14. Web Services and Peer-to-Peer

Markus Hillenbrand, Paul Müller (University of Kaiserslautern)

14.1 Introduction

Peer-to-Peer and Web services both address decentralized computing. They can be considered as rather distinct from each other, but a closer look at the Web services technology reveals a great potential for a combination of both Peer-to-Peer and Web services.

The basic idea behind Web services technology is to provide functionality over the Internet that can be accessed using a well-defined interface. This idea of a service-oriented architecture forms the next evolutionary step in application design and development after procedural programming, object orientation, and component-oriented development. During the last twenty years, different middleware approaches and application designs have been introduced to leverage dated technology and provide easy access over open and mostly insecure access networks.

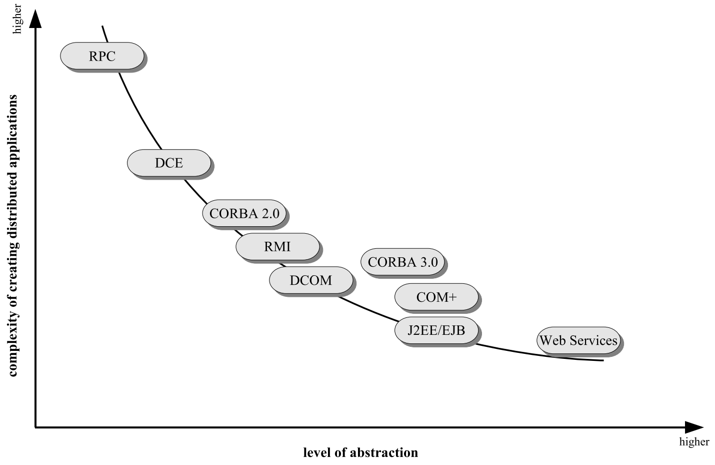


Fig. 14.1: Programming paradigms: abstraction and distribution

The most recognized and well established technologies for creating distributed systems are the Remote Procedure Call (RPC, 1988) from Sun Microsystems, the Distributed Computing Environment (DCE, 1993) from the Open Software Foundation (OSF), the Common Object Request Broker Architecture (CORBA, 1990s) from the Open Management Group (OMG), the Java Remote Method Invocation (RMI, 1990s) and Java Enterprise Beans (EJB, 1990s) from Sun Microsystems, and the Distributed Component Object Model (DCOM, 1997 and COM+, late 1990) from Microsoft. Each of these technologies introduced a higher level of abstraction for creating distributed applications and reduced the implementation effort necessary to achieve this goal. Figure 14.1 illustrates the relationship between the underlying programming paradigm, the level of abstraction, and the complexity of creating a distributed application.

Distribution aspects have always been an addendum to procedural programming and object-orientation (mostly using some kind of remote procedure call mechanism) and are not intrinsic to the paradigms. Solutions following the component-oriented paradigm provide middleware functionality and software containers that allow for distribution during software development and help managing the resulting software systems. In contrast to this, Web services are based on open, well-defined, and established standards and encompass distribution from within the specifications. In combination with currently evolving additional standards (cf. Chapter 14.2.6) they have a good chance to achieve the goals of a real and secure distributed middleware architecture.

The Web services technology has been initiated by industry and not academia, and more and more large companies are working on Web services technology and apply it in real world applications. Though what is the reason for this development? Unfortunately, there is no commonly used definition for Web services. Instead, several distinct definitions have to be consulted to investigate what Web services are and how to use them. Two major driving forces of the Web services technology – IBM and Microsoft – define Web services as follows:

Definition 14.1.1. *(IBM, 2003) "Web services are self-contained, modular applications that can be described, published, located, and invoked over a network. Web services perform encapsulated business functions, ranging from simple request-reply to full business process interactions. These services can be new applications or just wrapped around existing legacy systems to make them network-enabled. Services can rely on other services to achieve their goals."*

Microsoft favors a similar definition of the Web services technology, but it emphasizes standard Internet protocols:

Definition 14.1.2. *(MSDN, 2001) "A Web service is a programmable application logic accessible using standard Internet protocols, or to put it another way, the implementation of Web-supported standards for transparent machine-to-machine and application-to-application communication."*

The common aspect of definitions 14.1.1 and 14.1.2 is their focus on business and application-to-application communication. A more technical view on Web services is given by the following definition from the World Wide Web Consortium in 2003:

Definition 14.1.3. *(W3C: May, 2003) "A Web service is a software system identified by a URI (Uniform Resource Identifier), whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols."*

This definition completely abstracts from the implementation and usage of Web services and is entirely based on XML. During the definition phases of Web services related standards, the W3C has revised this definition several times to make it more specific in terms of technology while trying to keep it as general as possible. As of 2004, the current definition reads as follows:

Definition 14.1.4. *(W3C Feb, 2004) "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."*

Compared to definition 14.1.3, not only XML [89] but also WSDL [118, 115] and SOAP [265] are part of the definition. And HTTP [206] is mentioned as the typical transport protocol. This makes the definition of a Web service more precise from a technological view, but also narrows applicability and extensibility.

The relevant standards mentioned in the definitions will be briefly introduced in the next sections. A sample Web service (providing functionality to add an integer or a complex number) will be used to illustrate them.

14.2 Architecture and Important Standards

The Web services technology permits loose coupling and simple integration of software components into applications – irrespective of programming languages and operating systems by using several standards. The basic architecture is shown in figure 14.2.

Three participants interact to perform a task. A *service provider* is responsible for creating and publishing a description of a service interface using WSDL. The provider also contributes the actual implementation of the service on a server responding to requests from clients that adhere to this

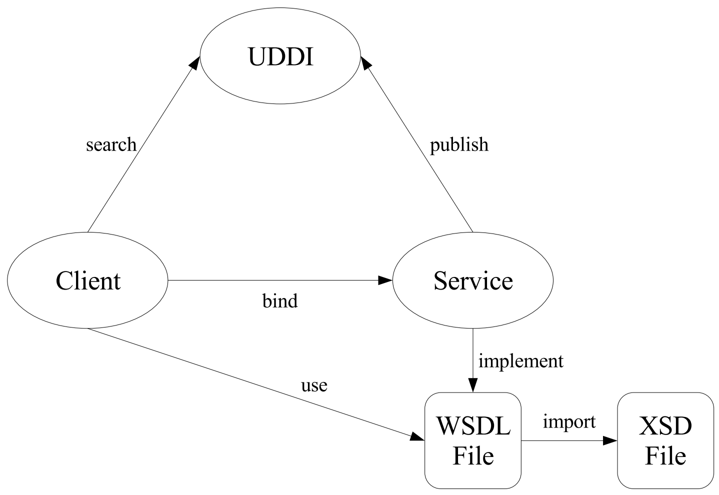


Fig. 14.2: Web services: Overview and Standards

interface description. A *UDDI registry* collects and categorizes interface descriptions and offers them to customers via a browsable directory or a search engine. A *client* can either be a human user or another software component acting on behalf of a user. It discovers a service by asking the UDDI registry and then contacts the actual service using the interface definitions and protocols defined in the associated WSDL document. This WSDL document might refer to external XML Schema (XSD) documents on the Internet where data types for the service are defined (this allows for re-use and compatible data structures).

The basic operational steps to consume a service are *publish*, *find*, and *bind*. A service provider publishes a service using a WSDL service description and the UDDI registry, a service requestor finds this service using the UDDI registry, and the service requestor binds his program to the service endpoint using the protocols defined in the WSDL document (mostly SOAP over HTTP).

The binding process on the client side can be realized using different techniques: stubs, dynamic proxy, or dynamic invocation. The automatic generation of *stubs* at compile time takes a WSDL document and creates a local representation of the remote Web service. This only allows for a tight coupling of client and service. The *dynamic proxy* technique does not create the stubs at compile time but generates a local representation of the remote service at runtime. Only a local interface definition is needed to make the actual call. *Dynamic invocation* on the other hand can be used to create a Web service call completely during runtime – which makes loosely coupled applications possible. In each case the information needed can be retrieved solely from the WSDL document.

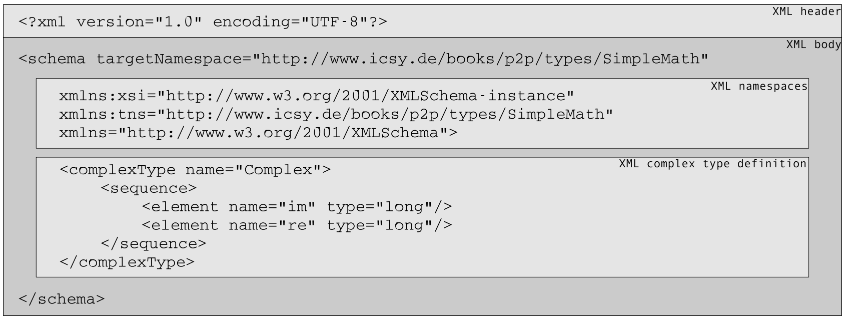
The necessary standards and protocols to either publish, find, or bind a Web service will now be explained in greater detail. Exemplification will adhere to the WSDL 1.1 specification because this version is currently widely used and has a large tool support.

14.2.1 XML and XML Schema

XML [89] is the key to platform and programming language neutral data exchange. It provides the mechanisms to create complex data structures as well as it allows for modeling dependencies between data sets. An XML document itself is a plain text file using a given character encoding scheme (e.g. ISO8859-15 or UTF-8). In the following, the necessary parts of the XML specification will be introduced to give a better understanding of the next chapters.

Structure

An XML document adheres to a well defined structure. It is divided into a *header* and a *body* part (cf. Figure 14.3). The header contains useful information for other software systems such as XML parsers. The XML version and character encoding are defined there. The body part of the XML document contains the actual data of the document. This information is contained inside XML elements and uses the "<...>" syntax known from HTML documents. Additionally, these elements can have *attributes* that provide more detailed information. The XML body part can then be seen as a tree consisting of XML elements and attributes attached to the nodes.



```
<?xml version="1.0" encoding="UTF-8"?>

<schema targetNamespace="http://www.icsy.de/books/p2p/types/SimpleMath"

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:tns="http://www.icsy.de/books/p2p/types/SimpleMath"
    xmlns="http://www.w3.org/2001/XMLSchema">

    <complexType name="Complex">
        <sequence>
            <element name="im" type="long"/>
            <element name="re" type="long"/>
        </sequence>
    </complexType>

</schema>
```

Fig. 14.3: XML Schema for the `Complex` data type

XML Namespaces (XMLNS)

XML allows to use any name as an element name. Thus, the vocabulary of XML documents is not fixed. To avoid collisions of such element names, XML namespaces [89] have been introduced in 1999 and were updated in 2004. A namespace can be defined inside an element (usually the root element) and is valid for all child elements ("XML namespaces" in figure 14.3). A namespace is specified using a Uniform Resource Identifier (URI [68]) which itself can either be a Uniform Resource Locator (URL) or a Uniform Resource Name (URN). A URL points to a specific location where more information about the namespace can be found while a URN is just a globally unique name. It is possible to use different namespaces inside an XML document, and the XML document itself can use elements from these namespaces in any suitable order.

XML Schema (XSD)

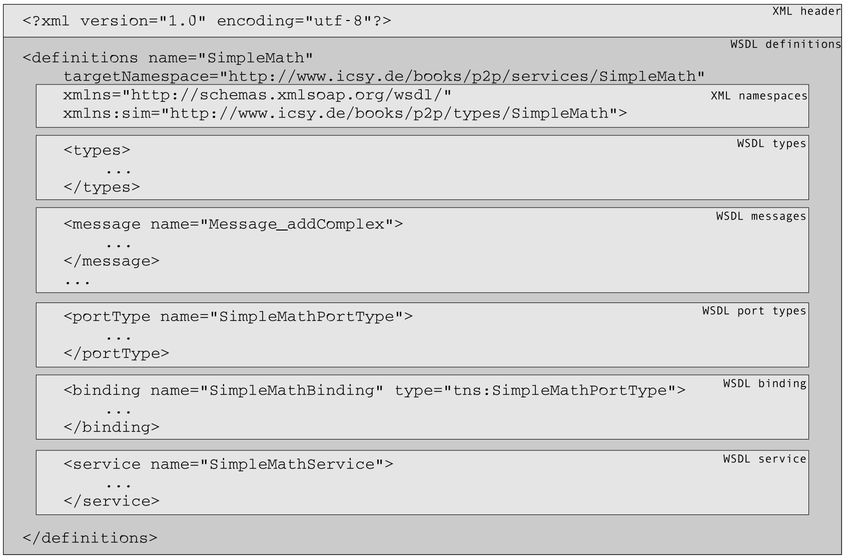
Together with XML Namespaces, XML Schema [200, 593, 75] is one important building block for creating modular XML documents. Its major goal is to make syntactical restrictions for XML elements, i.e. XML Schemas can be used to assign and define data types. Besides basic data types such as integer, string, date, etc. provided by the standard, it is possible to define new datatypes ("XML complex type definition" in figure 14.3). Using the appropriate XML Schema elements, it is further possible to define new simple (primitive) data types, complex data types (like structures, arrays, etc.) as well as enumerations and choices. It is also possible to define and assign structural patterns restricting the range of values for the data types. Additionally, XML Schemas can be imported into other XML documents, e.g. WSDL documents. This allows for re-use of XML data types and a modular design of XML documents.

14.2.2 WSDL

The Web Services Description Language (WSDL [118, 115]) is an XML based format for describing the interface of a Web service. The WSDL document starts with an XML header and the body is divided into several parts (shown in figure 14.4):

Root Element

The root element of a WSDL document is a `definitions` element and contains a target namespace, the namespaces used throughout the document



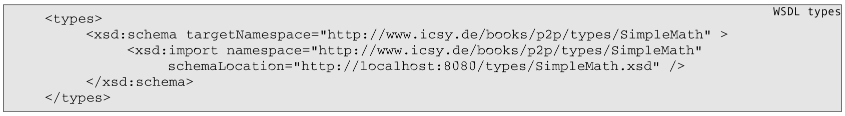
```
<?xml version="1.0" encoding="utf-8"?>

<definitions name="SimpleMath"
    targetNamespace="http://www.icsy.de/books/p2p/services/SimpleMath"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:sim="http://www.icsy.de/books/p2p/types/SimpleMath">

    <types>
        ...
    </types>

    <message name="Message_addComplex">
        ...
    </message>
    ...

    <portType name="SimpleMathPortType">
        ...
    </portType>

    <binding name="SimpleMathBinding" type="tns:SimpleMathPortType">
        ...
    </binding>

    <service name="SimpleMathService">
        ...
    </service>

</definitions>
```

Fig. 14.4: WSDL document and its structure

("XML namespaces" in figure 14.4), and an optional documentation of the Web service.

Types

The data types used by the Web service should be designed using XML Schema. Inside the `types` element it is possible to define data types for the current service or to import data types from remote documents using the XML Schema `import` element. In figure 14.5 the `types` element is used to import the Complex data type defined in figure 14.3. The actual XML Schema file location is specified using the `schemaLocation` attribute and its namespace is specified using the `namespace` attribute accordingly. The `targetNamespace` attribute can be used to map the namespace of the XML Schema document into another namespace.



```
<types>
    <xsd:schema targetNamespace="http://www.icsy.de/books/p2p/types/SimpleMath" >
        <xsd:import namespace="http://www.icsy.de/books/p2p/types/SimpleMath"
            schemaLocation="http://localhost:8080/types/SimpleMath.xsd" />
    </xsd:schema>
</types>
```

Fig. 14.5: WSDL types element used to import a XML Schema data type

Messages

Messages are exchanged between the client and the service and represent the data necessary to call a Web service function or to create a response. A `message` element has a name and several parts that make up the message. Every `part` element usually has a type – and this type is either imported or defined in the `types` element. In figure 14.6 four messages are defined. The first (`Message_addComplex`) has two child elements `x` and `y`, and the second message (`Message_addComplexResponse`) contains only one child element `result`. As the name of the message suggests, it is used as a response to the first message. Messages three and four work in the same manner.

WSDL messages

```
<message name="Message_addComplex">
     <part name="x" type="sim:Complex"/>
     <part name="y" type="sim:Complex"/>
</message>
<message name="Message_addComplexResponse">
     <part name="result" type="sim:Complex"/>
</message>
<message name="Message_addInt">
     <part name="x" type="xsd:int"/>
     <part name="y" type="xsd:int"/>
</message>
<message name="Message_addIntResponse">
     <part name="result" type="xsd:int"/>
</message>
```

Fig. 14.6: WSDL message element

Port Types

A Web service can have several `porttype` elements[1], each containing a set of operations provided by the Web service. The port types use the messages defined using the `message` elements to create input and output messages for each `operation`. In figure 14.7 the two operations `Message_addComplex` and `Message_addInt` are defined using the messages from figure 14.6 and thus form a request-response operation `addComplex`. With WSDL 1.1 other operation types are possible: one-way (the endpoint behind the operation receives a message), solicit-response (the endpoint receives a message and sends a correlated message), and notification (the endpoint sends a message).

Bindings

The `binding` element assigns a data encoding format and a transport protocol to the Web service operations. It is possible to assign more than one protocol

[1] In the WSDL 2.0 specification the `porttype` element has been renamed to `interface` and extended to support more types of communication.

WSDL port type

```
<portType name="SimpleMathPortType">
     <operation name="addComplex" parameterOrder="x y">
          <input message="tns:Message_addComplex"/>
          <output message="tns:Message_addComplexResponse"/>
     </operation>
     <operation name="addInt" parameterOrder="x y">
          <input message="tns:Message_addInt"/>
          <output message="tns:Message_addIntResponse"/>
     </operation>
</portType>
```

Fig. 14.7: WSDL port type element

to the same operation. In figure 14.8 both operations are defined to use SOAP over HTTP.

WSDL binding

```
<binding name="SimpleMathBinding" type="tns:SimpleMathPortType">
     <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
     <operation name="addComplex">
          <soap:operation/>
          <input><soap:body use="encoded"
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="http://www.icsy.de/services/test/SimpleMath"/></input>
          <output><soap:body use="encoded"
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="http://www.icsy.de/services/test/SimpleMath"/></output>
     </operation>
</binding>
```

Fig. 14.8: WSDL binding element

Service

The service element finally defines for each `binding` a `port` as the actual endpoint, i.e. the place in the network where the actual software runs and offers the service[2]. In figure 14.9 the `binding` defined in figure 14.8 is assigned to the SOAP access point provided by the software running on `localhost` on port `8080`.

WSDL service

```
<service name="SimpleMathService">
     <port name="SimpleMathPort" binding="tns:SimpleMathBinding">
          <soap:address location="http://localhost:8080/axis/services/SimpleMathPort" />
     </port>
</service>
```

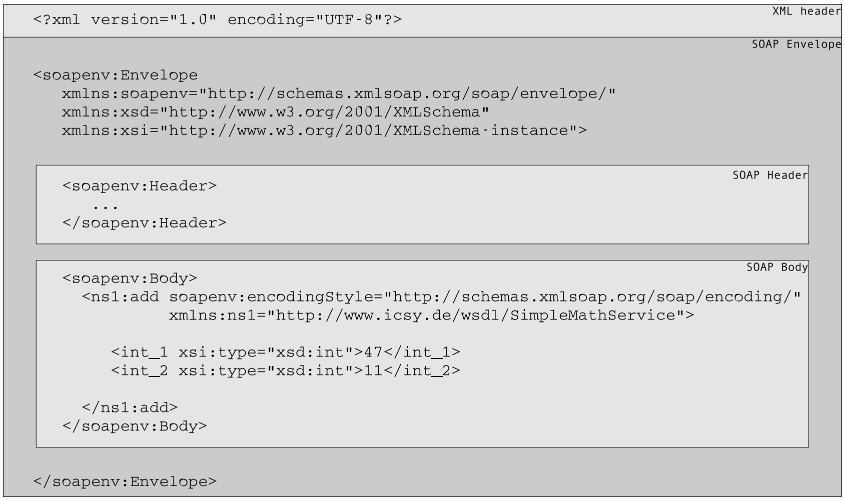
Fig. 14.9: WSDL service element

[2] In the WSDL 2.0 specification the `port` element has been renamed to `endpoint` in order to clarify the meaning.

14.2.3 SOAP

Designed as an XML-based lightweight protocol, SOAP[3] [265] is responsible for encoding and exchanging data between applications. According to definition 14.1.4 it is used as a communication means between service providers, service requestors, and service brokers. A SOAP message itself can be transported using various transport protocols. Most applications use HTTP (Hypertext Transfer Protocol [206]) as the underlying transport protocol; other protocols are SMTP (Simple Mail Transfer Protocol [357, 433]) or BEEP (Blocks Extensible Exchange Protocol [525]).

A SOAP message acts as a message container that delivers structured and typed data between applications. A SOAP message has three elements:



```
<?xml version="1.0" encoding="UTF-8"?>

<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

   <soapenv:Header>
      ...
   </soapenv:Header>

   <soapenv:Body>
     <ns1:add soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
              xmlns:ns1="http://www.icsy.de/wsdl/SimpleMathService">

        <int_1 xsi:type="xsd:int">47</int_1>
        <int_2 xsi:type="xsd:int">11</int_2>

     </ns1:add>
   </soapenv:Body>

</soapenv:Envelope>
```

Fig. 14.10: The SOAP message structure

The mandatory *envelope* ("SOAP envelope" in figure 14.10) provides a container for the next two elements and is the XML root element where referenced XML namespaces have to be defined.

The optional *header* ("SOAP header" in figure 14.10) can be used to transport additional information to recipients of a SOAP message. A recipient can either be the final destination of the message or any intermediate entity routing the message through a complex distributed Web service application. It can be used for routing information, information about quality of service, billing purpose, etc.

[3] Up to version 1.1 SOAP was an acronym for *Simple Object Access Protocol*. This is no longer the case, SOAP has become a term on its own. One reason for this is that Web services should not be conceived as objects.

The mandatory *body* (“SOAP body” in figure 14.10) finally carries all application specific information for the final recipient. This final recipient must be able to semantically understand the body elements. A fault element inside the body can be used to carry an error message to one of the intermediaries or back to the origin of the message.

An additional standard allows for attachments to be transmitted in MIME encoded form, enabling Web services to process large binary data files.

14.2.4 HTTP

The Hypertext Transfer Protocol (HTTP [206]) is a stateless application-level protocol for exchanging data between two entities. It is primarily used by Web browsers to access Web servers and retrieve HTML pages. Several extensions concerning request methods, header information, and error code have widened the scope of applicability. In the context of Web services it is the most commonly used protocol for exchanging SOAP messages between a client and a Web service.

14.2.5 UDDI

Universal Description, Discovery and Integration (UDDI [454]) can be used to publish or find a specific Web service. UDDI is basically a directory service providing registration and search capabilities for Web services. Such a UDDI registry offers a Web service interface for service providers and service requestors. Based on several meta data information structures and well established categorization formalisms, either of them can store or retrieve Web service information, respectively.

A globally synchronized UDDI registry is currently maintained by IBM, Microsoft, and SAP. It is also possible to establish a private UDDI for closed user groups or applications.

14.2.6 WS-*

In addition to the basic underlying standards and protocols some industry driven standardization efforts are undertaken to retrofit Web services for commercial and secure usage. They are usually referred to as WS-* standards, where * is a placeholder for the purpose of the standard. The following descriptions of the most relevant WS-* standards have been taken from their respective specification; a complete introduction can be found there.

WS-Addressing

The WS-Addressing [86] standard provides transport-neutral mechanisms to address Web services and messages. The specification defines XML elements to identify Web service endpoints and to secure end-to-end endpoint identification in messages. It furthermore enables messaging systems to support message transmission through networks that include processing nodes such as endpoint managers, firewalls, and gateways in a transport-neutral manner.

WS-Federation

WS-Federation [331] defines mechanisms that are used to enable identity, account, attribute, authentication, and authorization federations across different trust realms.

WS-Policy

The Web Services Policy Framework [540] provides a general purpose model and corresponding syntax to describe and communicate the policies of a Web service. It defines a base set of constructs that can be used and extended by other Web services specifications to describe a broad range of service requirements, preferences, and capabilities.

WS-ReliableMessaging

WS-ReliableMessaging [205] describes a protocol that allows messages to be delivered reliably between distributed applications in the presence of software component, system, or network failures. The protocol is described in an independent manner allowing it to be implemented using different network transport technologies.

WS-ResourceFramework

The Web Services Resource Framework [258] defines a family of specifications for accessing stateful resources using Web services. The motivation for these specifications is that while Web service implementations typically do not maintain state information during their interactions, their interfaces must frequently allow for the manipulation of state, that is, data values that persist across and evolve as a result of Web service interactions.

WS-Security

WS-Security [435] describes enhancements to SOAP messaging to provide message integrity and confidentiality. It can be used to accommodate a wide variety of security models and encryption technologies. The specification also provides a general-purpose mechanism for associating security tokens with message content.

WS-Transaction

The WS-Transaction [377] standard describes coordination types – namely Atomic Transaction (AT) and Business Activity (BA) – for building applications that require consistent agreement on the outcome of distributed activities.

14.3 Service Orchestration

As Web services evolve and are deployed on a larger scale, the need for the combination of several Web services in order to create a business process arises. Several languages and specifications can be identified that deal with service orchestration. The most relevant are XML Process Definition Language (XPDL [452]), Business Process Modeling Language (BPML [33]), Web Service Choreography Interface (WSCI [34]), Electronic Business using eXtensible Markup Language (ebXML [455]), and Business Process Execution Language for Web services (BPEL4WS [588]). The latter is currently the most promising candidate for a common standard.

BPEL4WS is based on XML and can be used to combine distributed Web services to a business process. Interaction between Web services can be modeled as well as between the business process and its clients. The clients can thus be detached from the actual business logic and be kept simple.

BPEL4WS is driven by major companies such as IBM and Microsoft and provides a language to implement complex processes by allowing for different actions like calling a Web service, manipulating data, and handling errors. Flow control can be realized using control flow statements like tests, loops, and threads. To the outside, a BPEL4WS business process can be described like a normal Web service and have its own WSDL description – a client does not need to know the internal structure or control flow of the process.

14.4 Comparison of Peer-to-Peer and Web Services

As Peer-to-Peer and Web services are both addressing decentralized computing, it is reasonable to compare the two techniques and show differences that might also be used as incentives for further research and development in this area.

14.4.1 What Can Peer-to-Peer Learn from Web Services?

The Web services standards evolve at a high rate and influence other technologies as well. There are several issues that also concern Peer-to-Peer technology:

XML

All data formats and all data exchange protocols in the Web services area are based on XML. XML Schema is used to define platform and programming language neutral data types, SOAP is used to transfer these data types to the service, and WSDL is used to describe the service itself. New XML standards or enhancements can be integrated into the Web services technology with small effort, as XML security or XML encryption have shown.

Another benefit would be to use XML schema definitions for describing resources, data, services, and peers within a Peer-to-Peer system with meta data. An XML based description of the resources and data shared in a Peer-to-Peer system would be more flexible (with regard different schema files and namespaces) and extensible, because a schema file can be easily extended without having effect on existing software and thus allow for a smooth upgrade or change in meta data description. A more detailed view on schema-based Peer-to-Peer systems is given in chapter 19.

Service Registration

Irrespective of its rather centralized approach, Web services provide an elegant registration mechanism with thorough content description and enhanced search capabilities. Classification schemes can be used to categorize or classify services so that users are able to find them using different search requests.

Security

Web services security is of major importance for Web services to be accepted as building blocks for distributed applications running on the Internet. Several standards have been developed to enable secure communication between Web service entities. These security standards are mostly based on XML and are thus not limited to the Web services world.

Interoperability

One of the design goals of Web services has been to be as open and interoperable as possible. Standardized interfaces (written in WSDL) can be used and accessed by any system capable to process XML documents. There is no artificial language or operating system barrier in a Web services scenario. Together with security standards this accounts for large business processes and applications to be deployed over the Internet using different programming languages and operating systems.

Service orchestration

Web services can be combined to create a business process using Web service orchestration. This allows for re-use and encapsulation. The JXTA SOAP project (`http://soap.jxta.org`) for example brings together Web services and Peer-to-Peer technology by defining a bridge between SOAP and the JXTA protocol. This can be further extended by defining workflows on top of these services. JXTA is explained in more detail in chapter 21.3.1.

14.4.2 What Can Web Services Learn from Peer-to-Peer?

Web services and Peer-to-Peer technologies are used to decentralize computing. However, Web services are based on a client/server architecture. In the following, some aspects of Peer-to-Peer systems will be highlighted that might be applied to the Web services world:

Decentralization

In a Web services scenario, a rather centralized UDDI registry is used to publish and find Web service descriptions. This accounts for a very easy usage but also means that all clients and service providers have to access this single service (or a few central services) and it thus might form a bottleneck and single point of failure. Additionally, the UDDI does not know whether a service is currently available or not. It only delivers stored information to the service requestors. In a Peer-to-Peer system, every node offers its service and distributed search algorithms are used to retrieve information from all nodes. A service currently not available will usually not be found in the system.

Transport Protocols

The success of the World Wide Web and Web services is partly based on the simplicity and scalability of HTTP. Operating in real time and being stateless allows for a tight coordination between client (browser) and server (Web server) – with little overhead. But in systems with a high need for synchronization (like instant messaging) HTTP is inadequate due to its design. This also applies to services that need a lot of time to process a request (large data base operations or complex calculations). HTTP is designed to deliver an answer immediately. Some systems have instead adopted the Simple Mail Transfer Protocol (SMTP) for asynchronous messaging in this case. But there are several other protocols that might prove useful in different usage scenarios. Especially Peer-to-Peer instant messaging protocols are designed to allow for a flexible two-way communication.

Addressing Scheme

Peer-to-Peer systems operate mostly outside the Domain Name Service (DNS) because its nodes might not have a permanent IP address. In order to access the resources of these nodes, a logical and often user-created address is continuously mapped to the current IP address. For Web services this could mean to make them accessible by using different addressing schemes and not only using IP addresses or host names, respectively.

Client/Server Architecture

On the World Wide Web roles like client and server are largely fixed – the Web server is always a server, and a Web browser is always a client. This also applies to Web services running on a Web server. In Peer-to-Peer systems however, these roles are only temporary. A node usually acts as client or server, depending on the current task. This also affects scalability. A strong client/server architecture only scales with the servers, while a Peer-to-Peer infrastructure scales depending on the roles taken by the nodes.

14.4.3 Side-Effects Arising when Joining Web Services and Peer-to-Peer

Compared to either Web services or Peer-to-Peer alone, any combination[4] of the two technologies would theoretically cause side-effects in different areas. The following list is not complete but addresses the most important issues:

[4] Such a combination could be a Peer-to-Peer system using Web service technology or a Web service application scenario adopting Peer-to-Peer techniques.

Bandwidth

Using XML message formats and searching for services using Peer-to-Peer technology in distributed applications will increase the need for bandwidth dramatically compared to a central registry such as UDDI. If there is no central registry, a lot of nodes (peers) of the system have to be queried for their services – this is especially the case when using unstructured Peer-to-Peer systems (cf. Part II).

Security

Security in closed client/server systems can be handled very well. It is easily possible to define access control and access policies. A server can always decide whether to answer a client request or not. If servers are replaced by peers in an open Peer-to-Peer system where all nodes are equal, security cannot be assured as easily anymore. Here new ways for providing similar security have to be found and applied.

Maintenance

The maintenance of distributed systems is a complex task. Security issues, the optimal usage and availability of distributed resources and services, and software deployment become even more complex in a heterogenous combination of Web services and Peer-to-Peer technology.

14.5 Resulting Architectures

Several architectures can be imagined when joining Web services and Peer-to-Peer technologies. One of the most promising can be outlined as follows.

Distributed applications will have two faces: Peer-to-Peer in a closed and rather secure system (i.e. the Intranet or a similar form) and additional Web service access points for external communication on the Internet – as long as security is weak there. It is possible to have the benefits of Peer-to-Peer systems like decentralization, scalability, and availability inside an application, inside a complex system, or inside a company. On the edge to the Internet this is changed to the benefits of Web services like security and standardized WSDL interface descriptions.

This approach could be used to design service brokers (i.e. the entities responsible for finding a service matching a request like in [211]) and search engines (i.e. entities responsible for finding arbitrary information matching a request like in [296]) by using Peer-to-Peer technology internally and offering their results in XML/WSDL.

Further Reading

This chapter about Web services and Peer-to-Peer was only a short introduction into the world of distributed services. A good start for obtaining more knowledge are the following references (in no particular order) [25, 291, 193, 496, 141].