the subclass is also a subtype, the results will also be the same.

To think at inheritance (subtyping) as a form of composition which maintains the interface (behavior) of the composed object, makes it easier to reason about similar concepts in the service world.

In OOP, **polymorphism** indicates an operation that can take operands of different type, i.e., objects of different classes. There are various kinds of polymorphism: parametric, inclusion, overloading and coercion.

Subtyping induces inclusion polymorphism. For instance, consider a class shape which has a method draw. The circle class, which subtypes the shape class, then also has a draw method. This allows to use a circle or a square object with the shape operations. One can then design a system relying only on the

methods of the inherited class; run-time binding mechanism will then call into use the proper object.

A similar concept is that of overloading. A symbol is overloaded when it is used for operations that have different semantics depending on the class of the operands (e.g., the '+' operator in Java which adds integers and concatenates strings).

In the service oriented architecture is hard to find equivalent notions, because a formal concept of typing and inheritance is missing.

Design **patterns** [8] are often connected with OO methodologies, especially for describing the interactions between the objects in a system. A Design Pattern is a well understood and proved solution to a design problem, such as creating a wrapper around an object or defining the interface between a client and a server. A pattern differs from an algorithm because it includes both procedures and architecture, described in a way resembling more a case study than a precise prescription.

This approach is quite effective and can be relevant for designing and developing individual services, but its application is more related to software engineering methodology while Service Oriented Computing appears to be more an information system engineering issue.

Other typical object oriented abstraction items to be found in [3] are (1) division into parts, encapsulation, interface and implementation, which directly map to SOC abstraction principles; (2) the service view which is exactly where the SOC emphasis lies; and (3) layers of specialization, history of abstraction, frameworks, which are not relevant in this first comparison between OOP and SOC.

3 Is a service an object?

We have so far seen that connections between objects and services are not at all new. Some references draw explicit links between the two concepts, e.g., "As a very rough approximation, one web service can be compared to one method in more traditional software context" [11]. Other connections are less evident; for instance in [1] the authors describe a methodology for defining what they call a "Compatible Service", that is an abstract description of a class of services, derived from concrete services description. While this generalization mechanism seems the opposite of creating an instance from a prototypical description, the concepts involved are quite similar.

In general, it is not immediate to identify in the world of services an analogy for the concepts of class and object. In OOP, a class is a category that represents a set of objects having the same characteristics, and an object is a concrete realization of a class – an *instance* of a class. While classes are stateless an instance of a class has a state which depends on the sequence of operations undergone by it. The object behavior in response to an external request is determined by the class. All the object derived from a class will respond in the same way in

response to the same invocation, provided they are in the same state, or the response does not depend on the state.

Are we now in the position to answer the question of whether a service is an object? We propose a negative answer to this question, but the analysis provided so far brings evidence to the fact that many similarities connect OOP and SOC. More object related concepts can move into the service oriented world in order to enhance the technology and, perhaps, clarify the role and scope of web services. Here are the most immediate example of concept migration:

Inheritance. Of the two concepts of inheritance for OOP, the interface inheritance seems to be the most immediate to apply to web services. Consider a payment service which could be subtyped in a service with acknowledgment of receipt. In a workflow, the former could be substituted by the latter as it is guaranteed that the same port types are implemented in the subtyped service.

Inheritance enables service substitution, service composition and it induces a notion of inheritance on entire compositions of services. Consider a workflow A built on a generic service and another one B with the same data and control links, but built on services which subtype the services of A. Could we say that B inherits from A or that B is a specialization of A?

Polymorphism. Both inclusion polymorphism and overloading can be extended to the service paradigm. A composition operation in a workflow may have different meanings depending on the type of the composed services. For example, composing a payment and a delivery service may have a semantics for which the two services run in parallel; on the other hand, the composition of two subtyped services in which the payment must be acknowledged by the payers bank and the delivery must include the payment transaction identifier have the semantics of a sequencing the execution of the services.

Composition. A formal and accepted notion of composition is currently missing in the SOC domain and, as just proposed, inheritance and polymorphism could induce such precise notions of composition over services. Could this help dissolve the fog around the meaning of composition for web services? Could this bring together "syntacticians" which claim that nothing can be composed if not by design with "semanticians" which claim that anything can be composed automatically? Perhaps not, but it could fill some of the gaps left by standards which do not have a clear semantics, most notably, BPEL [2] which is proposing itself as the standard for expressing aggregations of web services.

Statefullness. Finally, the difference between stateless class definitions and statefull objects in OOP can impact web services technology where services are stateless entities. Web services resemble more to class definitions, but the notion of an existing instance of a service with its state is paramount. Software entities need to access each others state in order to fully interoperate. Here the parallel is with the history of HTML pages. When first introduced HTTP/HTML interactions were stateless, but this limited by far the client-server communication. It did not take long before the introduction of statefull interactions via the invention of 'cookies' [9].

The object oriented paradigm has a solid formal background and is a well-established reality of today's computer science. Service oriented computing is, on the other hand, a new emerging field, which tryies to realize global interoperability between independent services. To meet this goal, service oriented technology will need to solve a number of challenging issues, such as how to manage precise service semantics. One way to attack this problems is by 'borrowing' concepts from the object oriented world. In this paper we presented a parallel between objects and services that might be somewhat arguable, but one cannot dispute that services exhibit a number of object-like behaviors. Our focus has been on inheritance and polymorphism for composition semantics and we have also stressed the need of state information for services, but we do believe that there is space for even further contamination between object oriented methodologies and service oriented computing.

References

1. V. De Antonellis, M. Melchiori, B. Pernici, and P. Plebani. A methodology for e-Service substitutability in a virtual district environment. In J. Eder and M. Missikoff, editors, *CAiSE 2003*, pages 552–567, 2003.
2. BEA, IBM, Microsoft, SAP AG, and Siebel. Business Process Execution Language for Web Services, 2003. http://www-106.ibm.com/developerworks/library/ws-bpel/.
3. T. Budd. *An Introduction to Object-Oriented Programming*. Addison Wesley, 2002. (3rd edition).
4. F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M.-C. Shan. eFlow: a platform for developing and managing composite e-services. Technical report, Hewlett Packard, 2000.
5. G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design*. Addison Wesley, 2001. (3rd edition).
6. F. Curbera, W. Nagy, and S. Weerawarana. Web services: Why and how. In *Workshop on Obejcet Orientation and Web Services OOWS2001*, 2001.
7. M. Papazoglou et al. SOC: Service Oriented Computing manifesto, 2003. Working draft available at http://www.eusoc.net.
8. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable ObjectOriented Software*. Addison-Wesley, 1995.
9. D. Kristol. HTTP cookies: Standards, privacy, and politics. *ACM Transactions on Internet Technology (TOIT)*, 1(2):151–198, 2001.
10. S. McIlraith and T. Son. Adapting Golog for composition of semantic web services. In *Proc. of the Int. Conf. on Knowledge Representation and Reasoning (KR2002)*, pages 482–493, 2002.
11. G. Piccinelli, A. Finkelstein, and C. Nentwich. Web service need consistency. In *Workshop on Obejcet Orientation and Web Services OOWS2002*, 2002.
12. C. Szyperski. *Component software: beyond object-oriented programming*. Addison-Wesley, ACM, 1998.
13. J. Yang and M. Papazoglou. Web component: A substrate for web service reuse and composition. In *CAiSE*, pages 21–36, 2002.

UML Modelling of Automated Business Processes with a Mapping to BPEL4WS

Tracy Gardner

IBM UK Laboratories, Hursley Park, Winchester, SO21 2JN, UK
tgardner@uk.ibm.com

Abstract. The Business Process Execution Language for Web Services (BPEL4WS) provides an XML notation and semantics for specifying business process behaviour based on Web Services. A BPEL4WS process is defined in terms of its interactions with partners. A partner may provide services to the process, require services from the process, or participate in a two-way interaction with the process.
The Unified Modeling Language™ (UML™) is a language, with a visual notation, for modeling software systems. The UML is an OMG™ standard and is widely supported by tools. UML can be customized for use in a particular modeling context through a 'UML profile'. We describe a UML Profile for Automated Business Processes which allows BPEL4WS processes to be modeled using an existing UML tool. We also describe a mapping to BPEL4WS which can be automated to generate web services artifacts (BPEL, WSDL, XSD) from a UML model meeting the profile.

1 Introduction

As service-oriented technology gains in popularity, it will be increasingly necessary to be able to design large-scale solutions that incorporate web services. The Unified Modeling Langauge™ (UML™) is widely used in the development of object-oriented software and has also been used, with customizations, for component-based software, business process modelling and systems design. UML provides a visual modeling notation which is valuable for solution design and comprehension. UML can be customized to support the modelling of systems that will be completely or partially deployed to a web services infrastructure. This enables the considerable body of UML experience to be applied to the maturing web services technologies. This paper introduces a UML profile (a customization of UML) which supports modelling with a set of semantic constructs that correspond to those in the Business Process Execution Language for Web Services[1] (BPEL4WS)[1].

Using UML primarily as a documentation tool has a real but limited benefit, and it is recognized that UML models developed for this purpose may not be maintained when a project is under severe time pressure. The value of UML-modelling of systems has the potential to increase significantly through the emergence of initiatives such as model-driven development and architected RAD [3]

[1] The current version of the profile is based on BPEL4WS version 1.0.

which enable executable systems to be generated automatically from detailed models. This approach is employed here to provide a mapping from models conforming to the UML profile for automated business processes to executable BPEL processes.

2 The UML Profile for Automated Business Processes

This section introduces a subset of the UML profile through an example that defines a simple purchase order process. A complete specification of the profile can be found in [2]. The example used here is taken from the BPEL 1.0 specification:

"On receiving the purchase order from a customer, the process initiates three tasks in parallel: calculating the final price for the order, selecting a shipper, and scheduling the production and shipment for the order. While some of the processing can proceed in parallel, there are control and data dependencies between the three tasks. In particular, the shipping price is required to finalize the price calculation, and the shipping date is required for the complete fulfillment schedule. When the three tasks are completed, invoice processing can proceed and the invoice is sent to the customer."

BPEL processes are stateful and have instances so in BPEL this scenario is implemented as a PurchaseOrder process which would have an instance for each actual purchase order being processed. Each instance has its own state which is captured in BPEL variables. In the UML profile, a process is represented as a class with the stereotype <<Process>>. The attributes of the class correspond to the state of the process (its containers in BPEL4WS 1.0 terminology). The UML class representing the purchase order process is shown in Figure 1.

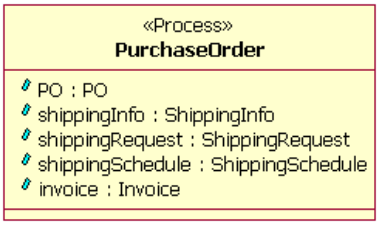

Fig. 1. A UML class used to model a BPEL process

The behaviour of the class is described using an activity graph. The activity graph for the purchase order process is shown in Figure 2. The partners with which the process communicates are represented by the UML partitions (also known as swimlanes): customer, invoiceProvider, shippingProvider and schedulingProvider. Activities that involve a message send or receive operation to a partner appear in the corresponding partition. The arrows indicate the order in which the process performs the activities.

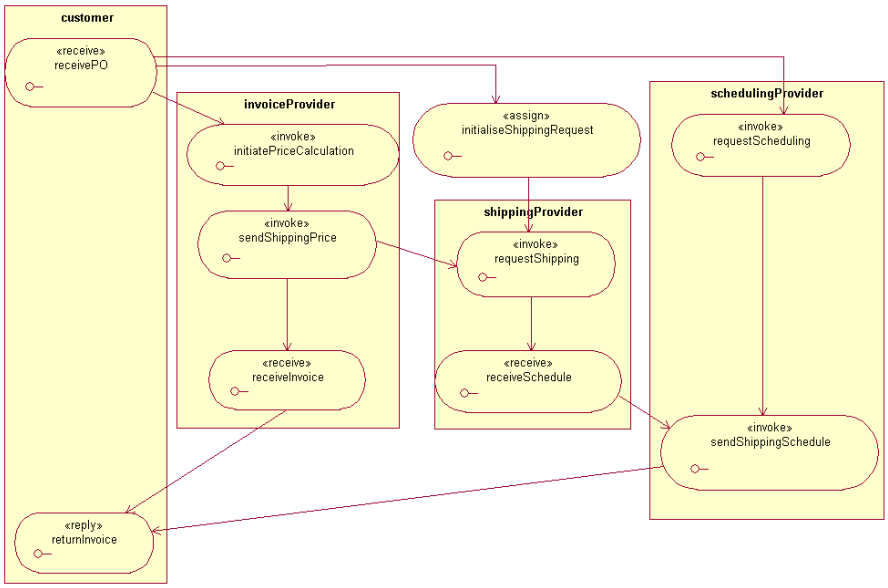

Fig. 2. Activity graph for the purchase order process with detail elided

The purchase order process begins by receiving a purchase order request from a customer. The initiatePriceCalculation, initialiseShippingRequest and requestProductionScheduling activities begin executing, triggering further activities as they complete. The arrows on the graph indicate control links, an activity starts when all of its preceding activities have completed. Note that the requestShipping activity requires that both the initialiseShippingRequest and sendShippingPrice activities have taken place before it begins. The returnInvoice activity returns a response back to the customer. Each activity has a descriptive name and an entry action detailing the work performed by the activity. Note that in Figure 2, the detail of the actions is hidden on the diagram due to space constraints. For a full explanation of the detailed expression of actions please refer to [2].

3 Mapping to BPEL4WS

The UML profile for automated business processes is sufficiently expressive that complete executable BPEL4WS artifacts can be generated from UML models. Table 1 shows an overview of the mapping from the profile to BPEL4WS (version 1.0) covering the subset of the profile introduced in this paper.

A cutdown version of the BPEL document that would be generated from the purchase order example in this paper is shown in Figure 3 (much of the detail is omitted here due to space constraints).

Table 1. UML to BPEL4WS mapping overview.

<<Process>> class	BPEL process definition
Activity graph on a <<process>> class	BPEL activity hierarchy
<<process>> class attributes	BPEL containers
Hierarchical structure and control flow	BPEL sequence and flow activities
<<receive>>, <<reply>>, <<invoke>> activities	BPEL receive, reply, invoke activities

```
<process name="purchaseOrderProcess" ...>
   <containers>
      <container name="PO" messageType="lns:POMessage"/>
      <container name="Invoice" messageType="lns:InvMessage"/>
      ...
   </containers>
   ...
   <sequence>
      <receive partner="customer"
               portType="lns:purchaseOrderPT"
               operation="sendPurchaseOrder"
               container="PO">
      </receive>
      ...
      <reply partner="customer" portType="lns:purchasePT"
             operation="sendPurchaseOrder"
             container="Invoice"/>
   </sequence>
</process>
```

Fig. 3. BPEL extract corresponding to the purchase order process.

4 Proof of Concept Demonstrator

A technology demonstrator supporting an end-to-end scenario from a UML tool (such as Rational™ XDE™) through to a BPEL4WS runtime (BPWS4J) is available from IBM™ alphaWorks™ as part of the Emerging Technologies Toolkit [4]. The mapping implementation is built using the Eclipse Modeling Framework (EMF) and takes the industry standard file format for exchange of UML models (XMI) as input. BPEL4WS artifacts along with the required WSDL and XSD artifacts are generated.

5 Conclusion

This paper has introduced a UML profile for automated business processes with a mapping to BPEL4WS. This approach enables service-oriented BPEL4WS components to be incorporated into an overall system design utilizing existing soft-

ware engineering practices. Additionally, the mapping from UML to BPEL4WS permits a model-driven development approach in which BPEL4WS executable processes can be automatically generated from UML models. A proof of concept demonstrator for the mapping is available. Future work includes the implementation of a reverse mapping to support the import of existing BPEL4WS artifacts and the synchronization of UML models and BPEL4WS artifacts with changes in either being reflected in the other. The profile and mapping currently support the 1.0 version of the BPEL4WS specification, support for BPEL4WS 1.1 is planned.

Acknowlegements

Thanks to Gary Flood and Keith Mantell for comments on this paper.

References

1. BEA Systems, IBM, Microsoft: Business Process Execution Language for Web Services, Version 1.0. IBM developerWorks (2002). Available from http://www-106.ibm.com/developerworks/webservices/library/ws-bpel1/
2. Gardner, T et al.: Draft UML 1.4 Profile for Automated Business Processes with a mapping to the BPEL 1.0. IBM alphaWorks (2003). Available from http://dwdemos.alphaworks.ibm.com/wstk/common/wstkdoc/services/demos/uml2bpel/README.htm.
3. Selic, B: The Pragmatics of Model-Driven Development. IEEE Software special issue on Model-Driven Architecture (2003). (To be published.)
4. Emerging Technologies Toolkit. IBM alphaWorks (2003). Available from http://www.alphaworks.ibm.com/tech/ettk/

A Classification Framework for Approaches and Methodologies to make Web Services Compositions Reliable

Muhammad F. Kaleem

Technical University Hamburg-Harburg
m.kaleem@tuhh.de

Abstract: Individual web services can be composed together to form composite services representing business process workflows. The value of such workflows is directly influenced by the reliability of the composite services. There is considerable research concerned with reliability of web services compositions. There is, however, no clear definition of reliability that establishes its scope in the context of a composite service. We propose a definition of composite service reliability in this paper that takes into account different aspects affecting reliability of a composite service. We also put this definition to use as the basis of a framework that can be used for classification of approaches and associated methodologies for making composite services reliable. The framework can be used to identify which aspect of composite service reliability is addressed by a particular approach. We will also reference some selected methodologies to classify them according to the aspect of reliability they address. A definition of composite service reliability and its use to classify methodologies for composite service reliability as described in this paper will prove useful for comparing and evaluating methodologies for making composite services reliable, and will also have a bearing on the quality of service aspects of architectural styles and methodologies of software solutions based on web services compositions.

Introduction

Web services represent autonomous services with clear service definitions. Interaction with web services is possible through their service definitions, which can be made available in WSDL [1]. Individual service definitions may represent limited business functionality. However, it is possible to compose functionality offered by different individual services, likely from different service providers, into a composite service representing a complete business process. A number of standards exist for composition of autonomous, individual web services into a composite service representing a workflow [2-5].

Given that a composite service will most likely comprise of a number of autonomous web services, the reliability of the workflow represented by the composite service becomes significant. Reliability of composite services is the subject of much research. There is, however, no clear definition of reliability that establishes its scope in the context of a composite service. Also, there is diversity of research in

this area, but different research approaches for ensuring reliability of composite services cater to a definition of reliability peculiar to that approach.
In this paper we present a definition that identifies main areas of composite service reliability, and lists the reliability aspects related to these areas. We will use this definition as the basis of a framework that can be used to classify different approaches that may be used to address the reliability aspects identified in the definition. We will also reference some selected methodologies complementing these approaches, gleaned from research and industry proposals, and classify them using the framework according to the aspect of composite service they address.

Definition and its use as a classification framework

We present a definition of composite service reliability that is split into two parts, so as to identify two main areas of composite service reliability. We also list the reliability aspects related to these areas. We will then provide more details about these aspects in the next sections, when we use the definition as a framework for classification of methodologies for composite service reliability. To this effect, we will first describe the approaches that may be taken to address each aspect of reliability, and then reference some selected solutions that provide methodologies complementing these approaches. We reference solutions from research and industry, and in this way classify the methodology according to the particular aspect of composite service reliability it addresses. With the help of the definition and using it as a basis of a classification framework, it is therefore possible to classify a solution according to the aspect of composite service reliability it addresses.

The two main areas of composite service reliability, and the reliability aspects related to these, are:

1. Reliability of composition
   – *that the composite service is correctly specified*

   This aspect requires that the notation for specifying a web services composition is correct, and that the composite service specification, when put into execution, translates to correct process flow which it represents.

   - *Approaches*

     A number of standards exist for composition of web services, as we mentioned previously. These standards describe how individual web services can be composed together to form an executable business process. For the composite service to reliably represent a business process, it is important that the composite service specification is correct. However, correct notation for specifying the composite service alone is not sufficient to guarantee reliability. It is also important that the specification representing the web services composition translates into an error-free process flow when the composite service executes. It is important to address both of these points.

   - *Possible Solutions*

     Specifying a web services composition according to a composition standard can be a complex task. This can, however, be facilitated by tools typically provided by implementers of web services composition standards. An additional advantage

such tools can provide is checking whether the specification notation is correct, and thereby identify potential sources of error. As an example, we may mention the commercial implementation [6] of the web services composition standard [2], which provides graphical tools to visually compose a composite service definition.

Once the composite service has been specified, it can be checked whether the specification will translate into correct process flow on execution. A possible solution for this activity is described in [7], which uses model checking techniques to verify the workflow specification created according to a particular web services composition standard. Another technique for verification of composite e-services is described in [8]. Even though the work described in [8] is not directly related to web services compositions, the methodology described can be useful for further work into composite service reliability.

– *that the composite service consists of functional interface definitions*

This aspect deals with the requirement that all individual web services making up the composite service are available and functional according to their interface definitions, which may have been obtained from an online registry or resource.

- *Approaches*

A composite service may be composed dynamically, with requisite web services being added as they are needed. The information about a service's interface may come from a registry, or any other online resource. However, it is possible that the requisite service itself does not exist any more. This aspect of reliability is also relevant if we take into account the fact that a composite service may exist for a long period of time, during which time a service may go offline, or the service provider may not offer it anymore. The composite service should be able to handle such a situation. The approach should be to address this aspect at composition time, so that the composite service consists of functional individual services, and errors during the execution of the composite service due to unavailability of individual services are prevented. However, the problem represented by this aspect is relevant during composite service execution as well, when an individual service may go down (due to software or hardware failure, for example), and an approach for addressing this aspect of reliability has to take this factor into account as well.

- *Possible Solutions*

Implementers of web services composition standards could provide tools for checking whether all individual web services making up the composite service are functional. Similarly, the implementations of the standards could provide a test framework that checks the availability of the web services implementation at composition time. The advantage of doing this at composition time is to avoid potential sources of error during composite service execution. However, even during composite service execution, the breakdown of an individual service will present a problem. This can be dealt with methodologies explained later when we go over the aspects related to execution of the composite service.

– *that the composite service specification is conformant to the specification of individual web services*

This aspect requires conformance between the process flow requirements represented by the composite service and the constraints expected by the component web services.

- *Approaches*

A composite service could comprise of individual web services from different service providers, and each individual service may have its own business service policy that drives its interaction with its users. This implies that used individually, the service may be able to impose constraints on its usage. As part of a larger composition, the constraints expected by the individual services may conflict with the business service policy of the business process represented by the composite service. It is important to avoid this for the reliability of the web services composition. Similarly, but seen from a different viewpoint, individual web services should conform to the business policy represented by the composite service so that that this aspect of reliability is addressed.

- *Possible Solutions*

The web services composition standards can allow specifying constraints on the usage patterns of individual web services that are commensurate with their business service policies, so that there are no inconsistencies between the composite service and the individual services it is composed of. For example, the composition standard [2] has the notion of abstract and executable processes, where abstract processes may be used to capture behavioural aspects of services. It is also possible to address this aspect of composite service reliability at a level other than that of the composition standard. [9] suggests an experimental methodology that is relevant in this regard. However, similar methodologies will have to be refined and enhanced in scope before they can be applied to making composite services reliable. It is also possible to allow a web service to specify its policies and characteristics with respect to the interface it exposes to the outside world, so that these policies are taken into account during the web service composition process. We are working on a framework that allows a web service to specify its policies with respect to its participation in a composite service using the set of standards [10-12]. The framework also allows a web service composition provider and the web service provider to negotiate participation in the composite service through an enrolment process. The enrolment process also helps cover the reliability aspect mentioned in the previous section, related to functional interface definitions.

– *that the composite service can handle interface definition changes within individual services gracefully*

This reliability aspect deals with the ability of the composite service to handle interface changes within individual services without producing unexpected errors.

- *Approaches*

As was mentioned previously, a composite service may be in operation for a long period of time. Given the autonomous nature of service providers, it is possible that a service provider may change the interface of a service that is part of the composite service. This should not cause the breakdown of the business process represented by the composite web service, or lead to unexpected errors. Therefore ability to deal with such interface definition changes, called graceful handling in the definition, is important for the reliability of the composite service. Possible approaches could be extensions in the web services composition standards to allow for graceful handling of interface changes within individual services in the

composite service, or providing a layer of functionality on top of the composition layer, which takes care of interface changes.

- *Possible Solutions*

  It is possible to build a layer of functionality on top of the composition standards implementation to address this issue, as described in [13]. This approach uses a conversational model of interaction between web services for graceful handling of interface changes in individual web services. Another solution could be facilitated by negotiated enrolment of the individual web service in the composite service (as mentioned in the previous section), where, for example, the web service may be bound by contract not to change its service interface.

2. Reliability of execution
   – *that the composite service execution is consistent with the business process flow it represents, and there are appropriate fault-handling mechanisms in place*

This aspect covers the reliability issues that are relevant when there is mutual interaction between a composite service (also within services of which the composite service is composed of) and external users and services as part of a business process flow scenario.

- *Approaches*

  Fault-handling mechanisms can deal with errors that may occur during the execution of the composite service in an appropriate manner. The appropriate manner would depend on the kind of the error and its effect on the composite service execution. For a simple error, the fault-handling approach may just be to log the error, but for an error that can lead to data corruption, for example, a fault-handling mechanism may have to restore the state of the data to the one before the error, and may have to take compensatory actions to achieve this. A general approach to handle such requirements is to use transactional mechanisms. Furthermore, since interactions between participants in a web services composition are expected to represent long-running transactions, where application of traditional rollback mechanisms is not possible, compensatory actions to reverse the effect of an activity that has already completed play an important role in fault-handling for composite services.

- *Possible Solutions*

  Due to the loosely-coupled interaction between web services, and the autonomy of service providers, requirements for transaction management in a web services environment are different from traditional transaction management requirements. Therefore traditional transactional mechanisms cannot be applied directly to web services. There are standards available [14, 15] that can be used to enable transactions over web services. These standards are the result of industry efforts. From among these, [14] is closely related to a web services composition standard [2], and therefore provides a ready model for using transactional behaviour to address reliability aspects during composite service execution. However, the standards for transactions over web services are quite new, and there is still some way to go for easy applicability of these standards to practical situations. An interesting comparison of these standards is presented in [16].

  There are research proposals [17, 18] catering to this area of reliability as well. [17] proposes a framework for building transactional compositions of web

services. With respect to the definition of composite service reliability that we presented, the scope of this framework relates to the reliability of composite service execution. [18] seeks the same objective as [17], that being building reliable web services compositions on top of autonomous web services, however the approach it takes is different. The framework proposed by [18] consists of a multi-layered architecture and a transaction model for building reliable composite services, and in general covers a wider spectrum of reliability issues related to compositions of web services.

Using compensation to reverse the effect of completed activities has a bearing on reliability of execution, and hence the overall reliability of the composite service. Compensation is, however, a complex task, and requires coordination between all participants in a composite service, in the form of correct specification of compensatory activities in the composite service on part of the composite service provider, and the provision of compensatory activities on part of the web service provider.

– *that the messages flowing between web services are reliably delivered*

- *Approaches*

Reliable delivery of messages flowing between web services is an important factor that influences correct execution of the composite service, thereby having a bearing on the overall reliability of the composite service. The approach for reliable message delivery should be to ensure that messages from a sender should reach the intended recipient despite any software and hardware failures along the way.

- *Possible Solutions*

There have been a number of efforts into reliability of message delivery between web services. We will not list these here, except for a recent industry proposal in the form of a set of web services standards [19] that provide for reliable message delivery. Among these standards is [20], which specifies a model that provides the guarantee that messages sent by the initial sender will be delivered to the ultimate receiver.

Conclusion

In this paper we presented a definition of composite service reliability that modularises reliability concerns into two main areas and identifies different aspects affecting reliability related to these areas. This definition is incipient, and as part of our research, we plan a systematic evaluation of this definition, so as to substantiate and further refine it, as well as to verify and possibly enhance its scope. Similarly, we plan to evaluate the classification framework presented in this paper as well, so as to enhance it, and verify its usefulness. We also described some possible approaches to address these aspects of composite service reliability, and then referenced selected works that propose methodologies complementing the approaches. This was done to demonstrate how the definition may be used as a classification framework, and it was shown how a particular methodology could be classified according to the aspect of reliability it addresses.

We believe a comprehensive definition of composite service reliability, and the classification of approaches and methodologies according to the aspect of composite service reliability they address is important for further research into this subject. A definition and classification framework can help with the comparison and evaluation of different methodologies for composite service reliability, based on the aspect(s) of reliability they address. Given that there are at present different standards for web services composition, and different approaches for ensuring reliability of composite services, such comparison and evaluation is essential.

The selected methodologies that we presented deal with reliability of composite services. However, through the use of the definition and classification framework presented in this paper, it can be seen that these address particular aspects of composite service reliability. The definition and classification framework as presented in this paper should help establish the context in which research work on composite service reliability is performed.

Another important use of the definition of composite service reliability relates to the view of web services as components, and composite services as orchestrations of components. Since components and their composition touches upon the issue of re-use, a comprehensive definition of composite service reliability can help identify issues that could affect reliability when reusing web services components.

References

1. *Web Services Description language (WSDL).* http://www.w3.org/TR/wsdl
2. *Business Process Execution Language for Web Services, version 1.1*. http://dev2dev.bea.com/technologies/webservices/BPEL4WS.jsp
3. *Business Process Modelling Language*. http://www.bpmi.org/bpml-spec.esp
4. *Web Service Choreography Interface (WSCI).* http://www.w3.org/TR/wsci/
5. *XML Process Definition Language*. http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf
6. Collaxa Inc. *Collaxa BPEL Orchestration Server*. http://www.collaxa.com
7. Shin Nakajima. *Model-Checking Verification for Reliable Web Service*. OOPSLA 2002 Workshop on Object-Oriented Web Services. 2002.
8. Xiang Fu, Tevfik Bultan, and Jianwen Su. *Formal Verification of E-Services and Workflows*. Workshop on "Web Services, e-Business, and the Semantic Web (WES): Foundations, Models, Architecture, Engineering and Applications". 2002.
9. Giacomo Piccinelli, Anthony Finkelstein, and Christian Nentwich. *Web Services Need Consistency*. OOPSLA 2002 Workshop on Object-Oriented Web Services. 2002.
10. *Web Services Policy Framework (WSPolicy), version 1.1*. http://www-106.ibm.com/developerworks/library/ws-polfram/
11. *Web Services Policy Assertions Language (WS-PolicyAssertions), version 1.1*. http://www-106.ibm.com/developerworks/library/ws-polas/
12. *Web Services Policy Attachment (WSPolicyAttachment), version 1.1*. http://www-106.ibm.com/developerworks/library/ws-polatt/

13. Santhosh Kumaran and Prabir Nandi. *Conversation Support for Web Services*. Accessed on: 15 June 2003. http://www-106.ibm.com/developerworks/webservices/library/ws-conver/
14. *Web Services Transaction (WS-Transaction).* http://www-106.ibm.com/developerworks/webservices/library/ws-transpec/
15. *OASIS Business Transaction Protocol (BTP).* http:/ www.oasis-open.org/committees/business-transactions/
16. Roger Sessions. *Shootout At The Transaction Corral; BTP Versus WS-T.* Accessed on: 15 June 2003. http://www.objectwatch.com/issue_41.htm
17. Thomas Mikalsen, Stefan Tai, and Isabelle Rouvello. *Transactional Attitudes:Reliable Composition of Autonomous Web Services*. International Conference on Dependable Systems and Networks (DSN 2002). 2002.
18. Paulo F. Pires, Marta Mattoso, and Mário Benevides. *Building Reliable Web Services Compositions*. NET.Object Days Conference (WS-RDS'02). 2002.
19. IBM and Microsoft. *Reliable Message Delivery in a Web Services World: A Proposed Architecture and Roadmap*. Accessed on: 15 June 2003. ftp://www6.software.ibm.com/software/developer/library/ws-rm-exec-summary.pdf
20. *Web Services Reliable Messaging Protocol (WS-ReliableMessaging).* http://dev2dev.bea.com/technologies/webservices/ws-reliablemessaging.jsp

Requestor Friendly Web Services

Ravi Konuru and Nirmal Mukhi

IBM Research,
19 Skyline Drive, Hawthorne, NY 10591, USA
Email: {rkonuru, nmukhi}@us.ibm.com

Abstract. Web service providers rely on the Web Service Description Language (WSDL) as the way to communicate information about an available service to a service requestor. This description or meta-data of the service is used by a service requestor to inspect the available interfaces and to access the service. In this paper, we argue that publishing a WSDL with a functional description of a service alone is not requestor friendly, i.e., it does not allow the requestor any flexibility in improving the end-to-end responsiveness and customize the Web Service behavior. We present some scenarios to back this argument and also outline a spectrum of solution approaches.

1 Introduction

Web services efforts, to a large extent, are being driven by the industry to solve business problems. Business needs demand seamless integration of information within and across enterprises to improve operational efficiency in terms of time and resources used. Information flow and updates must happen in a composite context of security, transactionality, processes, workflow, etc. Efforts are underway to specify and create new generation of systems based on Web services standards that abstract these concepts and incorporate security, transactions, orchestration and choreography, grid computing capabilities, business documents and processes, simplified integration and mapping to existing middleware systems and application assets. In other words, the Web services framework is defining a unified architecture and software infrastructure that can support the transformation of existing IT assets into services which businesses can use and integrate to perform effectively and efficiently in a heterogeneous, dynamic, distributed, multi-domain environment.

Service metadata is a core concept in the Web services framework. This metadata is used by service providers to generate partial service implementations and configure supporting middleware and by service requestors to generate the necessary client paraphernalia to communicate with a service. Traditional distributed object systems such as CORBA [1] rely on interface descriptions as being sufficient descriptions of a distributed object. Web services framework implementations have consciously or unconsciously adopted this notion for services and as a result rely on a WSDL [2] document as being an adequate rendition of the service for requestors to use.

The primary purpose and contribution of this paper is to motivate the need for giving service requestors access to a more complete metadata description of a Web service, covering aspects beyond the interfaces and protocols. This can allow service requestors to reason about a Web service and make smarter decisions when it comes to using it, resulting in improved responsiveness and customization. We describe various scenarios where detailed service metadata can provide superior value. Finally, we outline our thoughts on a spectrum of approaches to address this issue.

2 Scenarios

Currently, given a WSDL description, a client side tool or infrastructure is able to either generate a proxy or act as a universal proxy that can communicate with the web service. The advantage of this approach it simplifies service development by completely factoring out the infrastructure available at a service requestor's end from the service provider. This method of using a web service from a client must always be supported. However, the argument here is that it this isolation between the provider and the requestor comes with certain costs:

Since the proxy has no knowledge of the service beyond the service description, every call to the web service must necessarily perform a round trip to the provider. As a result, the proxy cannot leverage local resources (locally known Web services that provides better quality of service, available CPU cycles, etc.) when acting on behalf of the service.

Since all requestors are provided with the same coarse-grained view of the service, there is no direct support for customization of the service based on an individual requestor's execution context and its requirements.

In this section, we present scenarios where giving the service requestor prominence in the definition of the service metadata as well as service infrastructure can result in new levels of functionality and performance.

2.1 Responsiveness

Consider a web service that is being used to interactively query and update a catalog. The query interface is via an interactive forms interface where the user can enter input and select various criteria. A form submit translates to conceptually a new query or update on the catalog. In addition to the types of individual inputs, the catalog update and query governed by several rules that express the relationship among the inputs. For example, one rule might be if the value for the material input is Gore-tex then the product-type-list must contain one or more values from the set {Clothing, Shoes}. As another example, another rule might be that the value entered for a Suede jacket must always be greater than that of a Denim jacket of the same size. In other words, there is some validation that needs to be performed before the request is processed and it cannot be handled in the context of the basic type system.

With current Web services infrastructure, this validation can potentially result in several wasted roundtrips to the server, increasing load on the server and frustration levels at the requestor's end. There should be a standard mechanism by which a web service can provide either extra semantics about its data model or provide validation logic that can be used by the requestor to reduce round trips to the server.

The main point to take away from this scenario is that the issue of providing responsiveness exists in today's web application domain and is being addressed in a variety of ways (which we cover in the next section of this paper) all assuming some functional capability of the client endpoint. With web services as a normalization concept, there is a need to address the problem at a level above the technologies and languages and provide the right abstractions and mappings/bindings to a widely agreed upon set of technologies and lower-level standards.

While the above scenario describes a user-facing application, it can also be mapped to automated application-to-application interactions. The Web services standards already build on XML schema to define data types. This allows the requestor end to validate requests for prior to sending them on the wire. Providing a more complete model of the data, with additional semantics such as in XFORMs is another step in that direction.

2.2 Customization

Consider the scenario where an application is interacting with a Call Center Web service, that itself uses multiple services to perform its function: a customer lookup service, a mailer service, a spell checker service, and a problem report service. By default the spell checker service used is the Webster spell checker service. When the requestor creates in a problem statement in a particular technical domain, the requestor application would like intelligent prompts and spell checks. In the case where the service composition details are completely hidden from the client, the Webster service returns a highlighted text via the composing Call Center service indicating what it thinks are incorrect words. However, if the Call Center service exposed aspects of its composition in a well-defined manner along with the defaults, it would enable the requestor application to dynamically indicate which particular spell checker service to use based on the current problem. In fact, the new spell checker might be a native application that is part of the software infrastructure available at the requestor's end that can offer better response and integrates better with the eventual application.

This scenario has some aspects that are indirectly related to efforts in the industry related to Web service composition, coordination and orchestration (BPEL4WS [3], WS-C, W3C Choreography, …) and in particular to the notion of abstract definition of web service interfaces, relationships, flow and binding them to implementations at a later time. However, this scenario has a much simpler need, the ability for a service to simply export its dependent services and default bindings in a standard manner and the ability for a client to over-ride the default bindings.

3 Related Work

There is an enormous body of work in both research and industry that is relevant to support requestor friendliness as described in this paper, some of which we describe below. Research in distributed object systems that supports mobility and disconnected operation is another source towards a generic solution. The attempt of this section is not to be complete, but the main point to take away is that there is a lot of work that we can leverage and attempt to normalize at the level of web services without any bias towards a particular set of technologies.

The problem of frequent round tripping to the server is recognized in the web application domain where programmers resort to using JavaScript to perform not only just validation but also several other functions such as sorting and simple computations that do not go back to the server. XFORMs specification [4] also addresses this problem supporting the definition of data models and corresponding constraints. An XFORMs compliant browser, on receiving an XFORM document, can perform a great deal of validation including at the level of instance values without requiring any code from the server or round trips to the server.

Caching of data on the requestor's end is a well known approach to improving responsiveness of distributed applications. Mowbray and Malveau [5] illustrate the use of such smart client stubs in their "Fine Grained Framework".

WSRP [6] proposes a way for user-facing Web services to be plugged into portals. It also allows expression of cache policies so that presentation data for the service can be stored locally on the client. Our approach advocates a similar expression of suitable policies and other metadata, going beyond the realm of presentation data alone.

The BPEL4WS specification [3] defines *abstract compositions* of Web services. It allows the actual service instances used in a composition to be configured separately, possibly at runtime. This plays directly into our requirement for a Web service to expose its dependent services in a standard manner.

4 Towards a Solution

A key challenge in proposing requestor-friendly services and designing a supporting software infrastructure is that we need to be able to leverage existing open standards, in keeping with the philosophy behind the definition of the Web services platform. So much as XML Schema has become part of the Web services vocabulary, we can leverage higher level data models such as XFORMs. Another challenge is to be able to support existing methods of accessing Web services, while at the same time taking advantage of more metadata if that is available.

The abstract approach that we are experimenting with is based on the classic MVC paradigm with a little twist. Specifically, in this approach, a Web Service conceptually consists of one or more of the following elements:

- A data model that represents the business data of the service along with all its constraints.

- A presentation model, separated from the data model since the policies associated with its use are often unique.
- Execution model that operates on the data and can provide information about constraints on interaction with the service. The execution model can be a platform independent description such as WSDL and BPEL4WS documents or may correspond to java byte code.
- A connection model or a plug-in model that specifies the services/service interfaces that the service depends on and the additional requirements imposed on larger compositions due to the presence of such dependencies.

The basic idea is that we provide a means for a Web service to export the above models and in addition what is required on the requestor to interpret/execute those models. Then a model-aware requestor in conjunction with the service can make decisions the amount of processing to be performed on the requestor versus the provider end. All this has to be done in a secure and seamless manner especially when code deployment is involved on the requestor's end.

In the first phase of this work we have begun to design a model-aware service invocation framework based on WSIF [7] and hope that it eventually results in the definition of a more requestor-friendly metadata stack for Web services..

5 Summary

We motivated the need to extend Web services vocabulary and usage patterns to better support and exploit service requestors. Several solutions are possible but we believe that it can be completely transparent to application code by encapsulating this new function within a generic requestor framework. An application is not required to use this new functionality; existing application independent proxy generators can still be used. This work will leverage similar research in web applications and traditional distributed systems, and augment Web service descriptions and policies with existing standard ways of defining data and execution constraints wherever possible.

Acknowledgements

We thank Francisco Curbera and Sanjiva Weerawarana for their comments on this paper.

References

1. CORBA (Common Object Request Broker Architecture) 3.0, published on the World Wide Web by OMG, July 2002, http://cgi.omg.org/docs/formal/02-06-33.pdf
2. Christensen, E., Curbera, F., Meredith, G. and Weerawarana., S. Web Services Description Language (WSDL) 1.1. W3C, Note 15, 2001, www.w3.org/TR/wsdl

3. Andrews, T., Curbera, F. et. Al. Business Process Execution Language for Web Services (BPEL4WS) Version 1.1, ftp://www6.software.ibm.com/software/developer/library/ws-bpel11.pdf
4. Dubinko, M., Klotz, L. et. al. XFORMS 1.0, published on the World Wide Web by W3C, http://www.w3.org/TR/xforms/
5. Mowbray, T. and Malveau, R., "CORBA Design Patterns", published by John Wiley and Sons.
6. Diaz, A., F., Peter, WS-RP (Web Services for Remote Portlets) published on the World Wide Web by IBM, January 2002, http://www-106.ibm.com/developerworks/web/library/ws-wsrp/?dwzone=web
7. Duftler, M., Mukhi, N. et. al. Web Services Invocation Framework (WSIF), OOPSLA Workshop on Object Oriented Web Services, October 2001.

Using Web services in the European Grid of Solar Observations (EGSO)

Simon Martin and Dave Pike

Space Science and Technology Department, Rutherford Appleton Laboratory, Chilton, OX11 0QX, UK
{simon.martin, c.d.pike}@rl.ac.uk

Abstract. The European Grid of Solar Observations (EGSO) [1] is employing Grid computing concepts to federate heterogeneous solar data archives into a single 'virtual' archive, allowing scientists to easily locate and retrieve particular data sets from multiple sources. EGSO will also offer facilities for the processing of data within the Grid, reducing the volume of data to be transferred to the user. In this paper, we examine the use of Web services in EGSO as a means of communicating between the various roles in the system.

1 Introduction

To understand the Sun, solar physicists need access to data from a variety of instruments scattered across the globe. Data are stored in archives with varying degrees of accessibility. Even if easily accessible via the Internet, these archives are heterogeneous, with the metadata catalogues describing the data varying widely between archives [1]; hence obtaining the desired data can often be difficult. Additionally, the volumes of data involved are very large. Current archives may have accumulated as much as 1TB of data, whilst future missions may produce this quantity of data in a day. As well as the problems associated with searching large archives, transferring vast amounts of data across networks is undesirable.

EGSO is a Grid test-bed whose main aim is to improve access to solar data. This will be achieved by federating distributed data archives, creating standardised metadata catalogues of the data available and providing users with tools to search these catalogues for specific data sets and retrieve them, whilst insulating the user from the details of data access [2]. EGSO will reduce the amount of data transfer required by providing data processing facilities (e.g. to calibrate datasets) within the Grid; hence EGSO is both a data and service Grid [3].

In this paper we briefly outline the EGSO functional architecture (section 2) and describe how Web services are being employed in the current phase of the EGSO project (section 3). Section 4 assesses the suitability of Web services for this purpose.

2 The EGSO Functional Architecture

The functional architecture for EGSO defines three separate roles [4]: consumers, providers and brokers (an organisation may play multiple roles). In simple terms, a consumer represents the user interaction with EGSO, and interacts initially with a broker to discover which provider(s) may hold the desired data (or service). The consumer then contacts the relevant provider(s) to obtain the requested data (or service). Providers are usually linked to data centres and offer data access facilities, but may offer services such as data processing. Brokers collect information from providers, such as metadata catalogues or details of their services, which can then be used by consumers to perform data searches; the system has multiple brokers which can behave as a single 'virtual broker', although this multiplicity is invisible to consumers. Typical information to be exchanged between roles includes data files, images, fragments of metadata catalogues, and details such as authentication data or session IDs.

Roles interact with each exclusively via an *external interaction subsystem*, which must support passing messages of the types listed above, whilst being loosely coupled to the rest of the role to allow possible replacement as Grid technologies mature. The subsystem contains components which allow the consumer to interact with the broker and provider, the provider to interact with the broker and consumer, and the broker to interact not only with consumers and providers, but also with other brokers.

3 Using Web services in EGSO

Web services represent a service-oriented approach to distributed computing, with services accessed via XML messaging over Internet-based protocols for platform-independence [5]. Standards [6] such as XML and SOAP ensure interoperability, whilst UDDI and WSDL allow the discovery and description of Web services. Although Grid middleware is available (e.g. the Globus toolkit [7]), at this time we have decided to use Web services for inter-role communications (i.e. in the external interaction subsystem) in EGSO for several reasons. Web services are compliant with direction of W3C and industry, they are platform independent, they are lightweight, and can be easily replaced and deployed on systems. They are also loosely coupled, and enable remote procedure call (RPC) and document exchange type Web services to be implemented, synchronously and asynchronously [8]. Furthermore, the Globus toolkit is starting to implement the Open Grid Services Architecture (OGSA) [9], which integrates Web services and Grid technologies and concepts; OGSA is not yet a mature technology, but we are then well positioned to implement it if necessary in place of Web services.

3.1 Implementation

Document exchange and RPC-type Web services were investigated to determine their suitability for use in EGSO. Sun's Java Web Services Developers Pack (JWSDP) v1.1

[10] was used to implement both types of Web services. RPC-type Web services were also developed using Apache Axis, a SOAP implementation [11].

To develop RPC-type Web services and clients, the JWSDP provides the Java API for XML-based RPC (JAX-RPC); the current Reference Implementation uses SOAP as the application protocol and HTTP as the communication protocol. The API hides much of the complexity from the developer, representing method calls and responses as SOAP messages. On the server side, the developer specifies remote procedures by defining these methods in a Java interface, and codes the relevant classes that implement those methods. On the client side, the remote method is called on a stub object, which acts as a proxy for the remote service. The JWSDP tools create any required classes (e.g. stubs) and deploy the Web service in a Web container (Tomcat).

Axis can create Web services in a similar manner, but also allows for very simple deployment of RPC-type Web services; Java classes with public methods can be exposed as Web services by simply placing them in a target directory. Clients can then be created using a simple Axis API to access the service. Both Axis and JAX-RPC also allow the use of dynamic proxies or dynamic invocation interfaces to access Web services whose WSDL descriptions are only known at runtime, although this method was not tested.

Document exchange-type Web services were developed with the JWSDP using the Java API for XML Messaging (JAXM) and SOAP with Attachments API for Java (SAAJ). JAXM provides classes and interfaces for creating a special type of servlet (JAXMServlet) which can send and receive SOAP messages, and for using messaging providers, discussed below. SAAJ is used for creating SOAP messages (with optional attachments) and sending them synchronously without using a provider. XML messages can be sent between applications with or without the use of a messaging provider. In the former case, a standalone JAXM client can run independently, or within a Web container. It sends a SOAP message synchronously over a connection to a listening JAXM servlet; this is known as request-response messaging.

Alternatively, JAXM applications can use messaging providers (they are then peers). A messaging provider is a service hidden from the developer that handles the transmission and routing of messages. Very simply, the client sends the SOAP message to its messaging provider with the details of the recipient(s) in the SOAP Header. The messaging provider then forwards the message to the servlet's provider, which then sends the SOAP message to the servlet. There are several advantages to using messaging providers including the fact that they are continuously active, and so a JAXM application can close its connections after sending a message and the provider will still send the message; the provider can also be configured to resend messages until they are successfully delivered, and will store incoming messages for the application ready for delivery upon reconnection. A significant advantage of messaging providers in the context of EGSO is the ability to send a message to multiple intermediate destinations before the message is delivered to its final recipient. The intermediate destinations, or actors, are specified in the header of the SOAP message. Providers can also incorporate profiles which are implemented on top of SOAP; these are specifications that tell providers how to route messages.

4 Assessment of Web services for use in EGSO

In developing both types of Web service, the largest barrier to progress was found to be incomplete documentation. RPC-type Web services were quite easy to deploy using JAX-RPC and Axis, with Axis being the simpler of the two. Document exchange Web services were also found to be easy to employ using JAXM; creating and sending simple SOAP messages synchronously was quite straightforward. However, the real strengths of using JAXM were found to be the ability to add attachments of any type (e.g. images, text files) to the SOAP message, and the ability to use messaging providers to not only ensure delivery of messages, but to send the message to multiple recipients.

There are several issues to be considered before committing to use Web services in such a large scale project. Security is a prime concern when using Web services e.g. [13]; issues include authentication (verifying the identity of the message sender), authorisation (determining whether the sender has permission to perform the requested operation), integrity (verifying that the message received is unmodified), and confidentiality (keeping the message private from unauthorised users).

There are also several quality of service (QoS) and performance issues which need to be addressed [14]. In terms of reliability, SOAP messages are transmitted using HTTP; hence there is no guarantee of packets being delivered to their destination. In terms of RPC and synchronous document exchange services, this is a problem as the SOAP message will need to be resent. However, this can be overcome with JAXM since messaging providers will re-send a message until it is delivered. The use of SOAP (XML) can cause performance problems, both in terms of applications parsing the XML and the fact that XML messages tend to be substantially larger than equivalent binary data, increasing bandwidth usage. Furthermore, network latency, web server/container performance under load and back end systems can also affect performance. However, given the vast amounts of data which need to be searched in order to locate a particular data set for a user, along with processing of this data, then many of these performance issues may be inconsequential.

5 Conclusions

Web services are relatively easy to develop and deploy. RPC-type Web services can easily implement simple method calls, or could be used to initiate more complex tasks through a composition of method calls. Document exchange using JAXM appears to be very well suited to use in EGSO, with its ability to send messages reliably, to multiple recipients, and to send non-XML content as attachments.

Some further investigations need to be carried out regarding issues such as scalability, optimisations, and security, although these are not barriers to implementing Web services in EGSO. Web services appear to be a viable method for communicating between the roles in EGSO, particularly in the early stages of production to allow integration testing of various components under relatively light load levels. The lightweight nature and loose coupling of Web services means that it

should be relatively easy to add new roles into EGSO, and the external interaction subsystem can also be easily replaced with Grid middleware if required.

References

1. http://www.egso.org
2. Csillaghy, A., Zarro, D. M., and Freeland, S. L.: Steps towards a virtual solar observatory. IEEE Signal Processing Magazine, N.18/2 (2001) 41-48
3. Foster, I., and Kesselman, C. (eds.): The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers (1999)
4. Piccinelli, G. (ed.): EGSO Architecture. EGSO Report EGSO-WP1-D4 (2003)
5. Vasudevan, V.: A Web Services Primer http://webservices.xml.com/pub/a/ws/2001/04/04/webservices/index.html (April 2001)
6. http://www.webservices.org/index.php/article/archive/3/
7. http://www.globus.org
8. Chappell, D. A., and Jewell, T.: Java Web Services. O'Reilly and Associates (March 2002)
9. Foster, I., Kesselman, C., Nick, J. M., and Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum (June 22, 2002)
10. http://java.sun.com/webservices/webservicespack.html
11. http://ws.apache.org/axis/
13. Deitel, H. M., Deitel, P. J., Gadzik, J. P., Lomelí, K., Santry, S. E., and Zhang, S.: Java Web Services for Experienced Programmers. Prentice Hall (2003).
14. Mani, A., and Nagarajan, A.: Understanding quality of service for Web services. IBM Developer Works, http://www-106.ibm.com/developerworks/library/ws-quality.html, (January 2002)

Agile Modeling and Design of Service-Oriented Component Architecture

Zoran Stojanovic, Ajantha Dahanayake, Henk Sol

Systems Engineering Group, Faculty of Technology, Policy and Management,
Delft University of Technology,
Jaffalaan 5, 2628 BX Delft, The Netherlands
{Z.Stojanovic, A.Dahanayake, H.G.Sol}@tbm.tudelft.nl

Abstract. Component-Based Development (CBD) and Web Services (WS) have been proposed as ways of building high quality and flexible enterprise-scale e-business solutions that fulfill business goals within a short time-to-market. However, current achievements in these areas at the level of modeling and design are much behind the technology ones. This paper presents how component-based modeling and design principles can be used as a basis for modeling a Service-Oriented Architecture (SOA). Proposed design approach is basically model-driven, but incorporates several agile development principles and practices that provide its flexibility and agility in today's ever-changing business and IT environments.

1 Introduction

During the last years first Component-Based Development (CBD) [1] and then Web Services (WS) [4] have been introduced as paradigms for building complex Web-based systems and providing effective inter- and intra-enterprise application integration. Besides technology developments, there is a need to architect component-based and service-oriented enterprise-scale software systems. Service-Oriented Architecture (SOA) is an approach to distributed computing that considers software resources as services available on the network that in collaboration provide comprehensive and flexible system solutions. CBD and WS technology platforms are naturally the ways of implementing SOA. However, developers and system architects cannot just start using technology such as EJB or .NET or standards such as XML and SOAP in realizing the SOA. Effective methods for modeling and design of such a complex architectural model are required. Among the other benefits, SOA design should provide a necessary support in deciding:

- what component of the system can be exposed as a service, that can be potentially used in intra- or inter-organization settings, offering a business value to the consumer, and at the same time being as much as possible decoupled from the rest of the system.

- what part of the system logical architecture can be realized by invoking a particular service over the Web, and how that part should interface with the existing organization's system solution.

The SOA modeling and design approach should provide a way of capturing given business requirements in the platform-independent system architecture that can be further mapped into the particular implementation solution, providing effective bi-directional traceability between business concepts and implementation assets. This is the main idea behind the current Object Management Group's (OMG) Model Driven Architecture (MDA) [5]. On the other hand, due to ever-changing business, principles and practices of another development paradigm called Agile Development (AD) must be considered as well [3]. While both AD and MDA provide solutions for building flexible solutions under the high change rates and within short time-to-market, their targets and proposed mechanisms are quite dissimilar. Therefore the balance between the two must be made in order to use the benefits of both paradigms.

The aim of the paper is to propose a service-oriented component modeling and design approach organized around the concepts of services and components in the Service-Oriented Architecture. The approach provides a paradigm shift from components as objects to components as service managers that makes component concepts capable for modeling the architecture of collaborating and coordinating loose-coupled business-valued services. The approach is flexible and agile, providing the way of balancing business and IT concerns, and adopting changes from both sides.

2 Related Work

SOA is an evolutionary, rather than revolutionary concept. A basis of SOA is the concept of a service as a functional representation of a real-world business activity meaningful to the end user and encapsulated in a software solution. Using the analogy between the concept of service and a business process, SOA provides that loosely coupled services are orchestrated into business processes that support organization's business goals. Components and services modeled in implementation-independent way represent an abstraction layer between business and technology. Business goals, rules, concepts and processes are captured by components and services at the specification level that are further mapped to technology artifacts providing effective bi-directional traceability between business and technology. The representation of the building blocks of SOA in a conceptual way provides the level of communication and understanding that is above the level of XML-based languages such as Web Services Description Language (WSDL) [8]. This is particularly important for providing common understanding and effective communication among the project stakeholders.

The natural starting points for SOA modeling and design are component-based and interface-based concepts and techniques, as well as the standard UML as a modeling notation. The current version of the UML (version 1.5) still treats components mainly as implementation units, rather than the main building blocks of the logical system architecture (although there are some improvements in that direction from the version

1.3) [6]. An improved support for components has been promised for the next major revision of the UML (version 2.0) scheduled for this year.

On the other hand, classical CBD methods do not provide thorough support for business-level concepts and services within the SOA [1]. Their focus is mainly on finer-grained components that closely map the underlying entities such as Customer, Order, and Product, rather than on larger-grained, business value added services and components as required by SOA. By treating components as binary-code packaging artifacts during implementation and deployment and as larger-grained business objects during analysis and design, these methods are not well equipped for modeling loosely coupled coarse-grained service-based components that offer business meaningful services organized in the SOA. Moreover, by defining a number of modeling artifacts as well as a complex and prescriptive way of using them proposed methods are often heavyweight and not flexible and adaptable enough to fit into agile business environments of today. A SOA modeling approach must be business service-driven rather than data-driven with strong requirements for modeling service interaction, coordination and dependencies at different levels of granularity. The collaboration and coordination of service components become as important as components themselves.

Therefore a SOA modeling and design approach should be naturally based on standard practices of component-based and object-oriented (OO) paradigms integrated with business process and workflow design concept and techniques. Business and system modeling and design are, more than ever before, integrated around the same set of service concepts and solutions.

3 Service-Based Component Concepts

Components were first introduced at the level of implementation and deployment through the component implementation models such as CORBA Components, Sun's Enterprise Java Beans, and Microsoft COM+/.NET. They have been defined as packages of binary and/or source code that can be deployed over the network nodes. Just recently components have become important analysis and design artifacts in creating logical system architecture.

On the other hand, Web services are self-contained self-describing, modular units providing location independent business or technical services that can be published, located and invoked across the Web. They are natural extension of component thinking. From a technical perspective the web service is essentially an extended and enhanced component interface constructs. Web services, as components, represent black-box functionality that can be reused without worrying about how the service is implemented.

While the component technology has been rather proprietary (divided basically into two camps - Microsoft and Java-community), Web services have provided standards and protocols for interoperability of loose-coupled software constructs across the Internet. Although these technology achievements such as XML, SOAP and UDDI are necessary for enabling true interoperability, the way of designing a system has not been changed. The basic design philosophy is still founded around compo-

nent-based design techniques such as interface-based design, black-box modeling, design patterns, design by contract, dependency modeling etc. Therefore the component design concepts are a solid foundation of an approach for designing service-oriented architecture. While the classical objects in Object-Orientation are at too low level of granularity to be considered as a basis for defining Web services, larger-grained service-based business components represent a perfect mechanism for designing services in a SOA.

For the purpose of modeling the main building blocks of SOA we introduce the concept of *service component*. A service component is a self-contained service-based building block. It delivers services to its environment through the contract-like interface that abstracts its internal realization. Services can differ in granularity (coarse or fine-grained) and nature (provides a transformation, computation or information). Component collaborates with other service components in the single application space or across the Internet to provide a higher-level goal.

The service component meta-model can be divided into two parts. First part defines the basic concepts describing the very nature of a service component. At first place a component can be defined through the three basic aspects:

- Context (environment) inside which the component exists.
- Contract that is defined according to the component role in the context and that the component guaranties to fulfill.
- Content (interior) of the component that represents a realization of the component contract.

A component does not exist in isolation; it fulfils a particular role in a given context and actively communicates with it. A component participates in a composition with other components to form a higher-level component. At the same time every component can be represented as a composition of lower-level components. A component must collaborate and coordinate its activities with other components in a composition to achieve a higher-level goal. Well-defined behavioral dependencies and the coordination in time between components are of a great importance in achieving the goal.

The second part of the component meta-model defines the basic elements of the component contract as the main aspect of a service component. Component contract concepts represent the complete information about the component necessary for its consumer to use it without knowing its interior. This is an enriched and enhanced basic interface construct that now contains all the information about the component (or service) that must be known by its context in order to make use of it. In this way a component interface goes beyond simple operations' signatures to become a real business contract between a component as a service provider and the context as a service consumer. The following are the contractual concepts of a component:

- Component identification
  - Unique name in the naming space or identifier, the goal and purpose of a component (service).
- Component behavior
  - Operations (actions) provided and required,

  - Pre-conditions and post-conditions defined on these operations,
  - Events published and subscribed,
  - Coordination of operations and/or events to provide a higher-level behavior.
- Component information
  - Information types that the component uses or handles (not necessarily stores) mostly as parameters for services and operations the component provides and requires,
  - Invariants and constraints on these information objects.
- Configuration parameters
  - Parameters defined by the component that can adapt its contract to fit into possibly new requirements coming from the context, such as required Quality of Service (QoS), location in space, location in time, consumer profiles, etc.
- Non-functional parameters
  - Parameters that characterize the "quality" of component behavior in the context, such as performance, reliability, fault tolerance, priority, security etc.

The component contract can be fully specified using different mechanisms, from natural language to formal specification language and to XML-based language if we want a machine-readable specification of a component. On the other hand the component contract can be implemented using different implementation tools and techniques to provide the life of the component in the world of bits.

4 SOA Modeling Approach

Complexity of distributed enterprise systems raises the need for using the separation of concerns in specifying system architecture. Therefore, we use as underlying frameworks both OMG's MDA and ISO standard Reference Model of Open Distributed Processing (RM-ODP) [7] for defining the three architectural models that represent logical layers of our service-oriented component architecture:

- Business Architecture Model (BAM) – a model of the system as collaboration of components and services that offer business value.
- Application Architecture Model (AAM) – a model of the system that shows how business components and services are realized by the collaboration of finer-grained components and services.
- Implementation Architecture Model (IAM) – a model of the system that shows how business and application components and services can be realized using a particular implementation platform.

The BAM roughly corresponds to ODP Enterprise Viewpoint, AAM to ODP Computational Viewpoint, and IAM to Technology Viewpoint. Distribution concerns in the ODP described by the Engineering Viewpoint, and information semantics and dynamics in the ODP described by the Information Viewpoint are not treated separately in our application framework then integrated throughout all three architectural models. Thus distribution can be considered as business components distribution (virtual enterprises, legacy assets, web services), application distribution (logical

distribution tiers) and implementation distribution (support by the particular middleware). Similar to this, a conceptual information model is defined in the BAM, a specification information model is fully specified in the AAM, and the ways of permanent data storage are considered in the IAM. The Figure 5 shows our architectural modeling framework.

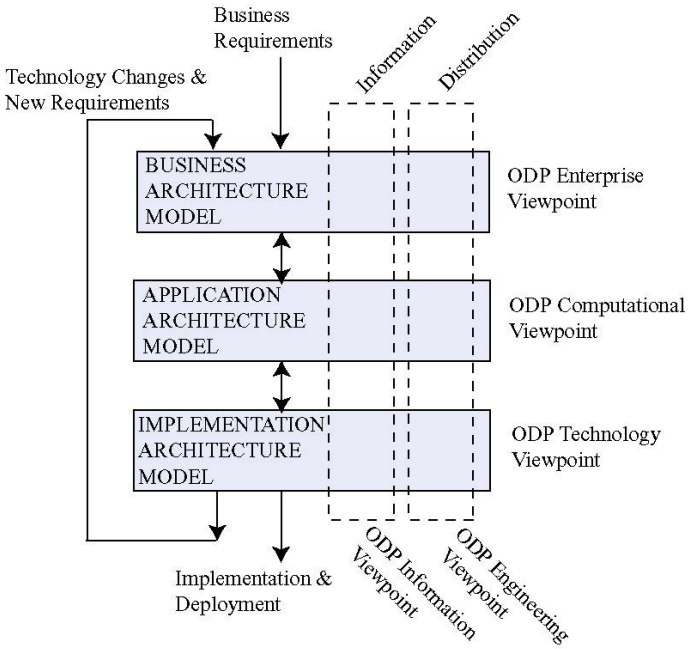

Fig. 1. Architecture Modeling Framework

The BAM and AAM actually represents two levels of abstraction of a service-oriented MDA's Platform-Independent Model (PIM), while the IAM describes a service-oriented Platform Specific Model (PSM) for a particular technology platform. By focusing on two basic component stereotypes – Business Service Component and Application Service Component, we can define two levels of a Platform Independent Model. The first PIM level defines how business process is supported through contractual collaboration and coordination of service-based business components. The second level "opens" black-box business components and defines how their interior design is realized through collaboration and coordination of finer-grained application components and services. By defining all three models in a consistent manner, the whole system is specified and ready for implementation. The best result is achieved using constant iteration and small increments during design, as suggested by agile development principles.

The main goal of the BAM is to specify the behavior of the system in the context of the business for which it is implemented in terms of collaborating and coordinating chunks of business functionality represented as business service components. BAM starts with the following models: activity model that shows the flow of activities in

the system, use case model and domain information object model. Based on use cases that fulfill business user goals (i.e. that correspond to Elementary Business Processes [2]) we define business services that system should provide, as well domain information objects used by these services. For each use case (and a service that supports it) the use cases that precede it, follow it, perform in parallel with it or be synchronized in other way with it should be defined, Figure 2. Furthermore, for each use case its superordinate and subordinate use cases should be defined providing a hierarchy of use cases, i.e. business goals. This can be illustrated using an activity diagram with use cases as action states of the diagram, or a sequence diagram enriched to express the action semantics (sequence, selection, loop, fork/join, etc.) with the use cases on the horizontal axis of the diagram. Domain information types are cross-referenced with the use cases defining, for each use case, what information types are needed for its performance.

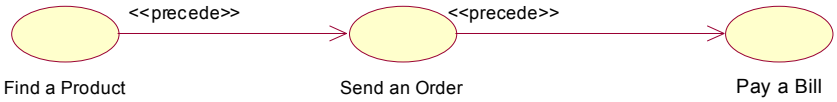

Fig. 2. The example of the relation <<precede>> between use cases

Services that support given use cases can be specified in two ways:

- in an agile-like manner using Service-Responsibility-Coordination (SRC) cards, Figure 3, as a variant of a CRC (Class-Responsibility-Collaborator) cards,
- by using more formal specification mechanisms derived from the use case specification template [2].

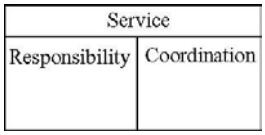

Fig. 3. Service-Responsibility-Coordination (SRC) Card

Main elements of the SRC card are:

- Service – its name reflecting its goal, purpose and scope.
- Responsibility – description of its behavior preferably through lower-level services or activity steps it provides together with information objects that should be used by the service as some kind of parameters.
- Coordination – what services (events) precede or trigger this one, what services should follow this one, or what events should be emitted; furthermore what are eventual subordinate services and a superordinate service of this one.

By using the set of different business and technical criteria, such as semantic cohesiveness, shared data objects, market value, reusability potential, existing assets, etc.,

identified services are allocated to Service Cluster Units, which represent blueprints for business service components of the system. Again, business service components can be specified in a more formal contract-based manner, or, if the nature of the project suggests, in more agile way using Component-Responsibility-Collaborator-Coordination (CRCC) cards as another variant of classical CRC cards. Collaboration and coordination of business service components that form the system can be represented using the component stereotype of the sequence diagram enriched to express control flow mechanisms. Information about that is derived from the relationships among use cases that particular business service components support. The relations among the concepts of business components, services and business goal-oriented use cases are shown in Figure 4.

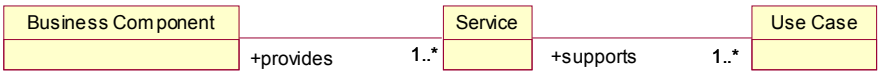

Fig. 4. Conceptual relations between business components, services (operations) and use cases

The goal of the AAM is to define how interior of business service components is realized in terms of collaboration of lower-level application components and services that do not provide a direct business value. There can be different types of application service components:

- Services that communicate with the business service component consumers by transferring their requests to the form understandable to the business logic and back. They should hide potentially different service consumers from business service logic.
- Services that provide some computation or data transformation logic;
- Services that represent contact points for information about business entities used by the given business component.
- Data access and data handler service components that hide variety of data storage formats from the business service logic.
- Service components that support included and extended use cases of the use case(s) realized by the given business service component;
- Coordination manager that coordinates other application service components inside the business component;
- Event manager component that manages the event subscription and notification mechanisms in an event-driven environment;
- Business rule manager component that handles business rules captured by the given business service component and maps these rules to pre-conditions, post-conditions, invariants, coordination conditions and other constraints defined on the component behavior and structure.

The result of the AAM is complete, fully specified, component-oriented platform-independent model that should be further considered for implementation on a particular technology platform. Functionality offered by a Business Component can be exposed as both inter- and intra-enterprise Web service in a SOA. On the other the

services offered by an Application Component can be also used as Web services, but mainly internally to the enterprise.

IAM uses the complete business-driven component-based distributed system architecture specified through the previous models, and translates them to platform-specific models according to the chosen target implementation platform. To provide further flexibility of the architecture models we propose a technique called component refactoring which aims at reallocating and rearranging sub-components or sub-services of the component being addressed, while preserving its contractual behavior, analog to code refactoring used in agile development [3]. Application components are normally implemented using implementation components, language objects/classes or other programming constructs. Application Components can be directly or indirectly instantiated (addressable) depending on their granularity. On the other hand, Business Components are implemented as a composition of software constructs that realize their sub-components (in which case they are indirectly instantiated), or can be used as already built third-party software units, such as wrapped legacy assets, Web services or COTS components (in which case they are directly addressable in a general sense).

5 Conclusion

The SOA modeling arises certain requirements on top of the standard OO and CBD modeling methods. Therefore, straightforward applying of existing UML and CBD concepts for the purpose of modeling the SOA, although a good starting point, is not a feasible approach. The UML component concept as a natural basis for SOA modeling is still mainly implementation-related, while popular CBD methods are mainly focused on finer-grained entity-driven components. Due to the business-driven character of SOA, a proper modeling approach should combine component-based and object-oriented (OO) modeling concepts on one side with activity and workflow modeling mechanisms on the other side.

This paper presents a business-driven agile approach for modeling component- and service-oriented architecture. The approach provides a paradigm shift from components as objects to components as service managers. In this way the approach is capable of modeling the system architecture representing a contract-based collaboration and coordination of components and services. Since components and services are identified based on business requirements, goals and rules, then fully specified inside the logical system architecture and implemented using advanced CBD and WS technology, the approach provides bi-directional traceability between business concepts and implementation artifacts. The approach is basically model-driven but incorporates certain agile development concepts, principles and practice (e.g. cards, refactoring, user involvement) making an effective combination between the two in order to achieve the goals of adaptable process and solution, high-quality and on-time development products that closely reflect business goals and needs. The approach makes use of standards OMG MDA and RM-ODP to provide iterative and incremental architectural modeling and design through different architecture abstraction levels pro-

viding complete specification of the system ready for implementation in chosen platform.

References

1. D'Souza, D.F. and Wills, A.C.: Objects, Components, and Frameworks with UML: the Catalysis Approach. Addison-Wesley, (1999)
2. Cockburn, A.: Writing Effective Use Cases. Addison-Wesley (2001)
3. Cockburn, A.: Agile Software Development. Addison-Wesley, Boston MA (2002)
4. IBM Web Services, http://www.ibm/com/webservices.
5. OMG Object Management Group, MDA- Model Driven Architecture, information available at http://www.omg.org/mda/
6. OMG Object Management Group, UML- Unified Modeling Langauge, information available at http://www.omg.org/uml/
7. ODP, International Standard Organization (ISO), Reference model of Open Distributed Processing: Overview, Foundation, Architecture and Architecture semantics, ISO/IEC JTC1/SC07, 10746-1 to 4, ITU-T Recommendations X.901 to 904, 1996.
8. W3C. World-Wide-Web Consortium, WSDL (Web Services Description Language). Available: http://www.w3.org/TR/wsdl

A Web Services based system for data grid

Irene Pompili[1], Claudio Zunino[1], Andrea Sanna[1], and Giacomo Piccinelli[2]

[1] Department of Computer Science
Politecnico di Torino
C.so Duca Abruzzi, Torino I-10129, Italy
irenella@tin.it,
{claudio.zunino,andrea.sanna}@polito.it

[2] Department of Computer Science
University College London
Gower Street, London WC1E 6BT, UK
g.piccinelli@cs.ucl.ac.uk

Abstract. In this paper an architecture for a data brokerage service will be proposed. The brokerage service is a part of the system that is being implemented within the European Grid for Solar Observations (EGSO) to provide a high-performance infrastructure for solar applications. A broker interacts with providers and consumers in order to build a profile of both parties. In particular, the broker interacts with providers in order to gather information on the data potentially available to consumers, and with the consumers in order to identify the set of providers that are most likely to satisfy specific data needs.

Introduction

Nowadays, Web Services are especially known as a way to improve business systems; in this paper, they will be exploited for the implementation of data grid services. The main aim of the proposed brokerage architecture is to collect information from providers and allow users to search data over a grid. For this purpose, the broker receives from providers a meta-catalogue that contains coarse granularity information. Brokers also act as access points for EGSO and allow data searches in the grid.

Brokers offer Web Services interfaces to consumers (i.c. the users) and data providers. In particular, brokers supply a mechanism to allow consumers to perform data searches, select the providers that can satisfy a specific request and forward the query. Finally, brokers collect query results and send them back to the consumer.

The content of this paper is organized as follows: in Section Background a brief background on Web Services is presented; while Section Framework describes in detail the proposed architecture. Conclusive remarks and future work are presented in Section Conclusion.

Background

Over the past few years, applications have interacted using ad hoc approaches. At the present moment, Web Services[1][2] are emerging as a framework for application-application interaction, based on existing Web protocols and based on open XML standards.
Web Services are essentially rely upon three technologies: Web Services Description Language (WSDL)[3]; Universal Description, Discovery and Integration (UDDI)[4]; and Simple Object Access Protocol (SOAP)[5].
WSDL is a specific XML format that can be used to describe Web Services interfaces. A WSDL specification provides a description of the service and the specific protocol that users have to follow to access the service itself.
On the contrary, UDDI is an industry-standard centralized directory service that can be used to advertise and locate Web services. UDDI allows users to search for Web services using various search criteria, including company name, category, and type of Web service.
Finally, SOAP is a protocol for exchanging XML data and provides the basic mechanism for Web Services communication. It can uses a textual format, as opposed to binary formats such as in CORBA[6] or Java RMI[7].
Various examples of Web services-based architectures can be found in literature. For instance, in [8] a Web Services-based system was proposed to integrate ad-hoc mobile applications with the Bluetooth and Wi-Fi technologies.
Web Services have also been used in [9] for the implementation of a biomedical portal. The proposed architecture consists of a grid portal for the management of biomedical images in a distributed environment.

The framework

The core of the proposed architecture is constituted by the broker, which offers a set of Web Services to both consumers and providers. The main broker interfaces are the connection, the provider data update and the data search interface. Core component of the proposed system is a two layer search engine. Each provider has a catalogue and periodically sends updates to a broker. The broker receives them and generates a version of this information which is inserted in this in the local database. This summarized catalogue (referred to as meta-catalogue) is obtained by using a set of parameters (e-y time, wavelength, spatial coordinate positions), in order to collapse sets of rows in the original catalogue to a limited number of rows in the meta-catalogue. For instance, if in the original provider catalogue ten rows are used to describe data related to a specific day, and the granularity of the time parameter is set to one day, then, the meta-catalogue will contain just one row. Therefore, the meta-catalogue allows the broker to immediately discard providers that certainly need not to be searched for a resource given, but cannot tell the broker if a provider actually posses a given resource. For instance, if a consumer searches data related to a specific hour, and the time parameter is used to generate the meta-catalogue with granularity equal to one day, then the broker by querying the meta-catalogue, can immediately

identify which providers can potentially satisfy the query; however, the selected providers will still need to be directly interrogated. In order to perform searches efficiently the meta-catalogue on the broker has to be constantly updated.
Moreover, after receiving an update from a provider, brokers have to propagate this information to other brokers in the network. To this aim, the following procedure is used:

Procedure propagate_updates	
B1 à B2 (all neighboring brokers, duplicates are discarded)	updates in XML format + time_stamp

The time stamp is used in order to allow brokers to select only new records.
In Fig. 1 a search session is shown. The network is composed by the broker, a consumer and three providers. The session starts when a consumer submits a query. The search session consists of the following phases:

1) The consumer submits a query to the broker.
2) The broker search engine queries the local database to obtain the list of potential providers that can answer the consumer's query. The local database contains the summarized version of catalogue (i.e. the meta-catalogue).

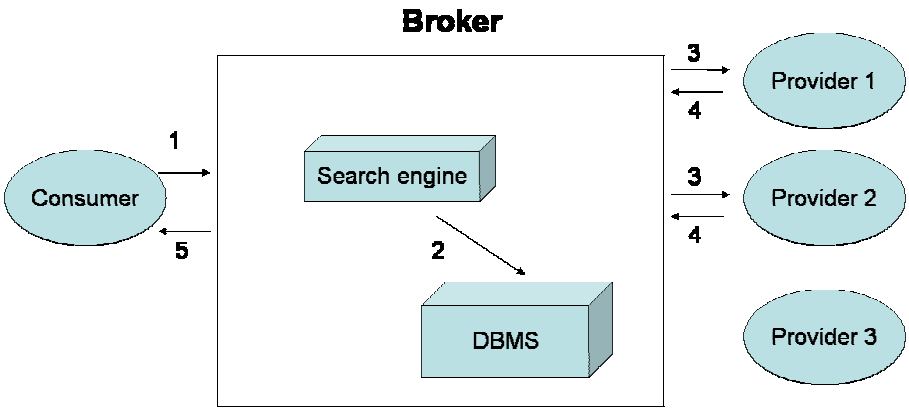

Fig. 2. A typical search session.

3) The broker forwards the query to all the providers that may satisfy the query (in this case the Provider 3 is excluded).
4) Each provider sends back results to the broker.
5) The broker collects and sends to the consumer the results obtained by each provider.

It has to be noticed that each consumer query is managed by a specific thread; in this way, the consumer, is not blocked. Only after that all results have been collected, the broker sends them to the consumer.
Finally, the consumer directly contacts the provider to retrieve the data.

The interface of the query procedure is the following:

Procedure query	
C à B	Query
B à C	The broker returns a query_ID
B à C	The broker sends to the consumer, in asynchronous mode, query results in XML

Conclusion and future work

In this paper a brokerage architecture dedicated to information retrieval and metadata management is proposed. Metadata mainly consist of catalogues of solar data that are produced and maintained by various providers. A local database in the broker allows a faster search.
The proposed architecture can be possibly improved by using a distributed version of the broker meta-catalogue. In the case of a distributed meta-catalogue, the propagate_updates procedure still needs to be used since the information has to be replicated on a certain number of brokers selected by an ad-hoc algorithm to ensure fault-tolerance. Moreover, a new procedure to propagate consumer queries to other brokers has to be included. In fact, a broker is no longer able to immediately identify the providers that can possess requested data if the meta-catalogue is distributed among brokers. A broker will thus propagate queries to neighbouring brokers to receive information about the meta-catalogue. The broker that starts will receive results from the other brokers in XML and merge them in order to select a set of providers to be interrogated, as in the current architecture.

References

[1] Curbera, F., Duftler, M., Khalaf, R., Mukhi, N., Nagy, W., Weerawarana, S. “Unraveling the Web Services Web - An Introduction to SOAP, WSDL, and UDDI”, IEEE Internet Computing, Vol.6 Issue 2, March-April 2002, pp.86-93.

[2] Roy, J. Ramanujan, A. “Understanding Web services”, IT Professional,Vol. 3 Issue 6, Nov/Dec 2001, pp. 69-73.

[3] http://www.w3.org/TR/wsdl.html

[4] http://www.uddi.org/

[5] http://www.w3.org/TR/SOAP/

[6] http://www.corba.org/

[7] http://java.sun.com/products/jdk/rmi/

[8] Steele, R., “A web services-based system for ad-hoc mobile application integration”, Proceedings. ITCC 2003. International Conference on , April 28-30, 2003, Page(s): 248-252

[9] Aloisio, G.; Blasi, E.; Cafaro, M.; Fiore, S.; Lezzi, D.; Mirto, M., “Web services for biomedical imaging portal”, Proceedings. ITCC 2003. International Conference on , April 28-30, 2003 Page(s): 432 -436