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.

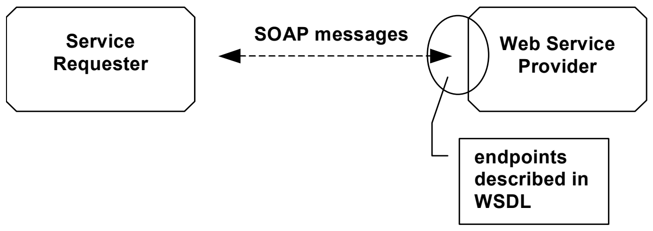

Fig. 1. Service requester and a web service provider interact through SOAP messages

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.

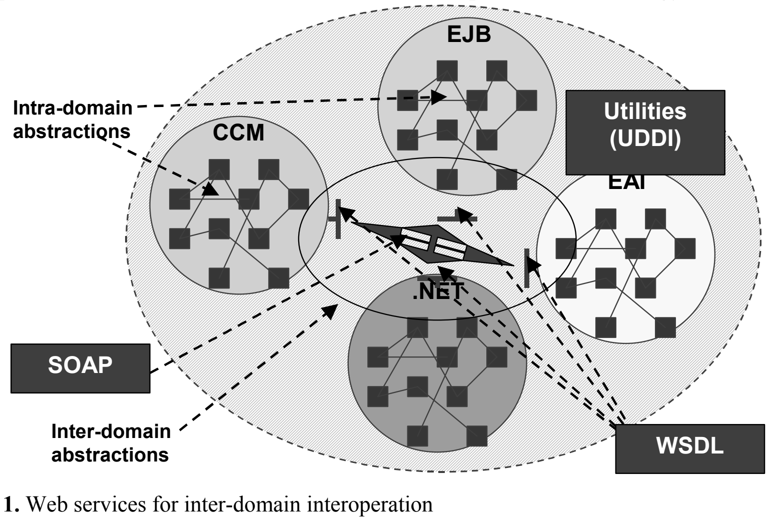

Fig. 1. Web services for inter-domain interoperation

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/