Collaborative Web Service Technologies

Wolfgang Schreiner, Schahram Dustdar
Distributed Systems Group, Information Systems Institute
Vienna University of Technologies
{schreiner, dustdar}@infosys.tuwien.ac.at

Abstract

Due to their heterogeneous nature, which stems from the definition of several XML-based standards to overcome platform and language dependence, Web services have become an emerging and promising technology to design and build complex inter-enterprise business applications out of single Web-based software components. The aim of this paper is to discuss the basic idea of service oriented architectures and the service oriented programming paradigm, their appliance in collaborative environments and the challenges that may arise when performing service based collaborative systems engineering. We present current approaches to overcome these difficulties, but focus on the many unsolved issues of the current state of the art in Web service technologies.

1. Introduction

Web services, as understood today, are built on the implementation of several XML based standards for consistent software component interface description, component registration and message exchange among interacting service components. Since many different standards and architectures have been proposed in the past years to enable distributed computing, Web service technologies provide an innovative approach to overcome the drawbacks of other existing mechanisms. In order to be able to develop distributed software components to increase QoS aspects, such as reliability or availability, or to enable collaboration and interaction among different users, companies have developed their own protocols, which lead to the implementation of adapters to overcome heterogeneity and to enable interoperability among different technologies. Considering n incompatible message formats would require $n*(n-1)/2$ adapters to make them work together and n additional adapters for each new technology that is being developed. Web services eliminate these drawbacks, by introducing standards for uniform software component description and message exchange. If there is only one way to exchange messages, this reduces the number of required adapters to n, implementing mappings from n different message formats to one standardized XML protocol.

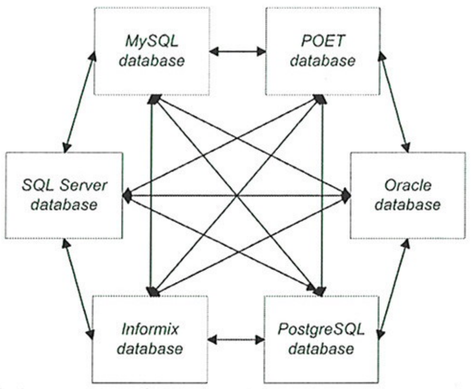

Fig. 1: Data transfer between heterogeneous data sources

Figure 1 illustrates the interactions in a heterogeneous network. Since every endpoint supports different communication protocols, mapping rules for each message format to the other ones are needed,

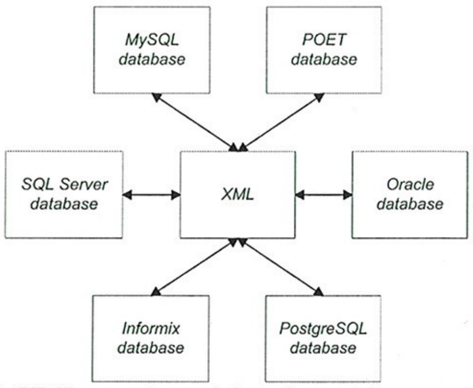

Fig. 2: XML as universal data format for message exchange

Figure 2 shows the benefits of a uniform XML protocol. Data transfer works via XML formatted plain text, which is parsed and transferred into platform specific message format at each communication endpoint every time a message is about to be sent or received.

With Web services it does no longer matter, which programming language a software component is implemented in or which operating system hosts a Web service. The interface of a software component, including namespace or operation signatures, is described in a standardized XML format called WSDL (Web Service Description Language) [1]. Remote procedure calls (RPCs) to a software component are also translated into a standardized XML format, namely SOAP (Simple Object Access Protocol). SOAP [2] allows different protocol bindings, such as HTTP or SMTP, for message transfer. Considering other technologies, RMI for instance, requires Java installed at each communication endpoint, since messages are sent as Java byte code. A host exposing a Java component as a Web service first translates the byte code, originating from a Java remote method invocation, into a SOAP-formatted message before sending the request over the network and accepts only SOAP messages respectively, which are then translated to a message format, specific for the certain programming language used for the underlying service implementation.

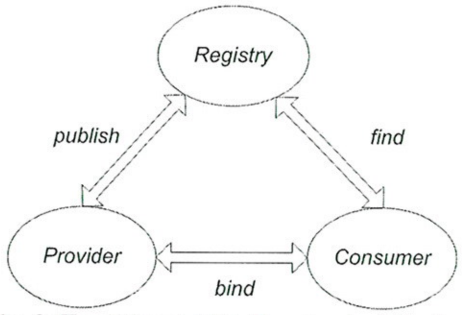

Fig. 3: Traditional Web Service Architecture

Figure 3 illustrates the traditional Web service model including the three entities Service Registry, Service Provider, and Service Consumer. This model is widely adopted and implies kind of client-server architecture. Other approaches, such as P2P infrastructure, where each node acts as server as well, have been suggested. The service provider entity implements or simply offers a certain service, which in this context refers to a software component consisting of several methods or procedures written in any programming language running on any operating system. The component's interface, including name, access point, signatures of visible operations and the protocol to be used for SOAP messaging is translated into WSDL format and published in a global service registry. The information stored in the registry is also XML formatted using the UDDI (Universal Description, Discovery and Integration) [3] standard, for instance, and allows the storage of additional information about the service provider, such as name, address or telephone number, and technical information about the service itself. The service consumer may then contact the registry by sending a SOAP formatted request, and search for services that have been published. If an appropriate service is found, the service consumer may invoke the service via SOAP using the service URL or access point stored in the registry. The corresponding basic operations are also depicted in Figure 3. Both SOAP and WSDL have been developed by the W3C, while UDDI is an OASIS release. In order to decouple the SOAP and WSDL standards from each other, IBM proposed the Web Services Invocation Framework (WSIF), which is now hosted by the Apache Software Foundation. WSIF hides the protocol binding through which the WSDL description is being invoked and allowing the service provider concentrate on the service interface definition.

So far, we introduced the basic Web service concepts. The standards named in the introduction have already been implemented and basically work for Web service implementation and usage. Meanwhile WSDL 2.0 and SOAP 1.2 have been proposed and UDDI already specifies version 3. Unfortunately, there are still many open problems concerning Web services to enable serious enterprise application integration. When service interactions become more complex, additional standards are needed to ensure consistency and coordination of the service invocations. Adding transactions to simple RPCs requires additional protocols, building complex applications requires composition support for Web services and distributed message transfer among multiple users requires certain security mechanisms, just to name a few unsolved issues. In Section 2, we will focus on service coordination protocols and transactions. Section 3 concentrates on the relevance of composition of software components in a Web service environment. We deal with issues influencing service composition strategies and briefly discuss currently existing composition concepts.

2. Service Coordination

Applications used in medium to large enterprises are often very complex, since they may map intra- and inter-organizational business processes of a company. This requires a huge amount of software components being linked together and coordinated. With current Web service technologies it is possible to overcome the issue of platform heterogeneity, but they do not solve any implications from both the internal or external perspective [4]. From the internal implementation perspective, a client must be able to execute different operations in an appropriate order and eventually maintain status and context information during runtime. From the external interaction perspective, Web services should provide constraints to allow clients to invoke its methods only in a certain order. These implications are referred to as conversation, dealing with the sequence of operations occurring between a client and a Web service. The set of accepted conversations is what specifies a conversation protocol, for example 2PC (two phase commit) to ensure ACID properties during a transaction. A coordination type, in turn, defines a set of coordination protocols. In order to provide conversation modeling for Web services, different standardization efforts have been taken, such as WS-Coordination proposed by IBM, Microsoft and BEA and WS-CF [15] as part of the WS-CAF [13] framework by Sun.

The goal of WS-Coordination was to provide a framework for supporting coordination protocols [6]. This has been achieved by standardizing the exchange of a unique identifier among interacting services, a registration interface for services to register their communication ports and an activation interface to assign roles to each involved service. The basic entities in WS-Coordination are coordinators and participants. WS-Coordination supports centralized, letting all participants in a conversation interact with the same coordinator, and distributed, with one coordinator for each service involved, coordination as well. Based on the standardization efforts named above, WS-Coordination defines three forms of interactions, called Activation, Registration, and Protocol-specific interactions. Activation creates a new coordination context each time a participant initiates a new conversation. Registration contains references to Web services taking part in a conversation waiting for notification in case of events in the coordination protocol. Protocol-specific interactions are used for protocol specific message exchange between coordinators and participants. Activation and registration are independent from the underlying coordination protocol. WS-Coordination defines SOAP extensions, meta-protocols and middleware components to enable coordination between XML Web services.

One of the main conversation protocols needed is transaction support. The WS-Transaction specification suggested by IBM, Microsoft and BEA has been replaced by two other proposals, namely WS-AtomicTransaction [7] and WS-BusinessActivity [8]. In 2003, Sun published a standard proposal as part of their WS-CAF framework, called WS-TXM [16]. WS-AtomicTransaction and WS-BusinessActivity respectively, build on the WS-Coordination specification and provide a common specification for transaction support among interacting Web services. Since transactions in Web services are often coupled with a long running business process, it is not always possible to ensure ACID properties for Web service transactions. Locking resources, for instance, over a time span of several hours is not always practicable. Transaction definitions for database interactions can thus not always be applied to transactions in Web services, which lead to a more flexible definition of what ACID properties mean in context with Web services. WS-BusinessActivitiy specifies long running and WS-AtomicTransaction is used for short duration transactions. Both, WS-BusinessActivity and WS-AtomicTransaction as part of the WS-Coordination specification assume the existence of coordinators and participating Web services and take advantage of the WS-Coordination coordination context to initiate and perform a transaction. Coordinators and participants as well handle the coordination protocols composed for the atomic transaction and business activity coordination types. Sun's WS-TXM specification is related to WS-CF and WS-CTX [14], defining the WS-CAF framework, and contains three distinct protocols for interoperability among different transaction managers. It supports different transaction types including 2PC, long running business activities and atomic transaction.

Other standards related to Web service coordination protocols are the XML Common Business Library (xCBL) [19], the Electronic Business Using XML (ebXML) [20] and the Electronic Business Service Oriented Architecture (ebSOA) [21] building on ebXML. xCBL. xCBL is a XML standard for B2B transactions used for eCommerce applications for example. ebXML is closely related to xCBL and like UDDI defines a business registry, but with far more storage capabilities concerning detailed service information.

The Web Service Choreography Interface (WSCI) [10] has been introduced by Intalio, SAP, BEA and Sun and contains a language definition for coordination protocols, but does not focus any lower level issues, such as service description or messaging. WSDL, for instance, could be extended with constraints about the execution order of service operations. Furthermore, WSCI allows the definition of exception handling mechanisms to specify workflow processes in case of an exceptional service behavior, transactions, including compensation and rollback mechanisms, correlators and time constraints, specifying the time interval that should elapse between two operation invocations [4]. Correlators are capable of identifying unique data items in a conversation and are thus able to associate messages to conversations, where WS-Coordination uses conversation identifiers. WSCI is an approach for protocol description. The Web Services Choreography Description Language (WS-CDL) [13] is an XML-based language that describes P2P collaborations by defining their common and complementary observable behavior. WS-CDL supports the Web service choreography specification in composing interoperable, P2P collaborations independent of programming language or platform used at the host environments.

Similar to WSCI is the Web Services Conversation Language (WSCL) [18]. WSCL allows the definition of abstract Web service interfaces, specifies the XML documents being exchanged and the allowed sequencing. As any other Web service standard, WSCL themselves consists of XML documents and can be used in conjunction with WSDL to specify abstract interfaces or binding information.

3. Composition Issues

Complex software systems consist of a huge number of software components and modules or classes in object oriented terms. In order to build large and powerful distributed applications out of Web services, we need to consider the task of service composition. Composite or aggregated services may recursively contain elementary services and other composite services. Service composition attempts to combine the functionality of many different Web services by progressively aggregating components at higher levels of abstraction.

In [5] we identified several issues that have an important impact on performing service composition. In the previous section we discussed service coordination protocols, including Web service transaction and conversation modeling referring to currently proposed standards. When dealing with service coordination aspects, the term context plays a highly crucial role, although a wide variety of different definitions exist in literature. In terms of Web services, context is often understood as information used by a Web service to adjust execution and output to provide the client with a customized and personalized behavior. Context may refer to information about client location, hard- and software used on the client device, network and protocol data or preference settings. In Section 2 we briefly mentioned WS-CTX by Sun in this context.

When considering service composition it is also necessary to think about composite service execution management and monitoring. Composite service execution flow may be scheduled centralized or distributed via several interacting service execution monitors. Furthermore, Web services may be provided in architectures different from the traditional Web service model introduced in Section 1, for example, in a P2P manner. Thus service infrastructure is another aspect that affects the service composition process. Since Web services not just introduce a new communication pattern on an XML basis but may revolutionize the way in which software is being designed and implemented comparable to the object oriented paradigm, Web service technologies enjoy an immense research effort and investments in order to facilitate Web service composition. In literature a variety of different approaches are documented that should facilitate the service composition process.

4. Composition Strategies

Static service composition deals with composition issues during design time of service based software. Software is being designed by identifying necessary modules, their functionalities and their interdependencies. Static service composition is rather trivial since it depicts the traditional software development process. Modifications on the software itself that may occur in case of changing user requirements have to be performed on the design level. The code has to be adapted, recompiled and redeployed. Although this does not imply any further complications besides any usual, well-known, problems that arise during the design decision making phase, changes that occur in the Web service environment during run time, require increasingly costly adaptation efforts in order to make the software executable. The reaction time to any anticipated or unanticipated events is very high, making the static composition approach truly inflexible. Hence, it is much more interesting to deal with dynamic service composition to enable software reacting autonomously in case of exceptions occurring in the execution environment. Apart from changes that may occur, it would be nice if there was a way to describe Web service functionality in a way that a service interface could be discovered and bound by other services or clients dynamically. Of course, service discovery plays an enormously important role when speaking of service composition. Obviously, when investigating currently existing Web service registry technologies, such as UDDI or ebXML, data structures used for storing the service information are insufficient, since they lack of functional description and QoS information. By now it is still an unsolved problem to describe software components that they could be discovered and used regarding their functionality. Several attempts have been made to enhance existing Web service standards with additional information to facilitate service discovery, including WSDL add-ons or storing supplementary data in service registries. To make dynamic service composition work, this would require the development of a system that is intelligent enough to decompose user requirements into machine-readable commands, compare them to Web service descriptions based on the desired features, such as QoS or functionality as previously mentioned, and is able to make autonomous decisions on whether a service suits the user's needs and can be used or not.

Model driven service composition is supposed to support the dynamic composition process using UML as high level abstraction to enable the direct mapping to other standards, such as the Business Process Execution Language for Web Services (BPEL4WS) [9]. BPEL enables the composition of Web services to realize business processes. Because of the close relation to workflow management, composition elements are seen as equivalent to the business process elements activity, representing a well-defined business function, condition, constraining the behavior of the composition by adding pre- and post-conditions, event, describing occurrences during the process of service composition, flow, defining a block of activities, message, containing input- or output information, provider, describing a party offering concrete services, and the abstract class role, describing a party participating in the service composition process. Rules can be modelled by the Object Constraint Language (OCL) constraints and can be structured into structural rules guide the process of structuring, scheduling and prioritizing within service composition, data rules control the use of data and message relations, behavioral rules to control event occurrences and enforce integrity constraints, resource rules to guide the use of resources, such as service selection, providers and event raisers, and, finally, exception rules to guide exceptional behavior in the service composition process. Similar to model driven composition is the business rule driven approach, which defines four components mapping the single phases of the service composition process: definition, scheduling, construction and execution. An individual rule model assists the service developer by introducing a classification scheme for business rules: structure related rules facilitate the specification of the way in which service composition is to be carried out, role related rules govern the participants involved in the service composition process, message related rules regulate the use of information, event related rules govern the behavior of service composition in reaction to expected and unexpected events and constraint related rules represent conditions in service composition.

Declarative composition translates client requests into declarative expressions using formal languages. The architecture is somewhat different from other composition platforms, since services are typically created on the fly to satisfy client requests. The request is taken as the initial situation from which a desired goal and generic plans to reach the goal are created. One generic plan is chosen and translated into a workflow definition using planning languages. After all existing modeling languages, such as BPEL, can be applied to realize the workflow. The unique architecture of declarative composition platforms demands a revision of the aims of basic Web service concepts, including UDDI as representative of the service registry entity or WSDL lacking of a description of Web service attributes and functionalities.

Static, model driven composition as it can be performed with BPEL can be seen as a manual composition strategy, since it is done by the service developers during design time. The final compilation and deployment can be automated by delegating the translation task from the model to executable service components to the machine. In contrast to manual composition, automated composition is an ontology-based approach. An ontology, in terms of Web service technology, is a collection of services that share the same domain of interest, which means that an online payment service may belong to an eCommerce domain, for example. Automated service composition is closely related to issues addressed by the Semantic Web community. Semantic description languages, such as the Web Ontology Language (OWL), DAML-S (DARPA Agent Markup Language for Web Services) or DAML+OIL including RDF schemas are quite suitable for organizing Web services into ontologies. DAML-S divides the service description into three layers, which are defined by profiles, providing a high level description of the service, models describing the execution flow of the service, and groundings providing a mapping between DAML-S and WSDL in order to describe how the service has to be invoked. The profile should contain enough information to actually enable dynamic Web service discovery, including functional descriptions of the service, such as input or output parameters. The description part provides human understandable information, such as service name and some textual description. The concept of monitors and notifiers introduces a notion of evolution management, in case service definitions change, new services appear, or services become unavailable. The Web Services Modeling Ontology (WSMO) working group [23] aims at developing a language called Web Service Modeling Language (WSML) that formalizes the WSMO.

Transaction based service composition, as the name suggests, focuses on the transaction issue of Web service composition. Service aggregation is performed on a level deciding whether service operations support the same transactional behavior or not. Certainly, transaction semantics cannot be the only relevant feature for service composition, since functional adequacy is still the top priority decision criteria to make the software do what it is supposed to do. At the beginning of this section we mentioned several issues relevant for service composition and they all complement one another but do not substitute.

To close this section, the concept of context based service composition may be introduced as another significant composition strategy. Recall that we already discussed context as information that can be used to provide a client with customized execution settings. For that reason it is necessary to know about the client hardware and software, since the appropriate protocols need top be chosen for message transfer and depending on the interaction type, different QoS aspects have to be prioritized. If, for instance, a Web service has been exposed with the intention to offer real time data, then enough bandwidth has to be allocated between the service and the invoking clients. For QoS assurance many different aspects have to be taken into account. Besides device dependent information, we must also consider localization information, which includes location and local time. For example, some services may only be available in certain locations at given times of the day or provide dedicated QoS aspects only at certain locations or a client may wish to access a service always at a fixed time a day. These issues are relevant especially for digital devices in wireless networking environments.

5. Conclusion

Service orientation is a very powerful mechanism for building Web based applications. Although Web service technologies, in their current state of development, lack many features to build complex software systems, the increasing interest in doing research in order to overcome the problems that arise when developing distributed large-scale software systems due to heterogeneity, makes the service oriented approach a very promising technology in the future years. With SOAP, WSDL and also UDDI, three standardization efforts have already been implemented and realize a Web service environment in its basic functionalities. We also addressed current lacks of service oriented architectures and mentioned some proposals for solving open problems. Service composition, in fact, is the final step to develop complex business applications and support the handling of dynamic changes in the service environment.

6. References

[1] Christensen, Erik, Francisco Curbera, Greg Meredith and Sanjiva Weerawarana, *Web Services Description Language (WSDL) 1.1*, W3C Note, W3C, http://www.w3.org/TR/wsdl, 15 March 2001.

[2] Gudgin, Martin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau and Henrik Frystyk Nielsen, *SOAP Version 1.2 Part 1: Messaging Framework*, W3C Recommendation, W3C http://www.w3.org/TR/soap12-part1/, 24 June 2003.

[3] OASIS Open 2005, *UDDI: Advancing Web Services Discovery Standard*, OASIS, http://www.uddi.org/, 2005.

[4] Alonso, Gustavo, Fabio Casati, Harumi Kuno and Vijay Machiraju, *Web Services: Concepts, Architectures and Applications*, Springer-Verlag, Berlin Heidelberg New York, 2004.

[5] Dustdar, Schahram and Wolfgang Schreiner, *A Survey on Web Services Composition*, International Journal of Web and Grid Services (IJWGS), January 2005.

[6] Cabrera, Luis Felipe et al., *Web Services Coordination (WS-Coordination)*, BEA, IBM, Microsoft, http://www-106.ibm.com/developerworks/library/specification/ws-tx/, September 2003.

[7] Cabrera, Luis Felipe et al., *Web Services Atomic Transaction (WS-AtomicTransaction)*, BEA, IBM, Microsoft, ftp://www6.software.ibm.com/software/developer/library/WS-AtomicTransaction.pdf, November 2004.

[8] Cabrera, Luis Felipe et al., *Web Services Business Activity Framework (WS-BusinessActivity)*, BEA, IBM, Microsoft, ftp://www6.software.ibm.com/software/developer/library/ws-busact200401.pdf, 2004.

[9] Andrews, Tony et al., *Specification: Business Process Execution Language for Web Services Version 1.1*, BEA, IBM, Microsoft, SAP, Siebel, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/, May 2003.

[10] Arkin, Assaf et al., *Web Service Choreography Interface (WSCI) 1.0*, W3C Note, W3C, http://www.w3.org/TR/wsci/, 8 August 2002.

[11] Austin, Daniel, Abbie Barbir, Ed Peters and Steve Ross-Talbot, *Web Services Choreography Requirements*, W3C Working Draft, W3C, http://www.w3.org/TR/2004/WD-ws-chor-reqs-20040311/, 11 March 2004.

[12] Burdett, David and Nickolas Kavantzas, *WS Choreography Model Overview*, W3C Working Draft, W3C, http://www.w3.org/TR/2004/WD-ws-chor-model-20040324/, 24 March 2004.

[13] Kavantzas, Nickolas, David Burdett, Gregory Ritzinger, Tony Fletcher and Yves Lafon, *Web Services Choreography Description Language Version 1.0*, W3C Working Draft, W3C, http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/, 17 December 2004.

[14] Bunting, Doug et al., *Web Services Composite Application Framework (WS-CAF) Ver1.0*, Sun Microsystems, http://developers.sun.com/techtopics/webservices/wscaf/primer.pdf, 28 July 2003.

[15] Bunting, Doug et al., *Web Services Context (WS-Context) Ver1.0*, Sun Microsystems, http://developers.sun.com/techtopics/webservices/wscaf/wsctx.pdf, 28 July 2003.

[16] Bunting, Doug et al., *Web Services Coordination Framework (WS-CF) Ver1.0*, Sun Microsystems, http://developers.sun.com/techtopics/webservices/wscaf/wscf.pdf, 28 July 2003.

[17] Bunting Doug et al., *Web Services Transaction Management (WS-TXM) Ver1.0*, Sun Microsystems, http://developers.sun.com/techtopics/webservices/wscaf/wstxm.pdf, 28 July 2003.

[18] Banerji, Arindam et al., *Web Services Conversation Language (WSCL) 1.0*, W3C Note, HP, http://www.w3.org/TR/2002/NOTE-wscl10-20020314/, 14 March 2002

[19] xCBL.org, http://www.xcbl.org/, February 2005.

[20] ebXML, http://www.ebxml.org/, February 2005.

[21] OASIS Electronic Business Service Oriented Architecture (ebSOA) TC, OASIS, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebsoa, February 2005.

[22] Srivastava, Biplav and Jana Koehler, *Web Service Composition – Current Solutions and Open Problems*, IBM, http://www.zurich.ibm.com/pdf/ebizz/icaps-ws.pdf

[23] WSMO Working Group, Web Service Modeling Ontology, http://www.wsmo.org/, February 2005.