Web Services: Foundation and Composition

JENS HÜNDLING AND MATHIAS WESKE

SERVICE ORIENTED ARCHITECTURE PARADIGM

The paradigm shift from product- to service-oriented information systems and infrastructure is currently shaking the markets both from an economic and technical point of view. Today, information systems are often companies’ core assets in their attempts to reach their goals of offering and selling products and services in an increasingly competitive market environment. However, the requirements of today’s dynamic business scenarios on the one hand and tight software budgets of many players on the other hand in many cases render inadequate development of information systems from scratch. Instead, assembling information systems from existing software components is the way to go. These software components can be specified as services. On an abstract level, business applications are built from local services and services provided by other companies. In this paper we will explain the principles of an architecture supporting these requirements and – at the same time – the evolution of a new paradigm: The Service Oriented Architecture (SOA) (Burbeck 2000). To make this abstract architecture more concrete, we describe a current implementation of the SOA, based on Web services technology.

The Service Oriented Architecture is another approach and – as will be detailed in this paper – a promising approach in the struggle of building reusable software components (Leymann et al. 2002). This evolution started with the idea of functional decomposition, i.e. breaking down a large system into distinct parts. This approach led to application programming interfaces as a collection of operation descriptions. Object orientation was the next concept; it encapsulated data and functionality as logical units of objects and classes to simplify software reuse. Service orientation is often seen as a new paradigm (Burbeck 2000, Mohan 2002). Organizations are offering services, implemented by a variety of software components. As we will see in the second part of the paper, services can be composed to reflect business processes, which in turn can be offered as a single service afterwards.

In a SOA, functionality of software systems is provided by services. Organizations may use services offered by other companies, and companies may provide services to the market. In addition, information systems may use a variety of services to produce additional, higher-level services. The key requirements of this scenario are as follows:

- **Service description:** The ability to describe services in a standardized way, so that potential users of a given service have sufficient information to decide to use the service.

Abstract

Today both business analysts and information systems engineers attribute a great potential to Web services as a vehicle to simplify the interoperability of services offered by different organizations in electronic business scenarios. In this paper, the Service Oriented Architecture is explained as the foundation of this new technology. The main implications and benefits of this architecture and the new possibilities offered are discussed. Since standards play an increasingly important role in this new technology, new and upcoming standards for an implementation of this architecture are summarized; these combined standards are known as Web services technology. Since the real potential for Web services is facilitating business processes, a recently specified framework for defining and executing business processes in a Web services environment is presented. Rather than solely discussing the strategic benefits of Web services, this paper also tries to point out current technological deficiencies and recent approaches to overcome them.

Keywords: Web services, Web services composition, Service Oriented Architecture

Authors

Mathias Weske (weske@hpi.uni-potsdam.de) is a Professor of Computer science at the Hasso Plattner Institute for Software Systems Engineering at the University of Potsdam, Germany, where he leads a business process technology research group. His current research interests include workflow management, case handling, Product Family Engineering, and web services technology. He is a member of the GI, vice chair of the executive committee of GI SIG EMISA, and a member of IEEE and ACM.

Jens Hündling (huendling@hpi.uni-potsdam.de) is a Research Assistant in a business process technology group at the Hasso Plattner Institute for Software Systems Engineering at the University of Potsdam. His research interests include workflow management and B2B information integration using Web services technology.
The Service Oriented Architecture implies advantages for participating companies. In an existing application built with services offered in a Service Repository, all partners can update their services internally as long as the public description does not change. If the description changes, other partners will automatically adopt this change when they lookup the new, modified service description in the Service Repository. Using a description of a new published service (e.g., faster, cheaper), this improved service can be integrated seamlessly. In addition, new potential business partners can use public descriptions to offer their own implementations of the services. As a result, individual parts of a complex application can be exchanged without rebuilding the whole application. This change envisions a new kind of business partnerships, commonly known as Business-To-Business (B2B) integration (Bussler 2001, Linthicum 2001). Traditional B2B is not new—in fact it started decades ago! Today it ranges from rigid point-to-point EDI (Electronic Data Interchange) transactions to electronic marketplaces. But traditional approaches require rather high investments and, in many situations, the flexibility offered is rather low. Based on the Service Oriented Architecture and realized in Web services technology, the flexibility is enhanced, time-to-market of new applications can be reduced. In this context, it is important to stress that Web services technology is the first major technological approach that vendors like IBM, Microsoft and BEA Systems join forces on common standards. Hence, it can be expected that Web services will have a massive market presence in the near future.

Some aspects of the SOA are already present in traditional middleware concepts such as CORBA (http://www.corba.org) or J2EE (http://java.sun.com/j2ee/), e.g. a dedicated broker component that does brokerage between service requestors and service providers. As will be detailed in the remainder of this paper, Web services represent a lightweight approach. This means that it is based on open and ubiquitous communication and data format standards, rather than specific software technology. This means that large-footprint software systems (such as a CORBA environment) are not required to run Web services, which is a strong benefit, particularly with respect to applications involving multiple companies. However, CORBA and J2EE may well be used to implement Web services.

**WEB SERVICES FOUNDATION**

Web services are a current implementation of the Service Oriented Architecture. Internet and print media (e.g. WebServices.Org 2002, Web Services Journal 2002, Cover 2002, Cerami 2002) as well as software vendors (e.g. BEA Systems 2002, IBM 2002a, Microsoft 2002, Oracle 2002, SAP 2002, Sun Microsystems 2002) have announced massive support for Web services and are offering toolkits and Web services wrappers for their software products. In addition, Web services are intensively discussed in research communities, as indicated by a number of recent workshops (e.g., Buchmann et al. 2002; Chaudhri et al. 2003).

Web services are a new way of using proven and ubiquitous Internet technology, which leads to loosely coupling of tasks using, for instance, the Hypertext Transfer Protocol (HTTP) and Extensible Markup Language (XML) messaging (Bray et al. 2000, Fielding et al. 1999). Mohan (2002) defines Web services as follows:

Web services are a new breed of web application. They are self-contained, self-describing, modular applications that can be
published, located, and invoked across the web. Web services perform functions, which can be anything from simple requests to complicated business processes. Once a web service is deployed, other applications (and other web services) can discover and invoke the deployed service. XML messaging is used to interact with a web service.

Commonly accepted standards are a key requirement for a successful implementation of e-Business in a SOA. The Web services standards are based on XML technology, and are shown in Figure 2; the organization of the Web services standards shown there is also known as the Web services stack. In the top layer, the UDDI (Universal Description, Discovery and Integration) standard is the current Service Repository standard; Web services are described in WSDL (Web Services Description Language); XML messaging is typically based on SOAP, while other approaches like XML-RPC are also available. In the transport layer, HTTP is the typical protocol; however, messaging can also use other protocols like SMTP.

XML is a simple, yet flexible language for describing the format of documents that contain data. XML is derived from Standard Generalized Markup Language (ISO 8879 1986). XML documents can be represented as trees of elements with attributes and data as Unicode text. The syntactical structure of XML documents can be described in an XML Schema document (XSD), standardized by the World Wide Web Consortium (W3C 2001); XML Schemas play a key role in describing Web services, invocation information, and service registries, since all of them are specified by XML Schema documents. For an introduction to XML see Eckstein and Casabianca (2001) or, for a more detailed overview, to Harold and Means (2002). Costello (2002) provides an excellent tutorial for XML Schema.

To illustrate the concepts, we are using an example of a loan approval service. For better readability, the code examples given below omit some technical details that are not relevant for the purpose of this paper, e.g., namespaces are not completely specified.

**XML messaging with SOAP**

One of the requirements for a Service Oriented Architecture is a standardized way of service invocation, including standards for message formats and communication protocols. W3C’s XML based message protocol standard SOAP meets this requirement. Since SOAP is attributed the greatest impact today, we will sketch the SOAP approach (Box et al. 2000). Furthermore, SOAP can sit on top of various transport protocols, but HTTP is the most common.

SOAP defines an envelope for each message describing both the payload of the message and how to process it. Encoding rules for application-defined data types and a

---

<table>
<thead>
<tr>
<th>Service Repository</th>
<th>Universal Description, Discovery and Integration (UDDI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Description</td>
<td>Web Services Description Language (WSDL)</td>
</tr>
<tr>
<td>XML Messaging</td>
<td>SOAP</td>
</tr>
<tr>
<td>Transport</td>
<td>Hypertext Transfer Protocol (HTTP)</td>
</tr>
</tbody>
</table>

**Figure 2. Web Services Standards**

In this brief excerpt of a SOAP message, SOAP-ENV refers to the namespace defined by the

```xml
<SOAP-ENV:Envelope
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <SOAP-ENV:Body
        xmlns:ns1="http://www.example.com/wsd1/loanapprover.wsdl"
        SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

        <creditInformationMessage>
            <name>Peter P. Pan</name>
            <amount>10000</amount>
        </creditInformationMessage>

    </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

**Figure 3. SOAP Message Elements Hierarchy**
W3C SOAP standard; the SOAP-ENV: Body consists of the name (LoanApproval_Service) and URI of the service and names and values of its parameters (creditInformationMessage).

SOAP will evolve in the near future to incorporate additional aspects. It was originally designed to work with HTTP and consequently able to tunnel firewalls (Mohan 2002). Some systems security experts render this situation a security hole, since third parties may invoke functionality behind a company’s firewall. This is an unsolved problem, because business application can easily be accessed and the origin and authenticity of a SOAP message and other security aspects are not properly handled by the recent version of SOAP. Nevertheless, standards like WS-Security (Kaler 2002) are building on SOAP and consider security aspects, so that the current security holes will hopefully be blocked in the near future.

Another critical point is interoperability of existing SOAP implementations, because some of them create XML-messages, which others cannot process correctly. Consider a toolkit implementing only a part of the full SOAP specification with some features missing. Additionally, the SOAP specification defines some optional features, e.g. sending type information for encoded parameters. If one SOAP implementation assumes that type information will exist in messages it receives, it may not interoperate with another implementation that chooses not to send that information. To handle this kind of problems, the Web Services Interoperability (WS-I) organization (www.ws-i.org) was founded. WS-I aims at promoting Web services interoperability across platforms, operating systems, and programming languages (Ballinger et al. 2002). To summarize, SOAP describes a standardized invocation mechanism in the Web services context, and it is suitable for supporting the Service Oriented Architecture. However, major issues like security and authentication for real world applications require additional work in research and development, which is currently underway.

The Web Services Description Language (WSDL)

The next key concept of the SOA is a standardized way to describe services. Such a description must be readable and understandable by a machine and also abstract, i.e. independent from concrete implementation or protocol. It must contain a function declaration list, input and return data formats independent from concrete type system, data type declaration of the messages sent and information about the communication protocol used. Finally, a service location (e.g. a URL) is needed. It should also be readable by a human, which is especially useful during development of Web services, but also when trying to build a Web service for a given description or interpreting the semantics of a service. The Web Services Description Language (WSDL) (Weerawarana et al. 2002) meets these requirements.

The elements of WSDL documents and their relationship can be recognized in the sample WSDL document given below. The interface function list consists of one or more operation elements, defining name and message types for input and output. Thus reusing a message specification for other operations is possible. The message data formats of the elements are described by a common type system or declared within the type element. In general, every existing type system in XML can be used to specify the data formats, but it is common to use the XML Schema specification for data types – in this case no types element is required. The operations available at an endpoint are grouped together in a portType element.

Besides the abstract interface specification, a concrete communication protocol has to be specified. This is done with the binding element, which binds a portType and its abstract message definitions to concrete formats and protocols. The port element specifies a Web service’s endpoint, e.g. its location by containing an URL and the information for the communication protocol used. WSDL introduces built-in binding extensions for SOAP and some other protocols, but this does not preclude using other protocols.

Finally, it is possible that one portType is accessible through different protocols. This will lead to different ports, too, which are all included within one service element. A sample WSDL document for a simple loan approval service is given below (Cerami 2002, Khalaf 2002).

```xml
<definitions name="LoanApprover"

targetNamespace=http://www.example.com/wsdl/loanapprover.wsdl

xmlns=http://schemas.xmlsoap.org/wsdl/

xmlns:soap=http://schemas.xmlsoap.org/wsdl/

xmlns:tns=http://www.examples.com/wsdl/loanapprover.wsdl

xmlns:xsd=http://www.w3.org/2001/XMLSchema">

<message name="approvalMessage">

<part name="accept" type="xsd:string"/>

</message>

<message name="creditInformationMessage">

<part name="name" type="xsd:string"/>

<part name="amount" type="xsd:integer"/>

</message>

<portType name="loanApprovalPT">

<operation name="approve">

<input message="tns:creditInformationMessage"/>

<output message="tns:approvalMessage"/>

</operation>

</portType>

</definitions>
```
means for describing a service in a SOA by expressing (Cerami 2002). In summary, WSDL provides appropriate and service requestor don’t have to deal with it directly or dynamic operation invocation. Hence, service provider from source code using tools. Correspondingly, WSDL required for actually invoking the service. Parameters; the binding element specifies all information defines the operation name and its input and output respective parameters and their data types. The port type and namespace definitions are followed by the two kinds of service requestor to use the service. The appropriate documentation WSDL file for Loan Approval Service</documentation>
<port binding="tns:Loan_Binding" name="Loan_Port">
  <soap:address location="http://soap.example.com/soap/servlet/rpcrouter"/>
</port>
</service>
</definitions>

The WSDL document shown above describes a fictive loan approval service. To use this service, the service requestor sends a message with a name and an amount, and the service replies with a response message including the notification whether the loan was accepted or not. The WSDL document holds all information for the service requestor to use the service. The appropriate namespace definitions are followed by the two kinds of messages used in this service, namely approvalMessage and creditInformationMessage, including the respective parameters and their data types. The port type defines the operation name and its input and output parameters; the binding element specifies all information required for actually invoking the service.

Today WSDL documents can easily be generated from source code using tools. Correspondingly, WSDL documents can be processed for generating proxy code or dynamic operation invocation. Hence, service provider and service requestor don’t have to deal with it directly (Cerami 2002). In summary, WSDL provides appropriate means for describing a service in a SOA by expressing abstract functionality and concrete bindings to communication protocols and service implementations; the latter distinguishes WSDL from other Interface Definition Languages (CORBA IDL).

Universal Description, Discovery, and Integration (UDDI)

This section introduces the Universal Description, Discovery and Integration (UDDI) standard for a Web services repository, put forward by the Organization for the Advancement of Structured Information Systems (OASIS). UDDI specifies the functionality, the data model and the architecture for a Web services repository (Bellwood et al. 2002).

UDDI version 3. specifies an open architecture, allowing the full range from huge public repositories for all businesses to private repositories that might share some of their service descriptions. A possible net of UDDI nodes and their interaction is depicted in Figure 4. Public repositories for storing and retrieving service descriptions imply one or more – possibly replicated – global repositories for Web services. This requires independent institutions to maintain and supervise the repositories. On the other hand, a company might want to offer some selected services in several public repositories, but they also want to store other services in an internal repository, e.g. to share resources between departments. Additionally, they might need a local test repository for developing new services or testing updates. UDDI supports all these scenarios; it allows the integrated handling of different repositories with replication and publish/subscribe mechanisms.

Web services repositories have to fulfil several functional requirements, like enabling companies to describe themselves and their services in an open and secure environment and a mechanism to discover businesses that offer desired services. The search functionality should support the requestor in choosing the ‘right’ service from the ‘best’ provider available out of a set of suitable services and providers. This could be based on selected service quality properties or statistical and historical information the repository maintains. After selecting a service, the requestor must be able to interact with the chosen service. The UDDI API specifies a SOAP based API for searching and publishing UDDI data.

The data to be stored in a Web services repository are specified in the UDDI information model that contains data types representing entities. The entities and their relationships are explained in Figure 5. Note that a technical model (tModel) can be used in different bindingTemplates. Additionally some more data has to be stored, mostly on behalf of the architecture described above. For instance, publisherAssertions describe relationships between business entities, and subscriptions describe a permanent request to keep track of changes to the entities described by 112
WEB SERVICES COMPOSITION

Today executing business processes within an organization is a well-known topic, and dedicated software systems known as workflow management systems are employed in many organizations to model and control the execution of business processes. However, workflows involving multiple companies – so called interorganizational workflows – are still in their infancy. Sources for issues in this context include heterogeneity of workflow management systems and workflow languages and also heterogeneity of the invoked applications. Conceptual research results on interorganizational workflows (van der Aalst and Weske 2001; Buchmann et al. 2002; Grefen et al. 2000; Martens 2001) are available, but no integrated software supporting interorganizational workflows has been implemented yet. In this section, we will examine a standard for integrating business process management into a Web services environment.

We briefly describe a recent approach that is likely to have considerable impact, since it is backed by major software vendors.

The Web services foundation explained above was designed for building a Service Oriented Architecture and concentrates on single Web services, but it does not define methods how individual services can be composed to a reliable and dependable business solution with an appropriate level of complexity (Leymann et al. 2002).

We will concentrate on the viewpoint of a single business process within one organization, calling Web services. In fact, each business process consists of a number of activities, which are executed according to a pre-defined process specification. The execution order is specified by control flow constructs. For instance, sequences, branches and joins are typical control flow constructs linking workflow activities. In the Web services context, workflow activities are executed by Web services. We mention that Web services composition is a recursive concept: Each composed Web services process may consist of both atomic Web services and composed Web services. In the workflow enactment side, the workflow execution control (the workflow engine) is responsible for invoking the respective Web services at the right time with the right parameters, so that the overall business process is executed correctly and the business goals are met.
Executing Web Services Processes

The Business Process Execution Language for Web Services (BPEL4WS or, abbreviated BPEL) is an XML-based process definition language (Thatte et al. 2002) that allows companies to describe processes for Web services environments. IBM, Microsoft and BEA Systems specified BPEL as a joined follow up to IBM’s Web Services Flow Language (WSFL) specified in Leymann (2001) and Microsoft’s XLANG by Thatte (2001). WSFL resulted from graph-based workflow languages like the one used in IBM’s MQ Series Workflow. XLANG was the script-like and block-structured language for Microsoft’s BizTalk software.

BPEL is script-like and designed like traditional programming language (block) structures. This indicates direct execution of a BPEL process description through a process engine. IBM offers the first implementation of a BPEL engine for Java called BPWS4J (IBM 2002b). In the following, the main elements of BPEL will be explained.

For a better understanding of the BPEL approach, we will show a small sample process dealing with loan approval, which consists of a sequence with three activities. The example is adapted from Khalaf (2002). In the beginning, the two partners (customer and approver) and message containers (request and approvalInfo) are defined. The following sequence element defines the process structure, and consists of a sequence with three activities. The example is adapted from Khalaf (2002). In the beginning, the two partners (customer and approver) and message containers (request and approvalInfo) are defined. The following sequence element defines the process structure, and consists of an initiating receive activity. In a second step, another loan approve Web service is invoked. Finally, the result is sent back to the customer. This example simply uses another loan approval service and offers it as an own service of higher business value.

```
    name="loanApprovalProcess">
  <partners>
    <partner name="customer" serviceLinkType="loanApproveLinkType" myRole="approver"/>
    <partner name="approver" serviceLinkType="loanApproveLinkType" partnerRole="approver"/>
  </partners>
  <containers>
    <container name="request" messageType="CreditInformationMessage"/>
    <container name="approvalInfo" messageType="approvalMessage"/>
  </containers>
  <sequence>
    <receive name="receiveapproval" partner="customer" portType="loanApprovalPT" operation="approve" container="request" createInstance="yes">
      <invoke name="invokeapprover" partner="approver" portType="loanApprovalPT" operation="approve" inputContainer="request" outputContainer="approvalInfo"/>
    </invoke>
    <reply name="replycustomer" partner="customer" portType="loanApprovalPT" operation="approve" container="approvalInfo"/>
  </sequence>
</process>
```

The root element is the process element, which embodies the process description and contains structuring activities like sequence, switch and while. Their semantics are the same as commonly known from programming languages. A flow element describes concurrent execution of the activities within. Separate fault handling and compensation capabilities for a nested group are declared with a surrounding scope element. Interaction with partners of the process is possible with invoke, receive and reply activities, which specify the related Web services by the triplet portType, operation and partner. Links are used to connect source activities to target activities and thus are enhancing the block-structured control flow of BPEL.

The process-relevant data are stored in globals named containers and can be manipulated by assign activities. Thus the data flow is not very complex, but for BPEL being a language for process execution in a determined domain it seems to be appropriate.

One of the special features of BPEL is the pick activity. It contains handlers for events including message events (onMessage with portType, operation and partner) and timed events like duration or deadline. Only the first event handler of a pick element to receive its event will be executed. The process is instantiated if a message is sent to a receive or a pick activity with a createInstance attribute set to true, as in the ‘receiveapproval’ receive activity in the example above. This could be a SOAP message from a partner to a portType and operation. If an error occurs during execution, normal processing terminates and control is transferred to the corresponding faultHandler. In addition, compensationHandlers define a compensating activity that needs to be called when something in the process has gone wrong and certain effects have
to be undone. The need for an independent representation of the interaction between parties is taken into account with the abstract process in BPEL, which we will not explain in detail here.

One key feature of a Service Oriented Architecture is dynamically finding a service and binding it at runtime. This makes sense in a process, where one or more activities are dynamically bound to a Web service, potentially depending on process instance data. BPEL does not support this directly; consequently such a process step has to be an activity, representing a service, implemented for the purpose of dynamically finding a Web service in a repository like UDDI. Nevertheless, the interactive activities inside a BPEL process represent loosely coupled Web services. Furthermore, the initiating receive and pick activities imply that the process is provided as a (composed) Web service, which makes conversational interactions between stateful, long-lived services possible. This is another motivation for the need of coordination explained in the next section.

Coordinating Web Services and Processes

In a BPEL process the distributed Web service activities need to coordinate their work for the overall process to succeed, especially when interoperability is needed across vendor implementations, trust domains, etc. Consider a scenario where an order already placed has to be cancelled because some other activity produced a fault. Thus, especially for long running business processes in a Service Oriented Architecture this coordination is necessary. The participants in a business process often already have specialized software systems for this purpose.

The basic idea is to wrap proprietary protocols for transaction processing, workflow and other systems and thus to operate in a heterogeneous environment, which is typical for a Web services based process. Web Services Coordination (WS-Coordination) and Web Services Transaction (WS-Transaction) specify a framework that enables participants to reach consistent agreement on the outcome of distributed activities. A BPEL engine may use coordinating middleware, thus the application developer should not be concerned about the low-level coordinating messages explained below.

Web Services Coordination. It is obvious that common information has to be shared between Web services to achieve coordination. The WS-Coordination specification supplies a framework for this purpose (Cabrera et al. 2002b). WS-Coordination specifies how to create information (coordinationContext) needed to propagate the Web service to other services and how to register for coordination protocols. The framework bases on a coordinator, offering activation services (AS), a registration service (RS) and a set of protocols called coordination types. The services themselves are implemented as Web services. The WS-Transaction specification explained below contains a set of predefined coordination types.

For a clearer understanding of the coordination, consider the example depicted in Figure 6. Two applications (A1 and A2) want to realize a coordinated interaction. The protocol services Y1 and Y2 are realizing a protocol Y, which is specific to a coordination type CT. The concrete coordination type and its semantics are not relevant for this example.

Coordination starts with A1 calling AS1 of its preferred coordinator C1 (1a) with the request to create a coordination context for coordination type CT. The reply (1b) contains a coordination context, with an identifier for A1, a reference to RS1 and the coordination type CT. A1 sends this coordination context to A2 (2) and A2 passes it to AS2 of its preferred coordinator C2 (3a). The activation service replies with a new coordination context.
context, containing the old A1 and CT, but a new reference to RS2 (3b). The latter is used by A2 to determine Y as appropriate protocol and registers for it (4), which results in a logical connection between Y2 and A2. When registering, A2 passes the RS1 reference to RS2, which is triggered to forward the registration to RS1 (5). RS1 and RS2 exchange port references for Y2 and Y1, which implies a logical connection between Y1 and Y2. Afterwards A1 and A2 can coordinate their work with a protocol compliant to coordination type CT.

**Web Services Transaction.** The WS-Transaction specification (Cabrera et al. 2002a) is closely related to the WS-Coordination specification, because it defines the coordination types mentioned above that are used for the coordination of distributed activities. The extensible framework distinguishes between atomic transactions with an all-or-nothing property and business activities for long-running business transactions. The atomic transaction coordination types are used to wrap existing transaction processing systems, whereas business activities coordination types enable the wrapping of existing workflow management and business process management systems and their mechanisms.

The coordination protocols for atomic transactions are called completion, completion with acknowledgement, phase zero, two phase commit (2PC), and outcome notification. In the first two protocols, a participant registers to tell the coordinator whether to try to commit or rollback the transaction. The phase zero protocol implies a notification of the registrar before starting a 2PC protocol. The two-phase commit protocol is pretty much the same as the one used in distributed database systems. A transaction participant that wants to be notified of the outcome of the commit-abort decision registers for the outcome notification protocol.

The characteristics of business activities (BA) and atomic transactions (AT) differ in consuming resources and also a significant number of atomic transactions may be involved in a business activity. The loss of state of a BA means the loss of important historical information, e.g. losing financial history can cause legal and accounting problems. In a business activity the response may take a very long time, so locking resources is not possible. This leads to design points like reliable recording of all state transitions, acknowledgement of all messages and defining the response as a separate, asynchronous operation. The complex protocols defined for business activities are based on several messages and acknowledgements for preparing commitment and for compensation and will not be explained in more detail in this paper.

Together WS-Coordination and WS-Transaction facilitate dynamic collaboration at runtime and thus fulfil the requirements for dynamic business processes. They enhance the flexibility of the Service Oriented Architecture.

**RELATED WORK**

There are a significant number of approaches to handle the complexity in the context of Web services based processes. Some are based on existing implementations from software vendors and others are based on research projects. In the following we will introduce two approaches to describe Web service environments in general and take a look at other service-based process languages.

The Web Services Modeling Framework (WSMF) suggested by Fensel and Bussler (2002) envisions the integration of Semantic Web and Web services to intelligent Web services. The WSMF is based on two complementary principles: Strong decoupling between the components that realize the web service and strong mediation of different terminologies as well as different interaction styles. WSMF consists of four different main elements; ontologies define the terminologies that are used, goal repositories define the problem that can be solved by a web service, web services descriptions define elements of a web service; and mediators solve problems of interaction between participants. To sum up, the WSMF is based on an SOA implementation and focuses on semantic issues. A key issue is searching for an appropriate (Web) service to fulfill a special goal; processes are not considered in this framework.

An interesting approach to handle the complexity of a Web services environment and especially composed Web services is proposed by Benatallah et al. (2002). The authors identify a collection of prospective patterns addressing various activities in the life cycle of a composite Web service. The presented life cycle identifies activities like wrapping native services, setting their outsourcing agreements and assembling composite services. Afterwards the services can be executed and monitored, which finally might lead to some kind of evolution. The presented patterns deal with service wrapping, negotiation, discovery, composition, execution and monitoring.

In the following, we will point to some other approaches for Web service composition and process specification. An overview of several business process languages including BPEL is offered at http://www.bpel.org, (Dubray 2002).

The Business Process Management Initiative (www.bpmi.org) is a non-profit corporation and has released version 1.0 of their Business Process Modeling Language (BPML) in June 2002 (BMPI.org 2002a). Interestingly, two contributors of BPEL, IBM and BEA Systems are members of BPMI, so it was not surprising that the BPMI welcomed BPEL shortly after it was released. Furthermore, they announced their intention to identify possible convergence paths with the goal to offer a unified BPM stack. BPML is seen as a superset of BPEL, because they add nesting and complex compensation transactions (BPMI.org 2002b).
The Web Service Choreography Interface (WSCI) is a process interface definition language for business processes and can be seen as the largest common denominator of BPML and BPEL, which has contributed greatly to the consolidation of a standard BPM stack (BPML.org 2002a). WSCI is based on XML and the Web services technologies mentioned above, and it describes the flow of messages exchanged in the context of a process. Hence, it allows the description of observable behaviour of stateful Web Services, but it does not address the internal process (like BPEL does) that drives the message exchange (WSCI 2002).

OASIS released a specification for a business transaction protocol (BTP) (OASIS 2002a). Like WS-Coordination and WS-Transaction, BTP solves part of the problem of coordinating loosely coupled activities by forcing consistent termination portions, i.e. BTP also focuses on transactions being coordinated across Web services. To this end, BTP and WS-Transaction are competing standards that differ significantly but are geared towards similar types of work (Sessions, 2002).

CONCLUSIONS

In this paper, we have discussed the Service Oriented Architecture, an architecture that is well capable of handling the complexity of today’s most promising approach towards Business-to-Business integration. We presented service orientation as a new paradigm and explained the Service Oriented Architecture. Web service technology was introduced and some problems and implications were discussed. Based on Web services foundation, current approaches to marry Web services technology with business process technology are described. These concepts are enlarge the flexibility of business applications through loose coupling and late binding. As a result, companies can dynamically choose the most suitable service with the best quality available, exactly when it is needed. This fosters new business opportunities also for small and medium sized enterprises. Nevertheless, we indicated business-critical issues like compatibility of implementation, security and performance that are still unsettled. Choosing service providers dynamically is lowering the market entrance barrier for providers of niche services. Nevertheless, new service providers have to prove their reliability. Offering Web pages for instance is well known, but offering remote service invocation has quite different impact on scalability and availability, and this is not properly explored.

The World Wide Web Consortium (W3C 2002) and the Organization for the Advancement of Structured Information Standards (OASIS 2002b) have been forums for many Web service related standardization activities. Today’s Web service technology is based on a bunch of standards, and even if some of them will not survive the next evolution step, we believe the concepts and the vision will emboss future software systems.

ACKNOWLEDGEMENTS

The authors appreciate the detailed comments by the anonymous reviewers, which helped us to improve the paper.

References


