Distributed and Parallel Databases, 12, 135–162, 2002
© 2002 Kluwer Academic Publishers. Manufactured in The Netherlands.

An Overview of Standards and Related Technology in Web Services

APHRODITE TSALGATIDOU* afrodite@di.uoa.gr
THOMI PILIOURA† thomi@di.uoa.gr; tpilioura@nbg.gr
University of Athens, Department of Informatics & Telecommunications, TYPA Buildings, Panepistimiopolis, Ilisia, GR-157 84, Athens, Greece

Recommended by: Boualem Benatallah and Fabio Casati

Abstract. The Internet is revolutionizing business by providing an affordable and efficient way to link companies with their partners as well as customers. Nevertheless, there are problems that degrade the profitability of the Internet: closed markets that cannot use each other's services; incompatible applications and frameworks that cannot interoperate or built upon each other; difficulties in exchanging business data. Web Services is a new paradigm for e-business that is expected to change the way business applications are developed and interoperate. A Web Service is a self-describing, self-contained, modular application accessible over the web. It exposes an XML interface, it is registered and can be located through a Web Service registry. Finally, it communicates with other services using XML messages over standard Web protocols. This paper presents the Web Service model and gives an overview of existing standards. It then sketches the Web Service life-cycle, discusses related technical challenges and how they are addressed by current standards, commercial products and research efforts. Finally it gives some concluding remarks regarding the state of the art of Web Services.

Keywords: web service, overview, standards, technical challenges

1. Introduction

The growth of Internet technologies has unleashed a wave of innovations that change the way business is conducted. Companies are moving their main operations to the web for more automation, efficient business processes and global visibility. In order to survive the competition created by this new online economy, companies should choose and implement the right software and technology solution. They should find an integrated, robust e-business solution that allows them to leverage existing applications, rapidly adapt to the unique needs of their business and continually evolve as business requirements change over time.

Nowadays, the current trend in the application space is moving away from tightly coupled systems (e.g. DCOM based business solutions [9]) and towards systems of loosely coupled, dynamically bound components (e.g. Jini [21] or Enterprise Java Beans (EJB) [12]). The latest evolution in this new category of systems is a new paradigm, called Web Services. Web Services are self-contained, modular applications, accessible via the Web, that provide a set of functionalities to businesses or individuals. What makes the Web Service vision

*To whom all correspondence should be addressed.
†Ms. Thomi Pilioura is also with the National Bank of Greece.

attractive is the ability to discover the Web Services that fulfil users' needs, negotiate service contracts and have the services delivered where and when the users need them.

The Web Service paradigm is the logical evolution from object-oriented systems to systems of services. As in object-oriented systems, some of the fundamental concepts in Web Services are encapsulation, message passing and dynamic binding. However, the service-based paradigm is extended beyond method signatures, as information related to what the service does, where it is located, how it is invoked and the quality of service and security policy related to this service can also be published in the service interface. The Web Service approach can be also considered as the evolution of the component paradigm. Web Services are lightweight, loosely coupled, platform and language independent components.

The goal of this paper is to examine the Web Service paradigm and the technical challenges brought forward, to analyse the contribution of standards and related technology and to identify open issues. It is therefore organized as follows. Section 2 presents the Web Service model and its potential benefits as compared to today's applications and gives an example of a Web Service based application to be used throughout the paper. Section 3 gives an overview of the key standards in the area of Web Services using the example Web Service application. Section 4 describes the Web Service life-cycle, the technical challenges brought forward and the way these are addressed by current standards, commercial products and research efforts. It then summarizes the contribution of current technology and identifies remaining open issues. Finally, Section 6 presents our concluding remarks.

2. The web service concepts

Web Services constitute a new model for using the Web. This model allows the publishing of business functions to the Web and enables universal access to these functions. Both developers and end-users can enjoy the benefits of web services. The Web Service model simplifies business application development and interoperation. Additionally, it greatly serves end-user needs by enabling them—through an intuitive, browser-based interface—to choose, configure and assemble their own web services. The following paragraphs present this model in detail along with the benefits that it could bring to e-business.

2.1. The web service model

Any service-oriented environment is expected to support several basic activities:

1. Web Service creation
2. Web Service description
3. Web Service publishing to Intranet or Internet repositories for potential users to locate
4. Web Service discovery by potential users
5. Web Service invocation, binding
6. Web Service unpublishing in case it is no longer available or needed, or in case it has to be updated to satisfy new requirements.

In addition to these basic activities there are some other activities that need to take place in order to take full advantage of the Web Service architecture. Such activities include

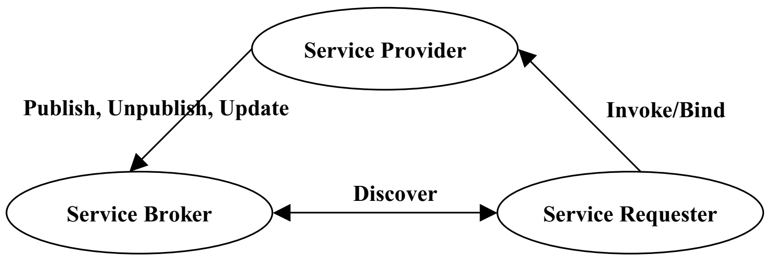

Figure 1. Web service model.

Web Service composition, management and monitoring, billing and security. However, we consider that the Web Service model (figure 1) requires at least the following basic activities: describe, publish/unpublish/update, discover and invoke/bind, and contains 3 roles: service provider, service requester and service broker [33].

Service provider. A service provider is the party that provides software applications for specific needs as services. Service providers publish, unpublish and update their services so that they are available on the Internet. From a business perspective, this is the owner of the service. From an architectural perspective, this is the platform that holds the implementation of the service.

Service requester. A requester is the party that has a need that can be fulfilled by a service available on the Internet. From a business perspective, this is the business that requires certain function to be fulfilled. From an architectural perspective, this is the application that is looking for and invoking a service. A requester could be a human user accessing the service through a desktop or a wireless browser; it could be an application program; or it could be another Web Service. A requester finds the required services via a service broker and binds to services via the service provider.

Service broker. This party provides a searchable repository of service descriptions where service providers publish their services and service requesters find services and obtain binding information for these services. It is like telephone yellow pages. Such examples of service brokers are UDDI (Universal Description, Discovery, Integration) [19, 24] and Xmethods [36].

It is clear that since the service provider, the service broker and the service requester interact with each other they should use standard technologies for service description, communication and data formats. This reliance on standards allows developers to implement Web Services in a language and platform independent way. A description of the most commonly used standards follows in Section 3.

2.2. *A web service example*

For an enterprise there are 3 scenarios that involve implementing/integrating Web Services [20]:

- Implement a new web service. The developer starts form scratch creating not only the Web Service description but also the functionality exposed as a Web Service.
- Expose existing functionality via a web service. This scenario follows along the same lines as the previous with the exception that the functionality being exposed as a Web Service already exists (for example a Java class, EJB or a COM object).
- Integrate Web Services from other vendors or business partners. In several situations, it makes sense to buy a Web Service rather than build a component from scratch. In these situations an enterprise would need to integrate other Web Services to complete/enhance its applications. Alternatively the enterprise may have business partners who have exposed functionality as Web Services, which the enterprise can use.

To illustrate all these scenarios, consider how Web Services could be used in an application offering real-time stock market information. Suppose an enterprise would like to build a portal offering the following services:

- *Real-time view of the stock market.* Number of stocks going down and number of stocks going up as well as a figure illustrating the fluctuation of the general index. It's not necessary to build this service from scratch. Instead the enterprise can buy it from a data vendor.
- *Stock quote.* A service that enables investors to retrieve a quote in EURO given the ticker-symbol of any publicly traded stock. The enterprise may decide to build a Web Service providing this functionality.
- *Account balance check.* A service that checks the investor's account balance in order to make sure that s/he has the necessary amount for the completion of the transaction. Suppose there is already a component with this functionality. In this case, the enterprise exposes it as a Web Service.
- *Portfolio management.* Allows an investor to track the performance of his/her shares and to make transactions (buy or sell shares). The appropriate Web Service can be composed by combining the two previous services and by adding the necessary functionality for the orchestration of these services.
- *News.* Provides the headlines of the latest financial news. A new Web Service may be built to offer this functionality.

This portal is intended to be available to everybody with access to the Internet. All Web Services are coordinated by the portal application.

2.3. Potential advantages of web services

The following paragraphs focus on the benefits of Web Services as compared to today's applications.

Easy and fast deployment. Enterprises using the Web Service model can provide new services and products without the investment and delays a traditional enterprise requires. They may develop new Web Services by reusing and/or combining existing ones. For example the portal Web Service provides a set of high-level features by orchestrating lower-level Web Services for portfolio management, stock quote and others.

Interoperability. Any Web Service can interact with other Web Services. This is achieved through an XML-based interface definition language and a protocol of collaboration and negotiation. By limiting what is absolutely required for interoperability, collaborating Web Services can be truly platform and language independent. This means that developers do not need to change their development environments in order to produce or consume Web Services. Furthermore by allowing legacy applications to be exposed as services, the Web Services architecture easily enables interoperability between legacy applications or between Web Services and legacy applications.

Just-in-time integration. Traditional system architectures incorporate relatively brittle coupling between various components in the system. These systems are sensitive to change. A change in the output of one of the subsystems or a new implementation of a subsystem will often cause old, statically bound collaborations to break down. Web Services systems promote significant decoupling and just-in-time integration of new applications and services, as they are based on the notion of building applications by discovering and orchestrating network-available services. This in turn yields systems that are self-configuring, adaptive and robust with fewer single points of failure.

Reduced complexity by encapsulation. All components are services. What is important is the type of behaviour a service provides, not how it is implemented. This reduces system complexity, as application designers do not have to worry about implementation details of the services they are invoking.

3. Web service related standards

Today there is a lot of activity in the Web Service area. We are currently witnessing the rapid development and maturation of a stack of interrelated standards that are defining the Web Service infrastructure along with a great number of development tools that support the Web Service development. Two of the many possible choices one can make for describing, advertising, discovering and binding Web Services in a decentralized, distributed service-oriented environment are the key standards of WSDL [34], UDDI [31], SOAP (Simple Object Access Protocol) [29] and the ebXML initiative [10]. The interoperation of the first three standards supports the basic activities of a service-oriented environment, namely publish, unpublish, update, find and invoke, while ebXML offers another solution for supporting these activities.

3.1. The web services description language (WSDL)

For an application to use a Web Service, the programmatic interface of the Web Service must be precisely described. In this sense, WSDL plays a role analogous to Interface Definition Language (IDL) [26] used in distributed programming. It is an XML grammar for specifying properties of a Web Service such as *what* it does, *where* it is located and *how* it is invoked.

A WSDL document defines *services* as collections of network endpoints, or *ports*. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: *messages*, which are abstract descriptions of the data being exchanged, and *port types* that are abstract collections of *operations*. The concrete protocol and data format specifications for a particular port type constitute a reusable *binding*. A port is defined by associating a network address with a reusable binding, and a collection of ports defines a service. Hence, a WSDL document uses the following seven elements in the definition of network services:

- *Type.* A container for data type definitions using some type system (such as XSD).
- *Message.* An abstract, typed definition of the data being communicated.
- *Operation.* An abstract description of an action supported by the service.
- *Port Type.* An abstract set of operations supported by one or more endpoints.
- *Binding.* A concrete protocol and data format specification for a particular port type.
- *Port.* A single endpoint defined as a combination of a binding and a network address.
- *Service.* A collection of related endpoints.

In our portal example, each Web Service is described by a WSDL document, which contains all the previous elements. For instance, the following code excerpt is a WSDL document for the "*stock quote*" service. The most important parts of the document are shown in boldface type. This document defines that the type of the request message is string and the type of the reply is float, the name of the operation that must be invoked is "GetLastTradePrice", HTTP is used to carry the messages which are formatted in SOAP and the network address of the service is http://www.stockquoteserver.com/StockQuote.

```
<types>
  <schema targetNamespace="http://example.com/stockquote.xsd"
    xmlns="http://www.w3.org/1999/XMLSchema">
      <element name="GetLastTradePrice">
        <complexType>
          <all>
            <element name="symbol" type="string"/>
          </all>
        </complexType>
      </element>
      <element name="GetLastTradePriceResponse">
        <complexType>
          <all>
            <element name="Price" type="float"/>
          </all>
        </complexType>
      </element>
  </schema>
</types>
```

```
<message name=“GetLastTradePriceInput”>
    <part name=“body” element=“xsd1:GetLastTradePrice”/>
</message>

<message name=“GetLastTradePriceOutput”>
    <part name=“body” element=“xsd1:GetLastTradePriceResponse”/>
</message>

<portType name=“StockQuotePortType”>
    <operation name=“GetLastTradePrice”>
      <input message=“tns:GetLastTradePriceInput”/>
      <output message=“tns:GetLastTradePriceOutput”/>
    </operation>
</portType>

<binding name=“StockQuoteSoapBinding” type=“tns:StockQuotePortType”>
    <soap:binding style=“document” transport=“http://schemas.xmlsoap.org/
        soap/http”/>
    <operation name=“GetLastTradePrice”>
      <soap:operation soapAction=“http://example.com/stockquote.xsd”/>
      <input>
          <soap:body use=“encoded” namespace=“http://example.com/stockquote.xsd”
              encodingStyle=“http://schemas.xmlsoap.org/soap/encoding/”/>
      </input>
      <output>
          <soap:body use=“encoded” namespace=“http://example.com/stockquote.xsd”
              encodingStyle=“http://schemas.xmlsoap.org/soap/encoding/”/>
      </output>
    </operation>
</binding>

<service name=“StockQuoteService”>
    <documentation>My first service</documentation>
    <port name=“StockQuotePort” binding=“tns:StockQuoteBinding”>
      <soap:address location=“http://www.stockquoteserver.com/StockQuote”/>
    </port>
</service>
```

There is also an alternative authoring style for WSDL documents. This style makes use of the *import* element, which allows the separation of the elements of the service description into two parts, referred to as “service interface definition” and “service implementation definition.” Typically, information common to a certain category of business services, such as message formats, portTypes and protocol bindings, are included in the reusable portion, while information pertaining to a particular port definition is included in the service implementation definition portion.

3.2. The simple object access protocol (SOAP)

SOAP is a standard for sending messages and making remote procedure calls over the Internet. It is independent of the programming language, object model, operating system and platform. It uses HTTP as the transport protocol and XML for data encoding. However, other transport protocols may also be used such as FTP, SMTP or even raw TCP/IP sockets.

SOAP defines two types of messages, Request and Response, to allow service requesters to request a remote procedure and to allow service providers to respond to such requests. A SOAP message consists of two parts, a header and the XML payload. The header differs between transport layers, but the XML payload remains the same. The XML part of the SOAP request consists of three main portions:

- The *Envelope* defines the various namespaces that are used by the rest of the SOAP message.
- The *Header* is an optional element for carrying auxiliary information for authentication, transactions and payments. Any element in a SOAP processing chain can add or delete items from the Header; elements can also choose to ignore items if they are unknown. If a Header is present, it must be the first child of the Envelope.
- The *Body* is the main payload of the message. When SOAP is used to perform an RPC call, the Body contains a single element that contains the method name, arguments and Web Service target address. If a Header is present, the Body must be its immediate sibling; otherwise it must be the first child of the Envelope.

A SOAP response is returned as an XML document within a standard HTTP reply. The XML document is structured just like the request except that the Body contains the encoded method result.

The following SOAP request sent over HTTP to the "*stock quote*" service specifies in the body that the request is being sent to the URL http://www.stockquoteserver.com/StockQuote. The request also specifies that the GetLastTradePrice operation is to be performed using the stock ticker symbol MS, which means get the last trading price for the stock of Microsoft.

```
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <m:GetLastTradePrice xmlns:m="http://example.com/stockquote.xsd">
            <symbol>MS</symbol>
        </m:GetLastTradePrice>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The SOAP reply contains the MS price of 143 and looks like this:

```
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    <SOAP-ENV:Body>
      <m:GetLastTradePriceResponse mlns:m="http://example.com/stockquote.xsd">
          <Price>143</Price>
      </m:GetLastTradePriceResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

3.3. Universal description, discovery, integration (UDDI)

UDDI defines a common means to publish information about businesses and services. It can be used to check whether a given partner offers a particular Web Service, to find companies in a given industry with a given type of service and to locate information about how a partner or intended partner has exposed a Web Service in order to learn the technical details required to interact with that service. The UDDI specifications consist of an XML schema for SOAP messages and a description of the UDDI APIs specification.

The UDDI XML schema defines four key data structures: *business entities*, *business services*, *binding templates* and *tModels*. Business entities describe information about businesses, including their name, description, services offered and contact information. Business services provide more detail on each service being offered. Each service can have multiple binding templates, each describing a technical entry point for the service (e.g., mailto, http, ftp, etc.). Finally, tModels describe what particular specifications or standards a service uses. With this information, a business can locate other services that are compatible with its own systems. UDDI also provides identifiers and categories to mark each entity using various taxonomies (related industry, products or services offered and geographical region).

The UDDI APIs contain messages for interacting with UDDI registries. Inquiry APIs are provided to locate businesses, services, bindings or tModels. Publishing APIs are included for creating and deleting UDDI data in the registry. The UDDI APIs are based on SOAP.

A UDDI Business Registry is itself a SOAP Web Service. IBM and Microsoft are implementing UDDI Registries. Service providers will only have to register at one of the implementations since updates to any are replicated in the others on a daily basis. Each of the business registries provides operations to create, modify, delete and query each of the four data structures. All these operations can be performed either via a Web site or by using tools that make use of the UDDI API specification.

For instance in our stock market example the business registration for the company "GetQuote Company" can be retrieved from the UDDI registry using the following SOAP message:

```
< Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
< Body>
  <find_business xmlns="urn:uddi-org:api" generic="1.0">
   <name>GetQuote Company</name>
  </find_business>
</Body>
</Envelope>
```

The reply to this query could be that the "GetQuote Company" offers the "*stock quote*" and the "Portfolio Management" services and will look like this:

```
<businessInfo businessKey="0076B468-EB27-42E5-AC09-9955CFF462A3">
    <name>GetQuote Company</name>
    <description xml:lang="en">Features portoflio managers, stock quote and other stock
       market related services</description>
  <serviceInfos>
     <serviceInfo businessKey="0076B468-42E5-AC09-9955CFF462A3"
              serviceKey="1FFE1F71-B788-09AF7FF151A4">
       <name>Stock Quote</name>
     </serviceInfo>
     <serviceInfo businessKey="0076B468-EB27-42E5-AC09-9955CFF462A3"
              serviceKey="8BF2F51F-8ED4-43FE-B665-38D8205D1333">
       <name>Portfolio Management</name>
     </serviceInfo>
  </serviceInfos>
</businessInfo>
```

3.4. Interoperation of SOAP, WSDL and UDDI

After having examined each standard, we will now view them from a use-case perspective in order to grasp the round-trip processing that takes place in a Web Service environment. UDDI plays an important role in the interoperation of the three standards. It can be used to perform two different sets of activities:

1. Create (figure 2, steps 1 and 2), view, update and delete business or service registrations. The participants of this kind of activities are the service provider and the service broker. The service provider can perform these actions through the exchange of UDDI SOAP messages with the service broker. This can be achieved either via a Web interface or another tool that uses the UDDI APIs.
2. Search the registry for business or service registrations. All 3 roles of a Web Service environment participate in this activity. This activity can take place both at development time and at run time. At development time, the registry can be searched (figure 2, steps 3 and 4) by a programmer for suitable services and can be used to locate the appropriate WSDL file (figure 2, steps 5 and 6). A WSDL document may reference other WSDL

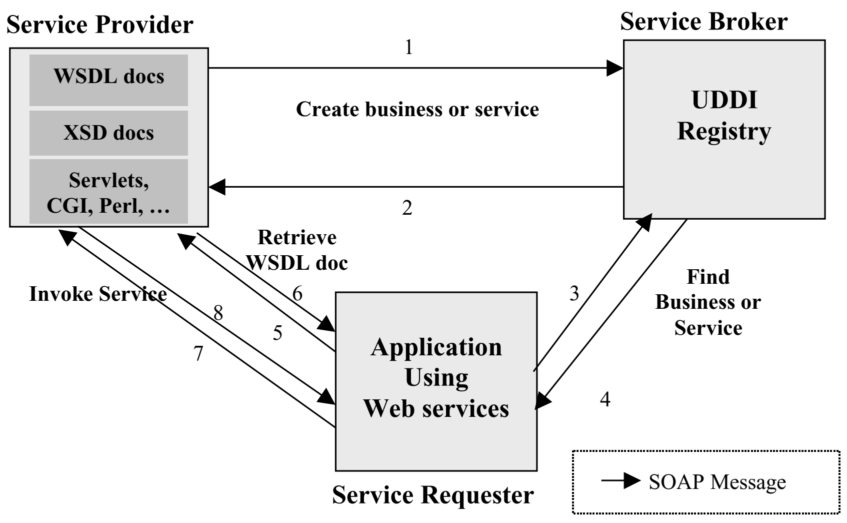

Figure 2. How SOAP, WSDL and UDDI are related.

documents and XML Schema (XSD) documents that describe data types used by Web Services. The WSDL document is stored on the service provider. Alternatively, it may be stored in special XML repositories; in this case, better performance is achieved as the network overhead for the service provider is reduced, since the latter hosts only the executable service (servlets, CGI scripts, Perl scripts, etc.).

After the programmer has studied the specifications for the Web Service described in the retrieved WSDL documents, s/he generates client proxies so that the application can access the service (figure 2, steps 7 and 8). Alternatively development tools can be used to generate the required client proxies.

At run time, the application does not have to use UDDI if the target Web Service is always available and always the same. However, if the Web Service becomes unavailable, the application can query the UDDI registry to determine if the service has been moved to another location. This capability is useful when service providers have to rehost their services. Furthermore, UDDI enables the selection of the appropriate service at run time. In our example, all stock market agencies offering the "*News*" Web Service could be located and queried by the portal application at runtime to find the latest financial news.

We can see that with the use of SOAP, UDDI and WSDL the integration and interoperability problem has been simplified in layers (figure 3). XML provides a cross-platform approach to data encoding and formatting. SOAP, which is built on XML, defines a simple way to package information for exchange across system boundaries. SOAP bindings for HTTP are built on this packaging protocol and define a way to make remote procedure calls between systems in a manner that is independent of the programming language or operating system choices made by individual companies. WSDL is an XML grammar for specifying properties of a Web Service. The UDDI specifications define a next-layer-up that allows two

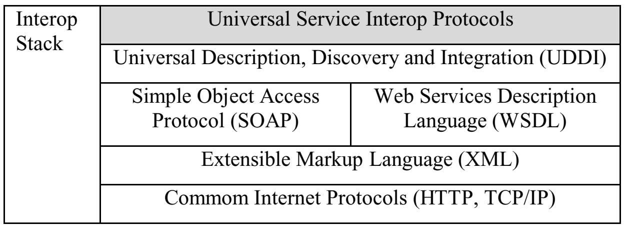
Interop Stack
Universal Service Interop Protocols
Universal Description, Discovery and Integration (UDDI)
Simple Object Access Protocol (SOAP)
Web Services Description Language (WSDL)
Extensible Markup Language (XML)
Commom Internet Protocols (HTTP, TCP/IP)

Figure 3. Layered view of SOAP, WSDL and UDDI standards.

companies share a way to query each other's services and to describe their own services. Universal Service Interop Protocols will be defined in the future on top of UDDI which will offer more advanced discovery features, such as the ability to locate parties that can provide a specific product or service at a given price or within a specific geographic boundary in a given timeframe.

3.5. ebXML

ebXML is a global electronic business standard that is sponsored by UN/CEFACT and OASIS. It defines a framework that allows businesses to find each other and conduct business based on well-defined XML messages within the context of standard business processes that are governed by standard or mutually-negotiated partner agreements.

The ebXML technical architecture provides: (1) Business Process and Information Models, (2) Company Profiles, (3) Messaging services, (4) Registry and Repository, (5) Collaborative Protocol Profiles (CPP) and (6) Transactional Support.

The Business Process models define how business processes are described. Business Processes represent the "verbs" of electronic business and can be represented using modelling tools. The specification for business process definition enables an organization to express its business processes so that they are understandable by other organizations. It identifies such things as the overall business process, the roles, transactions, identification of the business documents used, document flow, legal aspects and security aspects. This enables the integration of business processes within a company or between companies. The Information models define reusable components that can be applied in a standard way within a business context. These Core Components represent the "nouns and adjectives" of electronic business. They are defined using identity items that are common across all businesses. This enables users to define data that is meaningful to their business while also maintaining interoperability with other business applications.

ebXML also provides support for Collaboration Protocol Profile (CPP) documents. The CPP contains essential information about the provider including, but not limited to: contact information, industry classification, supported business processes, Interface requirements and Messaging requirements. CPP may also contain security and other implementation specific details, error handling and failure scenarios.

The ebXML Messaging Service specification enables interoperable, secure and reliable exchange of payload (which may or may not be an XML business document) between two organizations and between an organization and the registry. The specification allows any application-level protocol to be used. These can include common protocols such as SMTP, HTTP and FTP. Well-established cryptographic techniques can be used to implement strong security. For example, secure protocols such as HTTPS can be used to guarantee confidentiality. In addition, digital signatures can be applied to individual messages or a group of related messages to guarantee authenticity.

ebXML provides a registry similar to UDDI for discovering services. Technically speaking, a registry stores information about items that actually reside in a repository. The two together can be thought of as a database. Items in the repository are created, updated or deleted through requests made to the registry. The ebXML Registry has schema documents, business process documents and CPP documents.

ebXML additionally provides a negotiation framework through IBM's TpaML (Trading Partner Markup Language) [30]. A Collaboration Protocol Agreement (CPA), being a reflection of the CPPs of both partners, is created during an engagement between the two parties. It contains information that outlines the service and the process requirements agreed upon by all parties.

Furthermore ebXML supports transactions. Execution of a process is made up of transactions. A transaction is an atomic unit of work between trading partners, which is simply an exchange of documents, such as a purchase order. A transaction can succeed or fail. If it succeeds, it is legally binding. If it fails, it is null and void and the transaction is rolled back. Transaction semantics can specify a number of parameters such as required security, reliability and timeliness.

Figure 4 depicts a sequence of steps to establish a simple business transaction interchange between two Trading Partners using ebXML Applications and related Components. In the following we will briefly describe the steps that take place during such a business transaction interchange.

Initial phase. In figure 4, Company A has become aware of an *ebXML Registry* that contains a set of ebXML Specifications. Company A requests an ebXML specification in order to determine if it wants to become an ebXML compliant participant (figure 4, step 1). The request results in the ebXML process specification being sent to Company A (figure 4, step 2). Company A, after reviewing the specification, decides to build and deploy its own ebXML compliant application (figure 4, step 3). Company A then submits its own implementation details, reference links, and Collaboration Protocol Profile (CPP) as a request to the ebXML Registry (figure 4, step 4). The CPP submitted describes the company's ebXML capabilities and constraints, as well as its supported business scenarios (XML versions of the business processes). After receiving verification that the format and usage of a business object is correct, an acknowledgment is sent to Company A by the ebXML Registry (figure 4, step 5). At any time, Company A is free to access its own profile, review and make changes as necessary.

Discovery of partner and negotiation phase. Company B is then informed by Company A that they would like to engage in a business transaction using ebXML. Subsequently, the ebXML Application queries the ebXML Registry about Company A (figure 4, step 6). Company A's profile is retrieved (figure 4, step 7). Based on the CPP, the Application

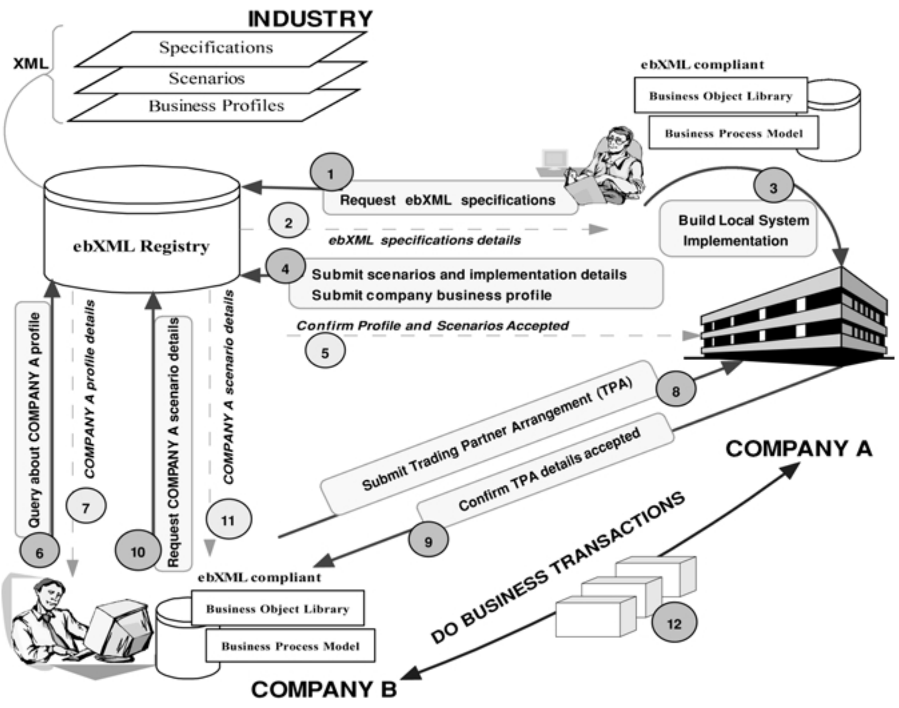
INDUSTRY
XML
Specifications
Scenarios
Business Profiles
ebXML compliant
Business Object Library
Business Process Model
1
Request ebXML specifications
2
ebXML specifications details
3
Build Local System Implementation
ebXML Registry
4
Submit scenarios and implementation details
Submit company business profile
Confirm Profile and Scenarios Accepted
5
Query about COMPANY A profile
6
COMPANY A profile details
7
Request COMPANY A scenario details
10
COMPANY A scenario details
11
Submit Trading Partner Arrangement (TPA)
8
Confirm TPA details accepted
9
COMPANY A
DO BUSINESS TRANSACTIONS
12
ebXML compliant
Business Object Library
Business Process Model
COMPANY B

Figure 4. Conducting e-business using ebXML (Copyright © ebXML 2000).

determines that it is able to execute a specific scenario that Company A supports. Before engaging in that scenario, Company B submits a proposed CPA directly to Company A's ebXML compliant software interface. The CPA (or TPA in figure 4) outlines the E-business scenario and specific arrangement(s) it wants to use with Company A, as well as certain messaging, contingency and security-related requirements (figure 4, step 8). Company A accepts the CPA and acknowledgement is sent directly to Company B's shrink-wrapped ebXML software Application (figure 4, step 9). Since the scenario from Company A was not available in the software package that Company B is using, the Application requests it from the ebXML Registry (figure 4, step 10). The scenario is then provided to Company B's Application (figure 4, step 11).

Transaction phase. Based on the processes (contained in the process models) and information parcels (presented in Class Diagrams) Company A and B are now engaging in eBusiness utilizing ebXML specifications via their respective software Applications (figure 4, step 12).

3.6. ebXML vs UDDI/SOAP/WSDL

The ebXML and UDDI/SOAP/WSDL initiatives are tackling the same problem using two different approaches. ebXML is following a top-down approach—identifying the requirements necessary for successfully conducting e-business over the Internet and then working to implement specifications that meet those requirements. The UDDI/SOAP/WSDL initiative however is following a bottom-up approach—implementing specifications that meet individual core requirements (such as service description, discovery and invocation) and building up from there.

ebXML can be viewed as a complex implementation of the Web Service model. It essentially has a specification to handle every aspect of the electronic business transaction, from describing simple service endpoint definition (using e.g. CPP) to describing the entire business process workflow for the transaction (using e.g. CPA). On the other hand, the scope of the UDDI/SOAP/WSDL initiative is narrower and the implementation is less complex.

More specifically, the description of a Web Service is realised by using WSDL in the UDDI/SOAP/WSDL initiative and CPP in the ebXML specification. WSDL provides information about a service name, parameters for that service and the endpoint to invoke it. In addition to this information, CPP provides other important parameters, such as security details, error-handling and failure scenarios.

UDDI is used to publish and discover Web Services in the UDDI/SOAP/WSDL initiative; the ebXML Registry Service, on the other hand, not only publishes and discovers Web Services, but also provides information about, for instance, business processes, business documents and business profiles. However, both systems can be used complementary; organizations can continue to use UDDI to inquire about businesses in the global UDDI Registry. Those entries can then be used in referring to ebXML Web Services in the ebXML Registry.

In the UDDI/SOAP/WSDL initiative, once the WSDL for a particular Web Service is developed, the service can then be invoked using SOAP. In ebXML, on the other hand, the service is invoked using the ebXML Messaging Service, which provides a uniform way of sending messages. Work is under way to make these standards interoperate. Along these lines, OASIS and UN/CEFACT reached an agreement with IBM and Microsoft to incorporate SOAP into ebXML [10, 29]. Thus, the ebXML Messaging Service can make sure that the CPA governs any business transactions and can now provide a secure and reliable transport infrastructure based on SOAP and HTTP.

4. Technical challenges

The Web Service paradigm brings forward a number of challenges. We believe that its success depends on the degree that these challenges are met by the key standards outlined so far and by the other emerging products and research efforts that also address the Web Service paradigm. In this section we examine these challenges and how they are met by existing solutions and we look at open issues.

The technical challenges brought forward by the Web Service paradigm are related to the Web Service life-cycle, which includes a number of activities (figure 5). It is obvious that some of these activities will take place at the requester's site, while other take place at the broker's or provider's site. The first activity is the Web Service creation that takes place either by building the service from scratch or by composing it using already existing Web Services. The second activity is the Web Service description followed by Web Service publishing in

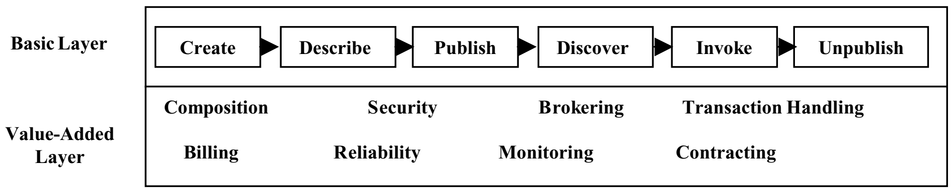

Figure 5. The web service lifecycle.

a registry. The published Web Service may need also to be updated or unpublished during its lifetime. Discovery of a Web Service can be facilitated by the means of a broker that is expected to support requirements analysis and description of user's needs, matching of user needs to offered Web Services, Web Service composition, negotiation and bidding.

After a Web Service has been discovered and has been decided that it matches the user's needs, and before it can be used, a number of activities related to contracting usually need to take place in order to govern the business transaction. Finally, a service can be unpublished if it is no longer available or needed. During the life-cycle of a Web Service the value-added activities of security, reliability, billing, monitoring and transaction handling also need to take place.

Thus, we classify the activities performed during the Web Service life-cycle into two layers: the basic layer and the value-added layer. The basic layer contains the basic activities of Web Service creation, description, publishing, discovery, invocation and unpublishing. These activities are necessary to be supported by every Web Service environment. The value-added layer contains the value-added activities of composition, security, brokering, reliability, billing, monitoring, transaction handling and contracting. These activities bring value-added functionality and better performance to any Web Service environment.

All activities either basic or value-added are applied either to a simple or to a composite service (a synthesis of simple services) and are expected to expose their functionality at two different levels:

- At a syntactic or implementation level (low abstraction level), mainly concerned with the syntactic or implementation aspects of a Web Service and thus tailored towards the programmers' requirements.
- At a semantic level (high abstraction level), i.e. at a level where the main focus is on the semantic or conceptual aspects of Web Services aiming to facilitate the work of business users by shielding off the lower level technical details.

In the following we examine the technical challenges related to Web Service basic and value-added activities and we examine the level of abstraction targeted by existing solutions.

4.1. Technical challenges for basic activities

4.1.1. Description. The description of a Web Service is essential for classifying, discovering and using a service. It needs to be understandable by humans as well as by machines and to contain functional (e.g. what a service does) as well as non-functional requirements (e.g. security, authentication and privacy issues related to the information exchange). A Web Service description is required to be at both semantic and syntactic level.

Semantic information must contain details about the service provider, description of what the service does and its characteristics in terms of reliability, security, sequencing of messages, etc. The semantic information enables service requesters to decide whether a service satisfies their needs or not. Also, brokers can use the semantic information to categorize the service. In our stock market example, the semantic information for the "*stock quote*" service may specify that the maximum time for a conversation between the service and the requester is 3 minutes, the cost of using the service is 5$ and the way of payment is by credit card.

Syntactic information describes how to use the service and may also concern non-functional requirements, e.g. security, by specifying for example the exact security certificates required or supported by a Web Service.

WSDL, analyzed in Section 3, describes only the syntactic aspects of a service. Essentially, a WSDL document describes how to invoke a service and provides information on the data being exchanged, the sequence of messages for an operation and the location of the service. WSDL is not able to sufficiently capture the semantics of the business domain and the characteristics of the service in terms of reliability, billing, etc. The ebXML, as described in Section 3.5, provides both semantic and syntactic information support for a Web Service via the CPP documents.

Apart from the examined standards, there are some other products and research activities addressing Web Service description. The Advertisement and Discovery of Services (ADS) protocol specification [1] supports the following three classes of service descriptions: the description of a simple service, the description of a collection of services and the description of a repository of services. ADS uses the service description architecture of WSDL. Therefore, it describes the syntactic aspects of a service. In HP Web Services Platform [17], a service is accompanied with specification and descriptions. A service specification is composed of interfaces, security policies and a filter constraint. An interface specification describes how clients interact with the service. A security policy prescribes who is granted which access right for a particular service. Service providers may specify those who may discover the service using the filter constraint. The service description consists of attributes specific to that service. The more attributes registered for Web Services, the richer users' requests for services can be. Technically speaking, the attributes are meta-data expressed in XML. They have to conform to a specific agreed-upon vocabulary developed for that type of service. A service may have multiple descriptions in different vocabularies. Service providers may create their own vocabularies or select a Microsoft BizTalk vocabulary or use a combination of the two. Thus, HP Web Services Platform supports semantic as well as syntactic description of a Web Service. The Service Description Language (SDL) described in [32], still at early development stages, promises to address Web Service description at both lower and higher level. It offers constructs to describe service properties, service contents, service ontologies, cost, payment, actors using the service, authorization and service capability. However, SDL does not specify how actors can be defined if Web Services are enacted through a web interface. Also, the revealing of service contents violates encapsulation.

It is obvious, that the need for a Web Service description at both semantic and syntactic level has been realised and addressed in some ways by ebXML and by other products and research efforts. However, what has not yet been addressed by current efforts is the analysis and description of Web Service related user requirements. This is essential for service discovery and therefore we revisit it in the sub-section related to service discovery.

4.1.2. Publishing. Publishing is one of the basic activities as it makes a service known and available to be used. Publishing just like description needs to be at both syntactic and semantic level. The offered solutions include the ones provided by UDDI Registry and the ebXML Registry and Repository. The UDDI Registry contains business information, service information, binding information and information about specifications for services, thus providing both semantic and syntactic Web Service related information. See for example in Section 3.3 the business registration for the company "GetQuote Company" that offers the "*stock quote*" service. The ebXML Registry has schema documents, business process documents and CPP documents thus providing semantic information, too.

Other solutions include the HP Web Services Platform and eFlow [16]. The HP Web Services Platform provides mechanisms for registering semantic and syntactic information about services. eFlow provides a repository of processes, service nodes and service data type definitions. The repository is organized into a hierarchy dividing the former into groups and simplifying its browsing.

Given the existence of these registries, the Advertisement and Discovery of Services Protocol (ADS) mentioned above, provides two mechanisms that facilitate the building of a crawler to pull Web Service advertisements off the Web, without people having to push advertisements for their services to the registry. More specifically, the ADS mechanisms allow service providers to announce the availability of service descriptions by the creation of a file placed in the root directory of their server and by the use of a meta tag that can be included in any HTML file to denote the location of the service description.

It is clear that, the effective publishing of services depends on effective categorization that in turn depends on effective information provided in service descriptions and on appropriate taxonomies built by brokers. Services may be registered in multiple categories, provided that they function correctly in each of these categories. Categorization of services is not an easy task and depends on both service provider and broker. The broker is mainly responsible for the offered taxonomies while the service provider is responsible for classifying the service into the appropriate taxonomy unless the latter prefers to give this task to the service broker. In any case, nothing can guarantee that the Web Service classification is the most effective. Furthermore, the taxonomies provided by service brokers may not be sufficient for effective service categorization. Service brokers may compete on the merits of their choice of taxonomy, on the up-to-date accuracy of their listings and on auxiliary information, such as quality-of-service data, statistical information for the use of the service and comments/evaluation results by service users. The latter can be an important factor for building trust on a specific service.

4.1.3. Discovery. Discovery depends highly on service description, categorization and publishing as well as on the analysis and description of user needs, besides searching and matching techniques. It is a basic activity that has to be supported at a semantic level, as it is important for requesters to describe their needs at a conceptual level and to make sure that their needs are matched with what the service provides. For example, in our stock market portal, a requestor may look for all parties that are Financial Institutions, play a provider role and provide history information about the stock market.

UDDI contains semantic information about companies and syntactic information about services. However, the discovery mechanisms are not so elaborated. ebXML supports discovery at a high-level by allowing discovering of CPP supporting specific scenarios.

Other approaches to Web Service discovery include the HP Web Services platform and the Service Request Language (SRL) [32]. The HP Web Services platform attempts to address the discovery issue at a conceptual level by allowing users to express their service request as a collection of attribute values. Then, it automatically discovers registered services that have the desired attributes. We believe that a higher level for specifying the user needs would be more helpful rather than a mere list of attributes. Another approach is provided by SRL, which is a declarative language for querying and requesting a Web Service described using the service description language (SDL) mentioned in the description section above. This language is able to query the system at two levels: metadata level (service location and semantic exploration) and service level (connect to the actual services and use their operations). SRL is still at the early stages of development and therefore it is not yet proved how effectively supports discovery.

Since the activities of description, publishing and discovery are interdependent, our observations here are similar to the observations in description and publishing. In other words, although some research efforts and products tackle somehow the discovery issue at a semantic level, they don't provide any support for effective capturing and expressing business needs. We believe that the latter, along with appropriate categorization of Web Services are fundamental issues for service discovery.

4.1.4. Binding and invocation. Binding refers to when the collaboration between services is established. This function must support semantic information such as expected Quality of Service (best-effort, at-least-once, etc), message ordering, delivery time constraints, priorities, etc. For example, in our stock market portal example, it can be specified that the result of the "*stock quote*" service is required to be returned within 2 minutes and that this request has a high priority. Binding can be (1) at design time, (2) dynamic but due to a static choice of collaborator and (3) fully dynamic. In the first case, the application (developer) knows exactly with which service it will interact and how. In the second case, the designer has encoded a specific query in the application to pass to the broker with respect to a precise collaborating service (static choice). However, the details of interaction, e.g. the choice of transport protocol or authentication, are defined when the broker returns the service description at run time. Finally, in the last case, the application knows the semantics and the APIs of the service to be used, but queries a broker with a search pattern that allows a set of alternative service providers to be returned and chooses from this list at runtime [4]. At the moment, most approaches focus on the first two alternatives. However, a fully dynamic model is required for exploiting the benefits of the service-oriented paradigm.

Invocation is mainly addressed at the syntactic level by SOAP. Invocation is also addressed by the ebXML Message Service, which, in contrast to SOAP, provides a Quality of

Service (QoS) framework that supports reliable message delivery, security and transaction semantics. It also, makes sure that the Collaboration Protocol Agreement (CPA) governs any business transactions. In HP Web Services platform, the Service Framework Specification (SFS) is a Document Exchange standard. The SFS is built on top of SOAP. SFS addresses the semantic aspects of invocation by supporting conversations that take advantage of HP Web Services platform security, latency and dynamic destination binding features. Conversations are specified using the conversation definition language CDL. It is the language used to define the business payload in the SFS messages and makes it possible for different service providers and implementers to provide compatible services. Each conversation defines a set of interactions between the participants needed to fulfill a specific purpose or task. It defines the documents exchanged in these interactions and it defines the possible sequences of interactions.

4.2. Technical challenges for value-added activities

4.2.1. Composition. Composition refers to the combination of basic Web Services (possibly offered by different companies) into value-added services that satisfy user needs. For example, suppose we want a service offering stock market information and more precisely the ability to retrieve a quote in EURO given the ticker symbol of a stock and a figure illustrating the fluctuation of the general index. This service can be the outcome of the composition of the "*Real-time view of the stock market*" service and the "*stock quote*" service.

Service composition is addressed by eFlow [5, 6] and by the service model of CMI [27]. eFlow supports the dynamic composition of simple Web Services at a semantic level. It allows configurable composite services, where different customers select the desired subset of features that this composite service can offer. The service model of CMI provides another solution that also enables dynamic composition of Web Services at a semantic level by separating the service interfaces from service implementations and defining the service interfaces in terms of service state machines and its input/output parameters. A further advantage of this is the ability to use multiple implementations for the same service without the need to modify the specification of the running process that invokes the service. Another research effort may be found in [7] where a process model and architecture supporting active flowcharts is proposed for supporting composition of discrete and session-oriented services. The work in [13] also addresses service composition by introducing a framework for declarative composition of business services. We consider service composition as an important issue since it is very likely that the offered services will not satisfy user needs and therefore more research efforts and standardisation work is needed along this dimension.

Composition of services as part of a business process is addressed by the Web Services Flow Language (WSFL) [18], the Web Services Conversation Language (WSCL) [18], XLANG [18] and the Business Process Modelling Language (BPML) [3]. WSFL is an XML language for the description of Web Service compositions as part of a business process definition. WSFL relies on an envisioned "endpoint description language" to describe non-functional characteristics of service endpoints, such as QoS properties. WSCL provides an XML schema for defining legal sequences of documents that Web Services can exchange. WSCL allows defining the abstract interfaces of Web Services, i.e. the business level conversations or public processes supported by a Web Service. XLANG is a notation for the specification of message exchange behaviour among participating web services. BPML is a meta-language for modelling business processes and it can thus be used in composing several Web Services in the same business process definition. BPML is concerned with the technical infrastructure for executing business processes and not modelling from a semantic or economic perspective.

4.2.2. Brokering. Brokering is the general act of mediating between requestors and providers in order to match buyer's needs and suppliers' offerings. It is more complete activity than discovery. A broker must have the capability to enable universal service-to-service interaction, negotiation, bidding and selection of the "best" service. This is heavily dependent on the way the services are modelled and described. If the service description contains semantic information, then a user can construct a query with the attributes of the service s/he wants. For example, s/he can ask for a "*News*" service where the cost is 5$ and the information is updated every 30 minutes. A query can also contain a constraint (condition that services of interest must satisfy), zero or more preferences (used to order the results) and an arbitration policy (how many results are returned). There are preliminary research contributions in the area of service quality and automatic service selection via service brokering in the literature [2, 14, 15, 25]. Brokering is supported by HP Web Services platform. After discovering Web Service providers that can respond to a user's service request, HP Web Services platform negotiates between them to weed out those that offer services outside the criteria of the request. This means that a user will only have to choose between a handful of Web Services instead of hundreds of choices. HP Web Services platform currently offers a simple form of negotiation that matches basic user requests. In the near future, the HP Web Services platform engine will be enhanced with protocols that perform very fine-grained negotiation using complex sets of criteria. Since brokering is central for effective service development, we expect to see in the future more standardization efforts and products to support this area.

4.2.3. Reliability. Reliability is the ability of a system or component to perform its required functions under stated conditions for a specified period of time. For a Web Service we need guarantees that it will be always available, i.e. for 24 hours, 7 days per week, 365 days per year. Some Web Service providers will be more reliable than others. Thus, it is needed to have a way to measure and communicate reliability. It is also needed to specify what happens when a Web Service provider goes off-line temporarily in situations such as temporary outages caused by nightly maintenance or backups. For example, in such cases, an alternative service hosted by a different provider could be located and used, or the application may wait for the original one to return. Furthermore, it is needed to specify what happens when a Web Service goes off-line in the middle of a payment transaction concerning this newly acquired service. Service providers need also to define processes for disaster recovery and migration of all their business partners to a backup system. The provided or requested QoS in general could be expressed in a set of parameters, e.g. throughput, response time, cost. Preliminary research contribution includes the CMI system that implements the CMM model [14]. CMM provides that each service has a set of QoS attributes that provide information on the expected QoS of the service execution. Thus, if there are multiple service providers, the provider that is selected is the one who offers a service that best matches the desired QoS goal. ebXML supports reliable messaging. An acknowledgement must be received for every message transmitted. If no acknowledgement arrives the sending ebXML Message service will either retry delivery or notify the sending application. The number of retries is configurable.

4.2.4. Security. Some Web Services will be publicly available and unsecured, but most business-related services will need to use encrypted communications with authentication. It is likely that HTTP over SSL will provide basic security, but individual services will need a higher level of granularity. It is important to specify how a Web Service authenticates users and how it learns about their security privileges. Proposed standards such as the XML Key Management Specification (XKMS), the XML Signature, the XML encryption and the OASIS Security services [18] seem to be able to be incorporated into Web Services systems. ebXML can sign and encrypt messages. In the HP Web Services platform model, entities are distributed and interact with one another using secure sessions that implement the SLS protocol. We would like to note here that security in Web Services could be essential during other activities in the Web Service life-cycle, e.g. during the Web Service composition. This aspect is tackled in [28] which deals with the specific security concerns introduced by distributed Web Service composition.

4.2.5. Transactions. In traditional distributed systems, the notion of transactions typically involves four properties: Atomicity, Consistency, Isolation, and Durability (ACID properties). The tighter coupling between the various parts of the distributed system makes the problem of transactions solvable to some extent. In the world of Web Services, however, guaranteeing all four properties can be a challenge. Traditional transaction systems use a two-phase commit approach—all of the participating resources are gathered and locked until the entire transaction completes, at which point, the resources are finally released. This approach works fine in a closed environment where transactions are short-lived, but it doesn't work well in an open environment where transactions can span hours or even days. Thus, it is needed to integrate a different kind of transactional mechanism into Web Services. It seems that the proposed standards such as XAML [35] and XLANG provide some solutions in this direction. More specifically, XAML has been designed to support current transaction monitors and includes support for commit, cancel, and retry operations. For example, one can ask for commitment among all Web Services involved in a business transaction. If one fails or refuses to commit, XAML provides the framework for canceling the operation. XAML can ensure transaction integrity and provide a single, complete business transaction to the consumer. XLANG is based on an optimistic model that has been proposed in database research, where actions have explicit compensatory actions that negate the effect of the action. XLANG is a notation for expressing the compensatory actions for any request that needs to be underdone. ebXML supports transactions at both syntactic and semantic level. Transaction semantics can specify a number of parameters such as required security, reliability and timeliness. In our stock market example, if we use the "*Portfolio management*" service to sell shares, we can specify that the time to complete the transaction is 2 days from start of transaction, that no repudiation is required (meaning that the two parties cannot refuse that they have made the transaction) and the business document which consolidates the transaction is a trading slip. In HP Web Services platform, SFS solves the problem of atomicity by proposing two alternative approaches to solve the problem. These two approaches are based on compensation and two-phase commit protocols. Conversations using the conversation definition language CDL are used to support transaction semantics.

4.2.6. Monitoring. Monitoring must take place during the life-cycle of a service and especially during service invocation [23]. Once a service requester (user) and a Web Service have been brought together, service execution need to be continuously monitored. Also, required adjustments are desirable to take place in real-time without affecting operations at the user's site. This is a challenging task as the Web Service may be running on a system that is not own or controlled by the user or running on an operating system that the user knows nothing about. For example, in our stock market example, we would like to specify that if the time it takes to the "*stock quote*" service to respond to a user request is more than 2 minutes for more than 5 user requests then use the "*stock quote*" service of another provider. Another example is a requirement to retrieve the traces of all the portfolio management service executions that have been triggered for a specific requestor. The situation with service monitoring becomes more complicated in the case of composite Web Services. The properties of a composite Web Service are a function of the properties of its component services and the managers of the component Web Services may need to coordinate in a particular way. One solution to this is to "outsource" the management of some Web Services to other Web Services. Initial research contribution in this aspect is provided by the Common Information Model (CIM) [8] that defines the schema and protocol standards for enabling the design of management Web Services. HP Web Services platform is able to continuously monitor service delivery, make adjustments, generate audit trails and ensure contract compliance. CIM and HP Web Services platform address the syntactic aspects of monitoring and no provision is made for semantics. Another contribution to monitoring may be found in [13] where the traces of composite service executions are collected incrementally through peer-to-peer interactions between the involved providers, stored in distributed repositories and are then made available for auditing, customer feedback and quality assessment.

4.2.7. Billing. Billing concerns service brokers and service providers. Service brokers create and manage taxonomies, register services and offer rapid lookup for services and companies. They may also offer value-added information for services, such as statistical information for the service usage and QoS data. One key issue for brokers is how to make profit out of these provided services. On the other hand, service providers need to charge the users of a Web Service. Unlike today's software packages, which are typically made available through licenses based on a per-user or site pricing model, Web Services will be most likely be licensed according to a "pay-as-you-go" subscription based pricing model. This has the potential to significantly reduce the IT-related costs for supporting software within an enterprise. Rather than having to buy monolithic software packages, wherein users might only use a fraction of the whole package, the Web Service model allows users to pick and choose the precise bits of functionality for the time interval that are needed to perform a specific task. This means that the use of the service should be monitored and billed. Standards do not provide any answers to these questions and research results are still minimal. An initial research contribution is the meter and accounting model described in [11] that operates on a base assumption that services with a high degree of value are contracted via Service Level Agreements (SLA's) or their equivalent. This implies that both the provider and the requester agree to the contract. This model has been developed as a service itself. The service stores the contract details, such as type of contract (long term, ad hoc, etc.), start and expiration dates of contract, the time model to be used (everyday, once weekly, etc.) and security details. A number of alternative business models such as pay-per-click/fee-for-use model, subscription and lease model can be used by the meter. HP Web Services platform continuously tracks service usage and allows the service provider to bill only for the time the service was actually used. None of the two previous solutions fully address the semantic aspects of billing. Referring to our stock market example, if we would like to specify that discounts of 10% are made to requesters that trigger the service more than 20 times per day, there is no way to do this using existing solutions.

4.2.8. Contracts. A contract specifies, usually in measurable terms, what services the provider will furnish. Some metrics that contracts may specify include: what percentage of the time services will be available, the number of users that can be served simultaneously, the schedule for notification in advance of changes that may affect users and help desk response time for various classes of problems. In ebXML a Collaboration Protocol Agreement (CPA) is created during an engagement between two parties. It contains information that outlines the service and the process requirements agreed upon by all parties. The TpaML (Trading Partner Markup Language) is an additional specification proposed by IBM to manage conversations between trading partners. The notion of conversations is also used in HP Web Services platform. Other research attempts in contracting include E-ADOME [22] where a contract model based on workflow views is proposed; e-contract management is supported in a cross-organizational Web Service workflow environment.

4.3. Summary of technical challenges

Table 1 illustrates the activities in the Web Service life-cycle and summarizes the contribution of each of the examined standards, i.e. WSDL, UDDI, SOAP and ebXML in these activities from syntactic and semantic point of view; the table also summarizes the contribution of the other solutions and research efforts, e.g. HP Web Services platform, e-Flow, the service model of CMI, SDL etc., briefly examined in the previous section. The contribution of research efforts appears in the table by using the reference number in the respective cell. e.g. the entry [13] in the table means that the work by Fauvet et al. contributes in Web Service Composition at the syntactic level.

We can see that standards focus mainly on the basic activities in the Web Service life-cycle, i.e. on description, publishing, discovery, binding and invocation. Especially WSDL and SOAP focus only on the syntactic aspects of service description, discovery, binding and invocation. UDDI attempts to tackle apart from the syntactic level, the semantic level of publishing and discovery. On the other hand, ebXML deals with the syntactic and semantic level of all basic activities. This is due to the fact that ebXML follows a top-down approach to

Table 1. Contribution of standards in the web service life-cycle.

Web service life-cycle	Syntactic level	Semantic level
Basic activities		
Description	WSDL, ebXML, ADS, HP WS Platform, SDL	ebXML, HP WS Platform, SDL
Publishing	UDDI, ebXML, eFlow, ADS, HP WS Platform	UDDI, ebXML, HP WS Platform
Discovery	WSDL, UDDI, ebXML, HP WS Platform, SRL	UDDI, ebXML, HP WS Platform, SRL
Binding and Invocation	SOAP, ebXML, WS Platform	ebXML, HP WS Platform
Value added activities		
Composition	eFlow, CMI, WSFL, WSCL, XLANG, BPML, [7, 13]	eFlow, CMI, [13]
Brokering	HP WS Platform, [2, 14, 15, 25]	HP WS Platform, [2, 14, 15, 25]
Reliability	CMI, ebXML	CMI, ebXML
Security	ebXML, XKMS, XML Sign, XML Encr, OASIS, HP WS Platform	
Transactions	ebXML, XAML, XLANG, HP WS Platform	ebXML, HP WS Platform
Monitoring	CIM, HP WS Platform, [13]	
Billing	[11], HP WS Platform	[11]
Contracting	ebXML, TpaML, HP WS Platform, E-ADOME	ebXML

application development and business interoperation, while WSDL, SOAP and UDDI follow a bottom-up approach, i.e. from the syntactic level using WSDL and SOAP to the higher abstraction level using UDDI. However, none of these standards deals with the significant task of analyzing and expressing Web Service related business (user) requirements.

The value added activities in the Web Service life-cycle, i.e. composition, brokering, reliability, security, transactions, monitoring, billing and contracting are not dealt with by the WSDL, SOAP and UDDI approach. These activities are partly addressed by ebXML. Other products and research efforts attempt to deal with some of these activities, as it was shown from the analysis before and it is illustrated in the table. However, most of these efforts, on one hand they are still at early research and development stages and on the other, they are characterized by their fragmentation. This makes it very hard to contribute toward a unified framework and standards to support the whole Web service life-cycle.

Following these observations, we present below our concluding remarks.

5. Conclusion

The Web Service paradigm is an evolution of the object-oriented paradigm as well as of the component paradigm. At the same time, it constitutes the next stage of evolution for e-business application development and interoperation. This paradigm changes the philosophy of building software as it motivates developers to build Web Services that can be reached, used by and collaborate with any business application across the Internet. Also, by allowing legacy applications to be exposed as services, it further promotes business interoperability. In addition, it motivates developers to build applications by locating and using existing Web Services, rather than building the required functionality from scratch, promoting in this way easy and fast deployment, efficient application development and just-in-time integration.

All these anticipated advantages of the Web Service paradigm, in order to be enjoyed, require adequate support by technology and standards. In this paper we analyzed the activities that take place during the Web Service life-cycle, the technical challenges associated with these activities and the contribution of the widely used evolving standards of WSDL, UDDI, SOAP and ebXML and of other standards, related products and research efforts. It became apparent that many of the technical challenges are either partially or not at all addressed by existing standards and solutions leaving open a lot of issues to be solved.

Although, all the activities and technical challenges in the Web Service life-cycle are essential for Web Service application development, we believe that the following ones are of highest priority; this is because they constitute the basis for the Web Service paradigm to be exploited by business users bringing them the promised benefits. These activities are:

- Analysis and description of business (user) needs
- Appropriate description of the functionality of Web Services at a conceptual level
- Classification of Web Services according to their functionality
- Matching of user needs to offered services
- Composition of existing services to match user needs

The analysis and description of business needs related to Web Services is the first step that needs to be supported. Otherwise, it is rather impossible for a business to benefit from the rest of the Web Service infrastructure. The existing standards and solutions offer minimal support to this task. We also consider the efficient description of Web Service functionality at a semantic level and its effective classification of great importance, too. A business user, after having analysed and described their needs, should be able to browse and examine offered solutions at a semantic level, before being concerned about the syntactic aspects of the offered services. Thus, further support is needed for efficient matching of business (user) needs to offered services. Along these tasks, Web Service composition constitutes a significant task as it is not always possible to find a Web Service that exactly matches a business request.

It is the authors' belief that a convergence between evolving standards, commercial products and research efforts can contribute in the development of a unified and globally accepted framework to support the whole Web Service life-cycle, i.e. from the description of business needs and the semantic description of offered Web Services to the value-activities of composition, monitoring, billing, contracting and so on. When such a framework is established, we can talk about a real revolution in the e-business application development and interoperability.

Acknowledgment

The authors would like to thank Dr. Dimitrios Georgakopoulos from Telcordia Technologies, Inc. for his comments in earlier drafts of this manuscript. Thanks are also due to the participants of the TES Workshop, Rome, 14–15 Sept. 2001, for the useful insights and discussions which contributed in the elaboration of this paper.

References

1. Advertisement and Discovery of Services (ADS) protocol for Web Services, http://www.cn.ibm.com/developerWorks/web/ws-ads/index_eng.shtml.
2. M. Bichler, A. Segev, and C. Bean, "An electronic broker for business-to-business electronic e-commerce in the Internet," International Journal of Cooperative Information Systems, vol. 7, no. 4, 1998.
3. BPML, http://www.bpmi.org.
4. S. Burbeck, "The tao of e-business services: The evolution of web applications into service-oriented components with web services," http://www-106.ibm.com/library/ws-tao/?dwzone=webservices.
5. F. Casati et al., "Adaptive and dynamic service composition in eFlow," http://www.hpl.hp.com/techreports/2000/HPL-2000-39.html.
6. F. Casati et al., "eFlow: A platform for developing and managing composite web services," in Proc. AIWoRC'00, Buffalo NY, April 27–29, 2000, IEEE Comp. Society.
7. V. Christophides, R. Hull, G. Karvounarakis, A. Kumar, G. Tong, and M. Hong, "Beyond discrete web services: Composing session-oriented services in telecommunications," in Proc. of TES 2001, Rome, Italy, 2001, pp. 58–73.
8. CIM Standards, http://www.dmtf.org/spec/cims.html.
9. DCOM, http://msdn.microsoft.com/library/backgrnd/html/msdn_dcomarch.htm.
10. ebXML, http://www.ebxml.org.
11. W. Eibach and D. Kuebler, "Metering and accounting for web services: A dynamic e-business solution," http://www-106.ibm.com/library/ws-maws/?dwzone=webservices.
12. Enterprise Java Beans, http://java.sun.com/products/ejb/.
13. M.C. Fauvet, M. Dumas, and H.Y. Paik, " Peer-to-peer traced execution of composite services," in Proc. of TES 2001, Rome, Italy, 2001, pp. 103–117.
14. D. Georgakopoulos, H. Schuster, A. Chicocki, and D. Baker, "Managing process and service fusion in virtual enterprises," Information Systems, vol. 24, no. 6, pp. 429–456, 1999.
15. A. Geppert, M. Kradolfer, and D. Tombros, "Market-based workflow management," International Journal of Cooperative Information Systems, vol. 7, no. 4, 1998.
16. Hewlett-Packard, eFlow Model and Architecture, version 1.0, 1999.
17. HP Web Services Platform, http://www.hpmiddleware.com/products/hp_web_services/.
18. http://xml.coverpages.org/.
19. IBM UDDI Registry, http://www-3.ibm.com/services/uddi/.
20. R. Irani, "Practical considerations in implementing web services," http://www.webservicesarchitect.com/content/articles/irani01.asp.
21. Jini, http://www.sun.com/jini.
22. E. Kafeza, D.K.W. Chiu, and I. Kafeza, "View-based contracts in a web service cross-organizational workflow environment," in Proc. of TES 2001, Rome, Italy, 2001, pp. 74–88.
23. V. Machiraju, M. Dekhil, M. Griss, and K. Wurster, "Web services management requirements," in Proc. of Technologies for Web Services Workshop, Cairo, Egypt, September 2000.
24. Microsoft UDDI Registry, http://uddi.microsoft.com/.
25. M.H. Nodine, W. Bohrer, and A.H. Ngu, "Semantic brokering over dynamic heterogeneous data sources in infosleuth," in Proc. of Int. Conf. Data Eng. ICDE, 1999, pp. 358–365.
26. OMG IDL, http://www.omg.org/gettingstarted/omg_idl.htm.

27. H. Schuster, D. Georgakopoulos, A. Cichocki, and D. Baker, "Modeling and composing service-based and reference process-based multi-enterprise processes," in Proc. of CaiSE 2000, Lecture Notes in Computer Science, vol. 1789, Springer-Verlag, Berlin, 2000, pp. 247–263.
28. S. Seltzsam, S. Borzsonyi, and A. Kemper, "Security for distributed web service composition," in Proc. of TES 2001, Rome, Italy, 2001, pp. 147–162.
29. SOAP, http://www.w3.org/TR/SOAP.
30. TpaML specification, http://www.oasis-open.org/cover/tpa.html.
31. UDDI, http://www.uddi.org.
32. W.J. Van den Heuvel, J. Yang, and M. Papazoglou, "Service representation, discovery and composition for e-marketplaces," in Proc. of the 6th Int. Conf. on Cooperative Information Systems (CoopIS2001), Trento, Italy, September 5–7, 2001.
33. Web Services architecture overview: The next stage of evolution for e-business, http://www-106.ibm.com/developerworks/library/w-ovr/?dwzone=ws.
34. WSDL, http://msdn.microsoft.com/xml/general/wsdl.asp.
35. XAML, http://www.xaml.org/.
36. XMethods, http://www.xmethods.com.