GigaTS

Web Services

About to integrate distributed applications?

GigaTS D2.2.9

Telematica Instituut

GigaTS

Colophon

Date :	November, 2001
Version :	1.0
Change :	
Project reference :	GigaTS/D2.2.9
TI reference :	TI/RS/2001/062
URL :	https://extranet.telin.nl/docuserver/dscgi/ds.py/Get/File-18739
Access permissions :	Public
Status :	Final
Editor :	J.W. Koolwaaij
Company :	Telematica Instituut
Author(s) :	P. Fennema
	J.W. Koolwaaij
	M. Lankhorst

Synopsis:

One of today's most challenging issues for real business-to-business integration is the integration of different and diverse applications. The promise of Web Services is that applications can be turned into easy to use web applications using open standards such as XML, SOAP and UDDI. This report explains the concepts and reveals the hurdles on the way to total integration of applications as Web Services.

Giga Transaction Services

The project *Giga Transaction Services* supports organizations in the development of innovative transaction services. It does so with state of the art knowledge, methods and software tools that allow for effective development of new services. GigaTS takes the business perspective as a starting point, looking at networks of organizations and the way e-commerce technology can support them. Methods and tools are rooted in a combination of technological and business knowledge of currently available and future components and e-commerce applications. In this way re-use of components is promoted and time-to-market of services is reduced. Fast and effective design and introduction of e-commerce services is our central objective.

Giga Transaction Services has three main results:

- *technology scans and demonstrators*, aiming at (next generation) Internet technology as well as general e-commerce tools, components, frameworks and standards;

- *component libraries for e-commerce and electronic trade,* capturing the essential elements of business models, transaction scenarios and ICT components for e-commerce;

- a tool-supported environment for *Rapid Service Development* (RSD), a methodology for the design and development of transaction services, linked to software and e-commerce development tools.

These products are incrementally built. On a regular basis, results come available to the public, and in seminars and on our web site [GigaTS] we actively disseminate the results.

Giga Transaction Services is a knowledge-providing project under GigaPort Applications. The Telematics Institute carries out the larger part of the work. The project emphatically tries to actively transfer knowledge and results to companies and organizations in pilot in a next generation Internet setting. Such pilots can consist of experiments with new services, the design or evaluation of e-commerce services and architectures, or company specific scans and demonstrators.

Table of Contents

1 Introduction

The time that applications were stand-alone and built for one specific purpose and one specific user(group) is not so far behind us. Today, however, applications must be generic and multi-purpose in a multi-user and multi-platform environment, and are increasingly interconnected. This report focuses on the difficulties and possibilities in integrating applications in the world we live in today.

Communication via the Internet has leveled out at least one of the traditional barriers in integrating applications across business boundaries. But leveling one barrier often reveals a couple of new ones. To name a few: how to integrate (or synchronize) business processes, what about integrity and quality of data, how secure is it to integrate applications over the Internet, how can application interfaces be described and discovered, et cetera.

In this chapter, we will discuss integration of applications both within and between enterprises, XML as an enabling technology in application integration, and finally Web Services as a new concept to expose applications and business functions via the Internet using open web protocols. In the next chapter, we will go into Web Services in more detail, provide an overview of existing technologies that support Web Services, and discuss the benefits and drawbacks of Web Services.

1.1 Application integration within the enterprise

EAI stands for Enterprise Application Integration and can be defined as application-independent, business process-oriented software that integrates the applications of the enterprise [Aberdeen], or several enterprises [Gilpin01]. Although the acronym EAI contains the E of Enterprise, EAI is also used for application integration across multiple enterprises. A synonymous acronym is IAI (Internet Application Integration), which stands for integrating applications across multiple enterprises in order to automate multi-enterprise business processes where the Internet provides the communications backbone.

In this document, we will reserve the acronym EAI for the process of bringing together systems that reside within a single enterprise. For integration between multiple enterprises we will use the acronym B2Bi (business to business integration). And although B2Bi might sound like a panacea, and look like the promised land, the road to successful B2Bi will not be smooth and without hurdles, and it is good to realize that B2Bi still requires EAI. In the next chapter, we will go into Web Services as a concept that connects the front doors of multiple enterprise, but first we have to connect the front door with all the different systems (databases, legacy systems and core applications) in the back office. That is the task of EAI. Here we will focus on B2Bi; for a more elaborate discussion of EAI we refer the reader to [Koolwaaij01] which deals with EAI in much more detail.

1.2 Business to business integration

Despite the barriers which are still left, and the struggling of organizations to support the internal requirements B2Bi is really taking off. Figure 1-1 shows the evolution in applications and integration from standalone systems (stovepipes) via EAI to real integration of component based, loosely coupled, Internet connected, distributed applications. Most of the EAI providers mentioned in [Koolwaaij01] are moving towards business integration, because most of the techniques and technology that make up EAI are also applicable to B2Bi.

WebMethods, one of the largest B2Bi providers, sells their integration platform by representing it as “the solution for integrating the extended enterprise by addressing the six key elements encompassing a business process - enterprise applications, mainframe and legacy applications, databases and data warehouses, human workflow, Web Services and business partners.” [WebMethods] In other words, on top of integration of enterprise-internal applications, databases and legacy systems, B2Bi has to integrate businesses, by exposing their applications to their business partners using Web Services, and to be able to find business partners and the services they offer in a dynamic way.

This way, B2Bi is inducing one of the most remarkable changes in the business world today, namely the blurring of corporate boundaries and the new capabilities to freely exchange the information that drives business processes. Although B2Bi is often over-hyped, the e-business is much more closely integrated with its customers, suppliers and partners simply because the communications technology enables this to happen. And whilst the focus of EDI (electronic data interchange) was mainly on integration of data, the focus of new Internet based technologies -like B2Bi, XML and Web Services- is not only on integration of data, but also on integration of business functions and processes, thus enabling companies to conduct business in a much more flexible manner.

This means that the relationships can be established and dismantled rapidly, making transient relationships viable. Especially for specific projects and non-core processes, it will be possible to dynamically find a suited third party that offers the desired product, service or information. In this scenario, it becomes more difficult to define the scope of an e-business application as parts of the business components are executing in completely different environments. This requires open protocols to connect to and integrate with these business components, in particular pre-defined, well-designed and agreed-upon interface definitions and process specifications, to obtain a much looser coupling of technologies [Lankhorst01].

In this context, one of most promising concepts in B2Bi of today is the notion of Web Services. Web Services can be seen as business functions exposed to the web using standard, open web protocols, which allow companies and individuals to make their digital assets available to the global community at large in a simple and effective manner. XML plays an important role in the definition, implementation, and execution of Web Services. Therefore, we will first introduce XML, followed by the concept of Web Services and its stack of XML based technologies for messaging, transport, description, and invocation.

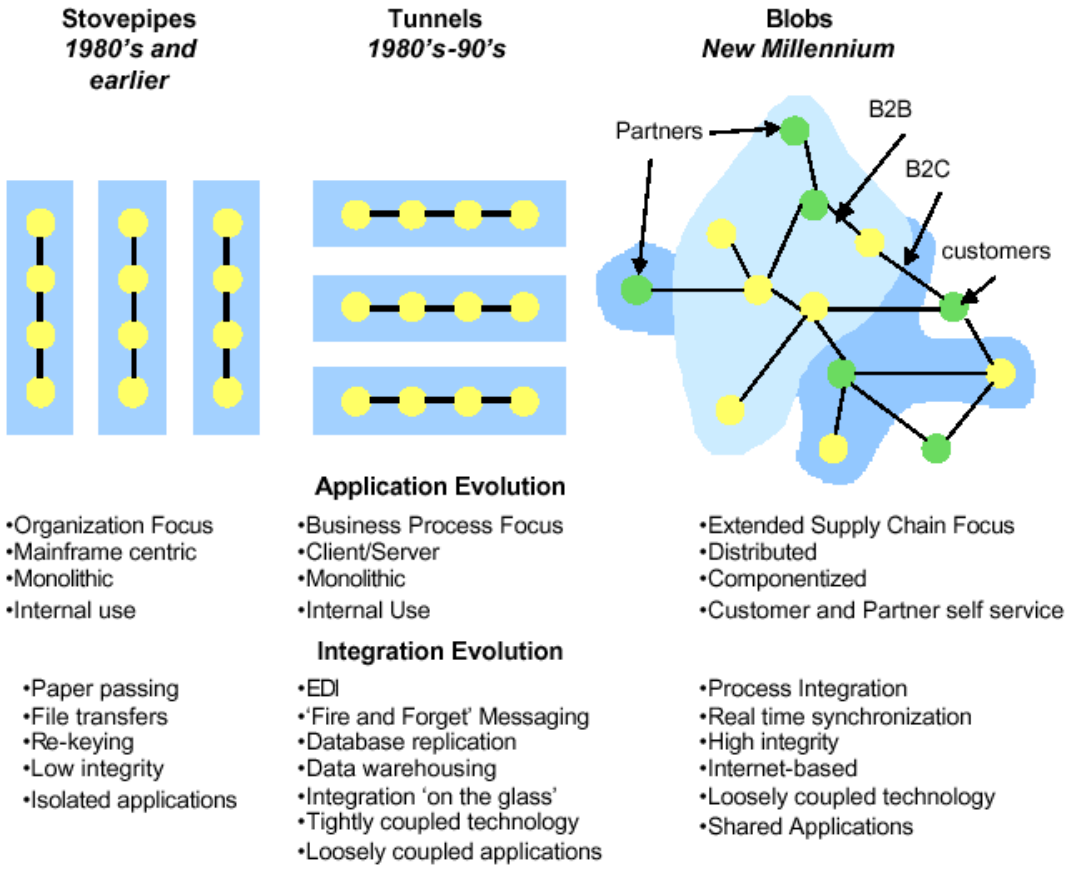

Figure 1-1: Application and Integration evolution [Wilkes01]

1.3 The role of XML

Extensible Markup Language (XML) is said to be the ASCII of the 21st century for structured data. And in a way, that is true: XML basically is very simple, but opens new perspectives for B2Bi as soon as parties agree to represent their data in XML.

XML is a flexible and extensible markup language derived from the Standard Generalized Markup Language (SGML, ISO 8879). SGML is the mother of all markup languages, and XML can be seen as SGML, without those parts that are hard to implement in software products, and targeted to web applications. For a complete overview of the differences between SGML and XML we refer to [Clark01].

As said, XML is a so-called markup language. Historically, the editor scribbled markup in a text to describe how the text should be laid out. Nowadays, markup defines the meaning of a text. One of the major benefits of XML is that it separates the structure, content, and layout of documents, as opposed to HTML, which is primarily a formatting language. HTML is loosely based on SGML and can be seen as SGML with exactly one fixed document type definition (DTD), which is focused on the formatting of documents. In XML, the user can specify the

document structure in a user-defined DTD, which is extensible. Structure and content are strictly separated, hence a DTD can be used for several XML documents as a template. Finally, the presentation is specified in a style sheet. Using different style sheets, the same content can be presented in numerous ways, without having to reorganize this content.

These are the primary ingredients: the content in XML, a document type definition or schema specifying structure, and a presentation specification that states how the information in an XML document should be displayed. Its simplicity, however, is also its power: instead of having a fixed language (e.g. HTML, which specifies the content of hypertext documents), XML is a meta-language that opens the possibility to define new languages, new formats. These can be used for many different purposes, including the specification of

- configuration files in servers,
- messages,
- documents (e.g. DocBook),
- interface definition (as in C#),
- structure documents (XML Schema is also written in XML),
- envelopes (e.g. SOAP),
- metadata (e.g. MPEG7),
- graphics (e.g. SVG),

and much more. XML is so powerful because it can be used in a wide range of applications, yet the support (in terms of tools and code) can be very generic. For data structuring purposes, XML will be what ASCII is for uniform character encoding, and both simplify (or even enable) communication between totally different applications. It is good to keep in mind that XML is often the foundation layer upon which a myriad of higher-level standards has been built. In the W3C family alone, there are already roughly twenty XML based standards. (An impression is depicted in Figure 1-2.)

Below you see an example of an XML document specifying a Scalable Vector Graphic (SVG). It conforms to the DTD for SVG documents, and shows a simple button with the text 'BehaviourDiagram'.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE svg SYSTEM "svg-20001102.dtd">
<svg viewBox="0 0 500 200" xmlns:telin="http://www.telin.nl">
    <desc>SVG image for RSD</desc>
    <title>B2B2ME Scenario in RSD</title>
    <g id="Button1" style="cursor:hand;" onclick="telin:show('BehaviourDiagram');">
        <rect class="menu" y="50" width="350" height="70" rx="10" ry="10" x="40"/>
        <text class="menu" y="100" x="60">BehaviourDiagram</text>
    </g>
</svg>
```

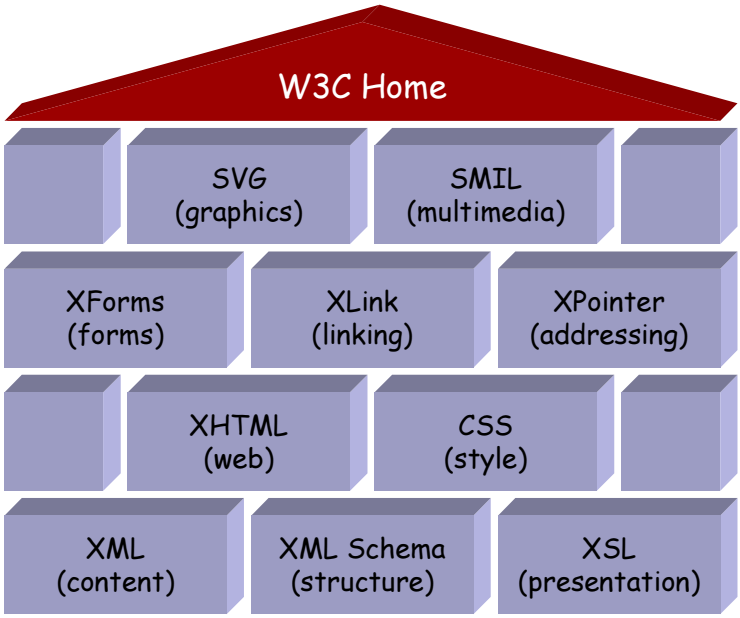

Figure 1-2: A few important components in the W3C family of standards.

In B2B communication, XML is most frequently used as a language to structure messages or documents according to an agreed-upon schema or DTD.

XML and XML schema are being there to stay. The power of XML lies in the separation between content, structure and presentation. Representing data in XML will ensure the durability and accessibility of the data for a long time to come. XML will be the ASCII of the 21st century, which enable standardized communication between parties.

1.4 Transforming XML

In practice, however, it will never be the case that all procurement systems in—let's say—the IT sector use exactly the same XML schema to describe their purchase orders. Typically, the in-house format (as specified in a database or XML schema) needs to be converted to a public XML format, which can be sent to a party that is known to understand messages in such a format.

The Extensible Style Sheet Language (XSL) is designed to apply representation style to XML documents. XSL is a specification under development by the World Wide Web Consortium for applying formatting to XML documents in a standard way. XSL includes a transformation language (XSLT), a formatting language (XSL-FO), and a query language (XPATH). Each of these languages is represented in XML. The 1.0 versions of the transformation language and the query language are W3C recommendations; version 1.0 of the formatting language is still a candidate recommendation.

The most important part is XSLT: it describes templates that define rules for how one XML structure is transformed into another XML structure. In other words, XSLT can be used to map an XML document conformant to XML schema A to a XML document conformant to XML

schema B. Examples of transformations include filtering, re-ordering, and extension of XML documents. For example, XSLT can be used to transform a Rossetanet purchase order to an xCBL purchase order (for an explanation on RosettaNet and xCBL see Section 1.5), or to take that same purchase order and transform it into good-looking, human-readable HTML, or to translate into an in-house XML format which can be read by a (legacy) application. Thus, its ability to transform data from one XML representation into another XML representation has numerous applications in XML-based electronic data interchange, metadata exchange and in general everywhere where there is a need to convert between two or more data representations.

During an XSL transformation, an XSL processor reads an XML document (or better: its DOM, the XML representation in memory) and the desired XSL style sheet. Based on the instructions in the style sheet the XSL processor outputs another XML document. There is special support for outputting HTML documents (and even for output of plain text documents).

Let us consider the following sample XML document describing one single party.

```
<?xml version="1.0" encoding="UTF-8"?>
<PARTYINFO party_type="INSURANT">
    <NAME>Bob Jonson</NAME>
    <ADDRESS>
        <STREET>Viaweg</STREET>
        <NUMBER>15</NUMBER>
        <CITY>Enschede</CITY>
    </ADDRESS>
</PARTYINFO>
```

An XSL document basically contains a list of templates. A template rule has a pattern specifying the nodes it applies to, and its content is ‘executed’ when a node matching the pattern is parsed. A template normally contains new XML elements and values, copies sub-trees, elements and values from the source XML tree, and applies other templates. An example is shown below. It contains two simple templates: one matching the document root, and one for the PARTYINFO element.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:template match="/">
        <html>
            <head>
                <title>
                    Information about
                    <xsl:value-of select="PARTYINFO/@party_type"/>
                </title>
```

```
            </head>
            <body>
                <xsl:apply-templates select="PARTYINFO"/>
            </body>
        </html>
    </xsl:template>
    <xsl:template match="PARTYINFO">
            Name:<xsl:value-of select="NAME"/><br />
                Street: <xsl:value-of select="ADDRESS/STREET"/><br />
            Number: <xsl:value-of select="ADDRESS/NUMBER"/><br />
            City: <xsl:value-of select="ADDRESS/CITY"/><br />
    </xsl:template>

</xsl:stylesheet>
```

The result is a simple HTML document representing name and address of each PARTYINFO element in the source XML document.

Name: Bob Jonson
Street: Viaweg
Number: 15
City: Enschede

A number of tools exist to assist you in writing XSL transformations. The most well known include Biztalk mapper, IBM XSLerator, Whitehill <xsl>composer, and XMLSpy 4.0. To give a impression of these tools: Biztalk mapper offers a GUI to design a mapping between one XML schema and another. It provides drag-and-drop mapping of input fields to output fields, pre-build transformation functions ("functoids") for string manipulation, mathematical functions, logical functions, date & time, scientific functions, scripting. And finally, it generates transformation maps based on XSLT. Figure 1-3 offers a view on the GUI of Biztalk mapper. In the lower panel you see the generated XSLT.

XSLT is an indispensable technology for transformation of XML content from one schema definition to another. It will be supported more and more by dedicated tools as well as the larger B2Bi platforms. It has a very powerful addressing language and a modular, template-based approach, which make it useful for almost all transformations that do not require too much 'external intelligence', otherwise additional scripting might be necessary.

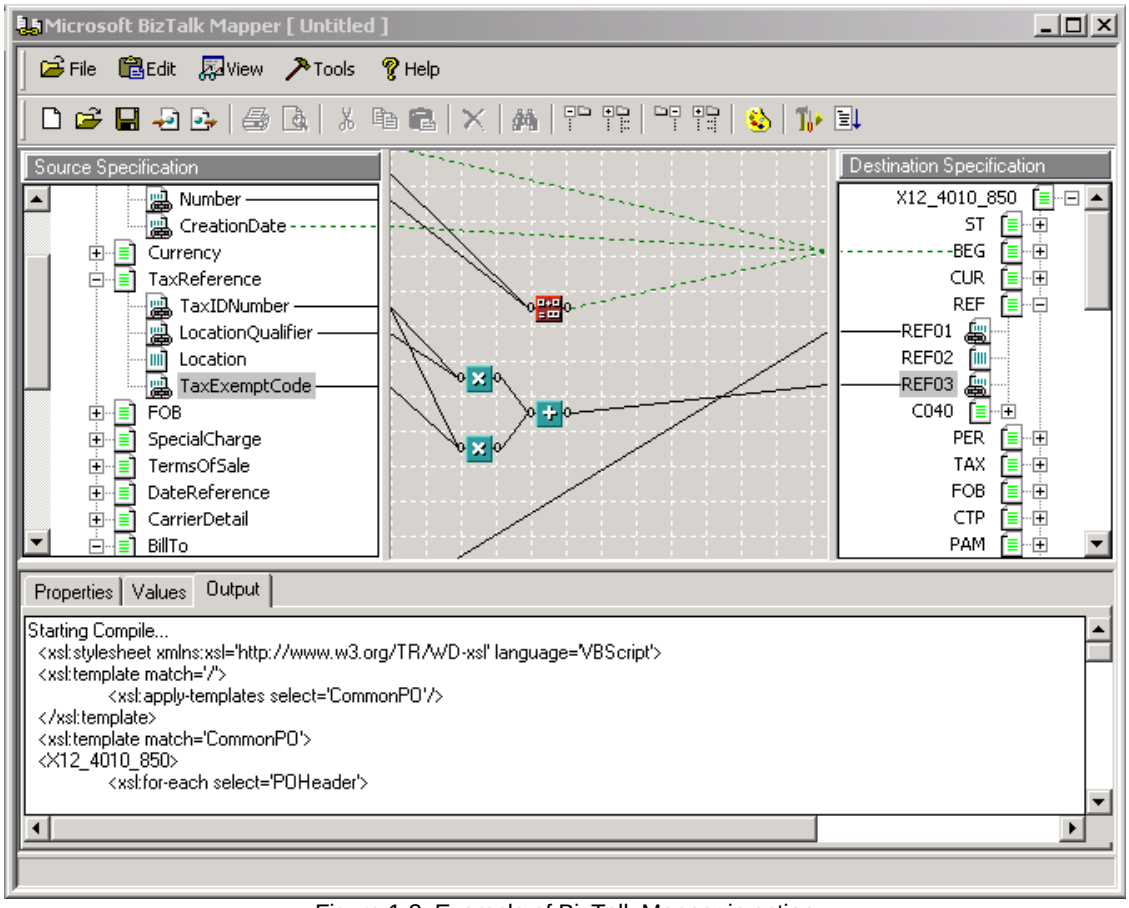

Figure 1-3: Example of BizTalk Mapper in action

1.5 Messaging standards

We briefly go into XML messaging standards for e-business. There exist quite a number of standards, commercial (cXML, BMEcat, ebXML), financial (finXML, OFX, BIPS), supply-chain (RosettaNet, eCo, BizTalk), et cetera. For a more complete overview we have to refer to other sources, as [Janssen02] and [XML02]. In this section, we will focus on two open initiatives that supply concrete structure definitions of their XML messages for B2B exchange of business documents in the form of document type definitions or schemas, namely xCBL and RosettaNet.

xCBL

Commerce One's Common Business Library (xCBL) 3.0 was announced in November 2000. It covers a wide variety of B2B e-commerce scenarios, in both the MRO/Indirect Procurement and Direct Goods arenas. It is an open XML specification for the cross-industry exchange of business documents such as product descriptions, purchase orders, invoices, and shipping schedules. xCBL 3.0 is a set of XML building blocks and a document framework that allows the creation of robust, reusable, XML documents for electronic commerce. Using the xCBL 3.0 document framework, businesses everywhere can exchange business documents of different types, resulting in frictionless electronic commerce across multiple trading

communities. For businesses already using traditional Electronic Data Interchange (EDI) standards, xCBL 3.0 provides a transition path to an XML-based commerce capability [xCBL].

RosettaNet

RosettaNet is an organization set up by leading information technology companies to define and implement a common set of standards for e-business. RosettaNet is defining a common parts dictionary so that different companies can define the same product the same way. It is also defining up to 100 e-business transaction processes and standardizing them. Because RosettaNet is supported by all or most of the major companies in the IT industry, its standards are expected to be widely adopted [RosettaNet]. The standards being developed by RosettaNet include:

- PIPs: RosettaNet Partner Interface Processes define business processes between trading partners in several so-called clusters, including order management, product information, and inventory management.
- Dictionaries: a common set of properties for PIPs. The RosettaNet Business Dictionary designates the properties used in basic business activities. RosettaNet Technical Dictionaries provide properties for defining products.
- RNIF: The RosettaNet Implementation Framework provides exchange protocols for quick and efficient implementation of PIPs.
- Product & Partner Codes: Product and partner codes in RosettaNet standards expedite the alignment of business processes between trading partners. Included are both identification codes (DUNS and GTIN) and classification codes (UNSPSC).

A practical example of a PIP is number 3A2 with the name ‘Query Price and Availability’. This PIP contains two message specifications (PriceAndAvailabilityQuery and PriceAndAvailabilityResponse) in the form of DTDs, and a Word document specifying the business process flow (choreography), and the message exchange control parameters. The focus is on the "public business process", the description of what the seller will do and how the buyer will respond.

RosettaNet should upgrade to XML schema, define their PIPs in a machine-readable format (for example, according to ebXML’s business process specification scheme), and provide service interface definitions in order to be successfully applied in Web Services

1.6 Emerging Web Services

These XML messaging standards form the basis for new B2B services offered over the Internet, or in short Web Services. Web Services are a new model for using the web for building applications. Transactions can be initiated automatically by a program, not necessarily by a human using a browser, can be described, published, discovered, and invoked dynamically in a distributed computing environment, in which XML plays an important *enabling* role. And that while users of these services will be often unaware of the XML technologies under the hood, and developers not spending too much time on some layers in his implementation, because a substantial part of the work has already been done.

The notion of Web Services can fundamentally change application development. Rather than building monolithic applications that are not built with the vision to integrate with other applications one day, the next generation of applications will be built from the ground up with integration in mind. This will open up a broad array of options for developers. For example, rather than building their own billing component for an e-commerce application, developers will be able to call an external billing component, which has published its interface definition in a central registry, and which can be invoked using open protocols via the web.

In the next chapter we will see what Web Services are exactly, and which XML standards are important in that context.

2 Web Services

2.1 Concept

Before we introduce *Web Service*s, let us first think a moment about what a *service* is exactly. In our view,

> *A service is the externally visible part of a specific business function.*

For example, a bank offers a service to change money. Such a service is 'invoked' by stepping up to the counter and asking the clerk behind the counter to change your money into a specific foreign currency while handing over an amount of money in your national currency. You do not have to be aware of the business function itself, e.g. where and how the foreign money is stored and that the clerk has to check all bank notes with UV light. It suffices for you to ask to change this amount of money in that currency.

This looks very simple, but how do you know where the bank is, how do you know which services the bank offers, and how do you know what to ask the clerk when you want to have your money changed? The answer is simple: humans have learned all the answers throughout your life and have built up enough context information to complete the process.

For application to application communication, however, all context information has to be gathered at the moment of the interaction. Web Services make application to application communication possible by defining exactly the semantics, protocols, and interfaces needed in the interaction. Let us first define what a Web Service is.

> *A Web Service is the externally visible part of a specific business function that is exposed to the web via a well-defined interface, and described and invoked via standard web protocols.*

For example, a bank offers a CurrencyConversion Web Service. Such a service is invoked by pointing to the correct web location (usually a URL), sending a method call to that location in an agreed-upon format, using the input parameters FromAmount, FromCurrency, ToCurrency, as specified in the service description, and waiting to receive the ToAmount result. You do not have to be aware of the Web Service implementation details, on which platform it runs, in which language it has been written, et cetera.

This also looks very simple, but how does an application which wants to convert an amount from currency A to currency B know where the Web Service is located, which Web Services the bank offers, and what the input and output parameters are? The answer is that all these things have to be specified exactly in a machine-readable, open and agreed-upon format, and

be stored at a specific web location. In this chapter, we will investigate in detail which technologies are necessary to use Web Services successfully.

Web Services can be seen as building blocks for creating open distributed systems, which allow companies and individuals to make their digital assets available to the global community at large in a simple and effective manner. The spectrum of possible Web Services is impressive, and ranges from simple single functions such as specific financial calculations to entire processes such as airline ticket reservation. Web Services can be reused across multiple applications, hence allowing fast and efficient web application development. In this context, we can understand the slogans ‘apps on tap’ of HP and ‘no more gaps between apps’ of Microsoft.

In general, three roles can be distinguished in the Web Services concept: a service provider, a service requestor, and a service registry. The requestor and the provider are able to discover each other via a third party, the registry, which holds up-to-date information about businesses, and the services they offer. This way, a service requestor can discover existing Web Services, determine their purpose, functionality, and operation instructions, and hence use a service to his benefit. Thus, we can identify the following actions:

- The party providing a service must *publish* the availability of this service via a central registry.
- The party requesting a certain service is then able to *find* all services meeting his needs and select the best fitting service by consulting the registry.
- Finally, the service requester *binds* to the service provider and is ready to use the service.

These roles and actions are schematically depicted in Figure 2-1.

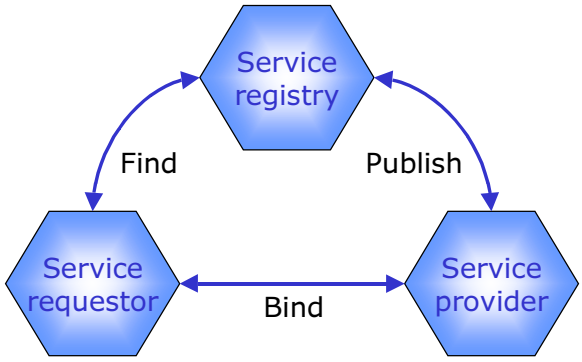

Figure 2-1 Three different roles and their interactions in the Web Services concept

2.2 Web Service architecture

In this section, we will investigate an example of a Web Service architecture. Suppose that four single servers expose functions (stock, calendar, e-mail and news) as Web Services. Since their interface definitions are published in the registry of a central third party, a client will be able to invoke these services at a specified access point (usually a URL). Communication

with the Web Service is possible via SOAP (see section 2.4.1) through firewalls since they use HTTP as a transport mechanism (as depicted Figure 2-2).

The client is then able to build multiple Web Services into a web application simply by pointing to the appropriate access point. At runtime, all the service calls will automatically be packaged and handled through an XML interface. Developers can create and use Web Services on any platform, in any programming language, since the standardized service interface hides the implementation details of the service. This enables e-business application developers build applications more quickly by assembling them from existing, reusable Web Services instead of recreating functionality again and again. It is not hard to imagine that the keywords for Web Services will be loosely coupled, component-oriented and cross-technology.

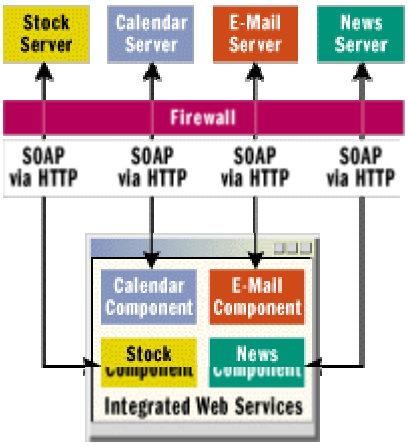

Figure 2-2: Example of an aggregated Web Service

A generic architecture for a Web Service is shown in Figure 2-3. The architecture is divided into five logical layers. Furthest from the client is the data layer, which stores information required by the Web Service. Above the data layer is the data access layer, which presents a logical view of the physical data to the business layer. The data access layer isolates business logic from changes to the underlying data stores and ensures the integrity of the data. The business layer implements the business logic of the Web Service. [Kirtland01]

As in Figure 2-3, a web application is often subdivided into two parts: the business façade and the business logic. The business façade provides a simple interface that maps directly to operations exposed by the Web Service. The business façade uses services provided by the business logic layer. In a simple Web Service, all the business logic might be implemented by the business façade, which would interact directly with the data access layer. Client applications interact with the Web Service listener. The listener is responsible for receiving incoming messages containing requests for service, parsing the messages, and dispatching the request to the appropriate method on the business façade. If the service returns a response, the listener is also responsible for packaging the response from the business façade into a message and sending that back to the client. The listener also handles requests

for contracts and other documents about the Web Service. So, the only part of the Web Service that knows it is part of a Web Service is the listener!

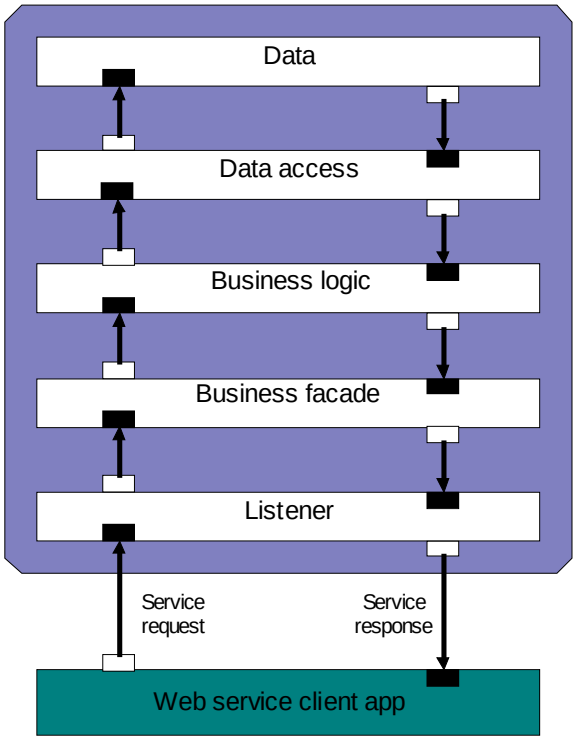

Figure 2-3: Generic Web Service architecture

2.3 Examples of Web Services

Current examples of Web Services usually perform a single function, but still they are very useful to demonstrate the concept of Web Services to work in practice. In this section we elaborate on three examples of such Web Services, namely a distributed computing service, a stock quote service and a translation service.

A quick search in March 2001 (via http://www.salcentral.com) delivers 54 examples of Web Services, including an address book service, a random number generator (dice thrower), simple arithmetic services (calculator, factorial, gcd), conversion services (unit of measure, currency), financial services (tax, stocks), and information services (weather, traffic, FedEx tracker). In July 2001, there are already 105 examples to be found at SalCentral, including more complex examples are airline flight information and Ebay bid services.

2.3.1 Distributed computing service

A Web Service may offer its computing power to users with less powerful or very busy computing systems. Especially when a specific process is resource-intensive it may be useful to find a service provider who offers his CPU time to perform these processes. Examples of

resource-intensive computing processes include audio/video editing, financial computations, and scientific analysis.

Suppose a geological institute has developed a new mathematical model for the propagation of waves through the earth's soil during an earthquake and the effect on buildings and infrastructure. To market this new model, they may issue a complete new software package implementing the new model, but it has many advantages to make this model available via a Web Service: global reach, easier discovery, and one-spot maintenance. After the client discovered the new earthquake service, he supplies a description of the geological situation (in a format specified by the service description), and receives a report illustrating the impact of an earthquake in this situation. This way the client can investigate multiple situations without overloading his own computing power and with the most up-to-date version of the model.

2.3.2 Stock quote service

Financial services are among the most frequently mentioned examples of Web Services. A stock quote service supplies the service requestor with the most recent trade price for a given ticker symbol. An application implementing an investment rule set can use these trade prices to propose buy-or-sell decisions to the investor.

2.3.3 Translation service

A translation Web Service translates text from one human language in another. In international trade, companies often need to have an impression of the meaning of a text in a foreign language, and a service offering automatic translation may be of great help. The most well-known example of a translation service is Babelfish, which translates between English, Spanish, Portuguese, French, Italian, German, Chinese, Japanese, Korean and Russian [Babelfish].

2.3.4 Service broker

Until now, we have seen examples of simple, single-function Web Services. Let us look into the more complex example of a service broker. A service broker is also a Web Service, which provides an aggregated service combining multiple single-function Web Services. In Figure 2-4, we see a service broker which decomposes the complex need of a certain buyer for a product in the need for a catalog/inventory service offered by a supplier, a transport service offered by a logistics provider, and an information service offered by a trusted third party.

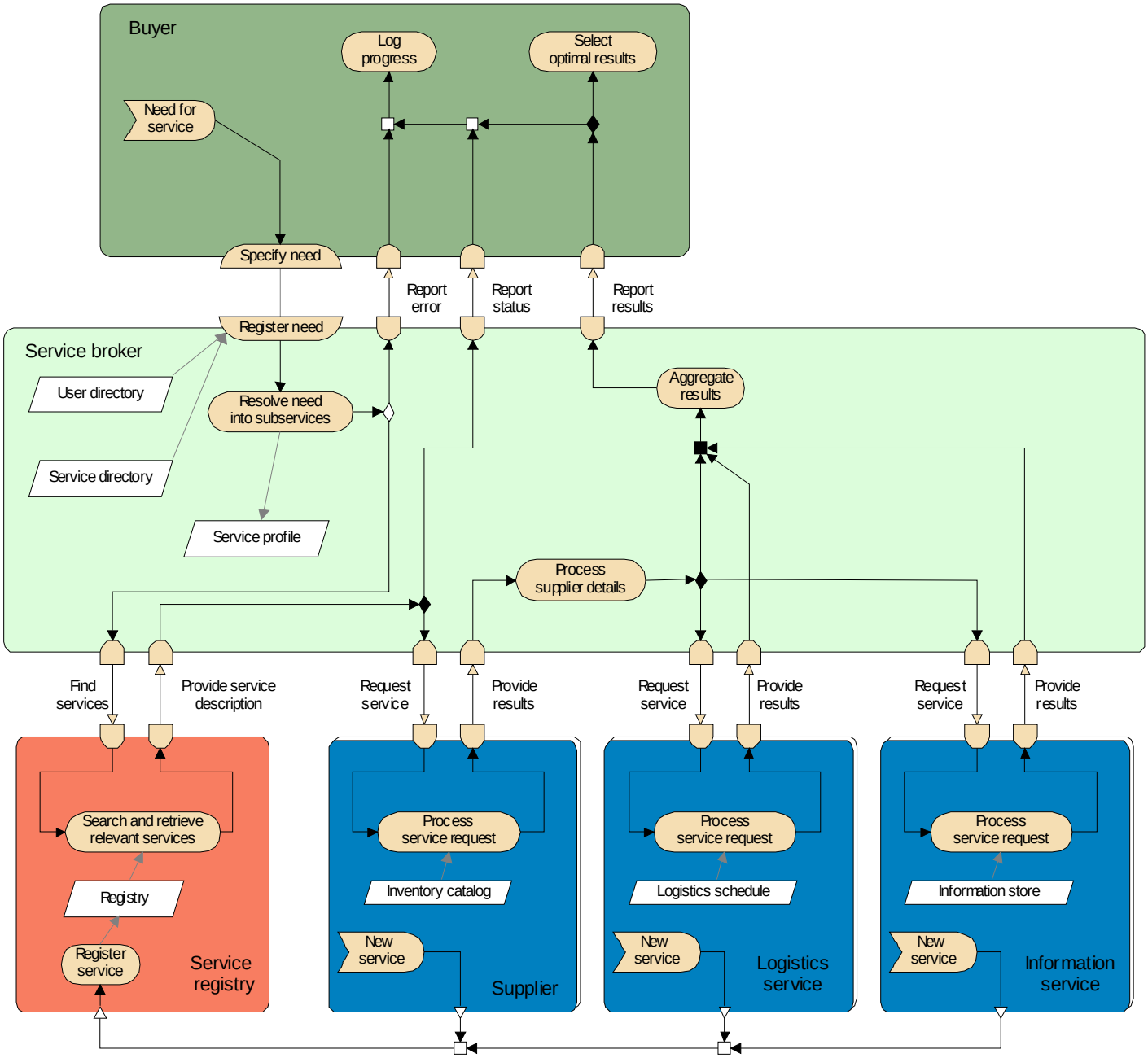

Figure 2-4: Service broker scenario

2.4 Protocols for Web Services

Let us now think about the necessary technologies to build successful Web Services. In this section, we identify the building blocks that enable the concept of Web Services to become a reality. These building blocks can be classified in a layered model. These layers (depicted in Figure 2-5) are transport, exchange, syntax, messaging, discovery, description and process.

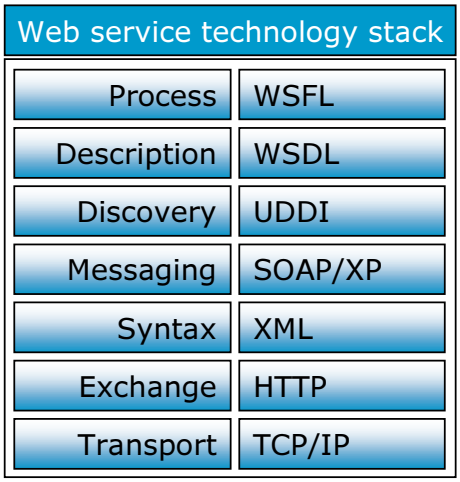

Figure 2-5: The most frequently foretold Web Service technologies

We will focus on the most important technologies in the five top layers:
1. XML offers a syntax to describe information,
2. SOAP for messaging between service requester, registry, and provider,
3. UDDI to find the necessary services,
4. WSDL to describe how the Web Service works,
5. WSFL to describe the process of interaction between multiple Web Services
In the next sections we will briefly discuss SOAP, UDDI, WSDL and WSFL. For XML we refer to the more elaborate discussion in section 1.3.

To put things in a broader perspective, we show a model with more high-level layers (process, negotiation and transaction) and also the different approaches by Microsoft, IBM, HP and ebXML in Figure 2-6. Note that although there is much competition between these approaches, there are also a substantial number of open standards that are agreed upon between these four parties. The slogan seems to be 'collaborate on standards, compete on implementation', which is of course a commendable initiative. By this time (November 2001), HP already cancelled their efforts on e-speak and adopted the MS/IBM approach of WSDL and UDDI.

The horizontal dividing line between readily applicable and supported standards and the almost unsupported standards under discussion is somewhere halfway between the discovery and the process layer. Hence, in this document we will focus on the message, transport, discovery and description layer, and only briefly go into the more high-level layers. But it is good to know that there still will be many unsolved problems in the process, negotiation and transaction layers.

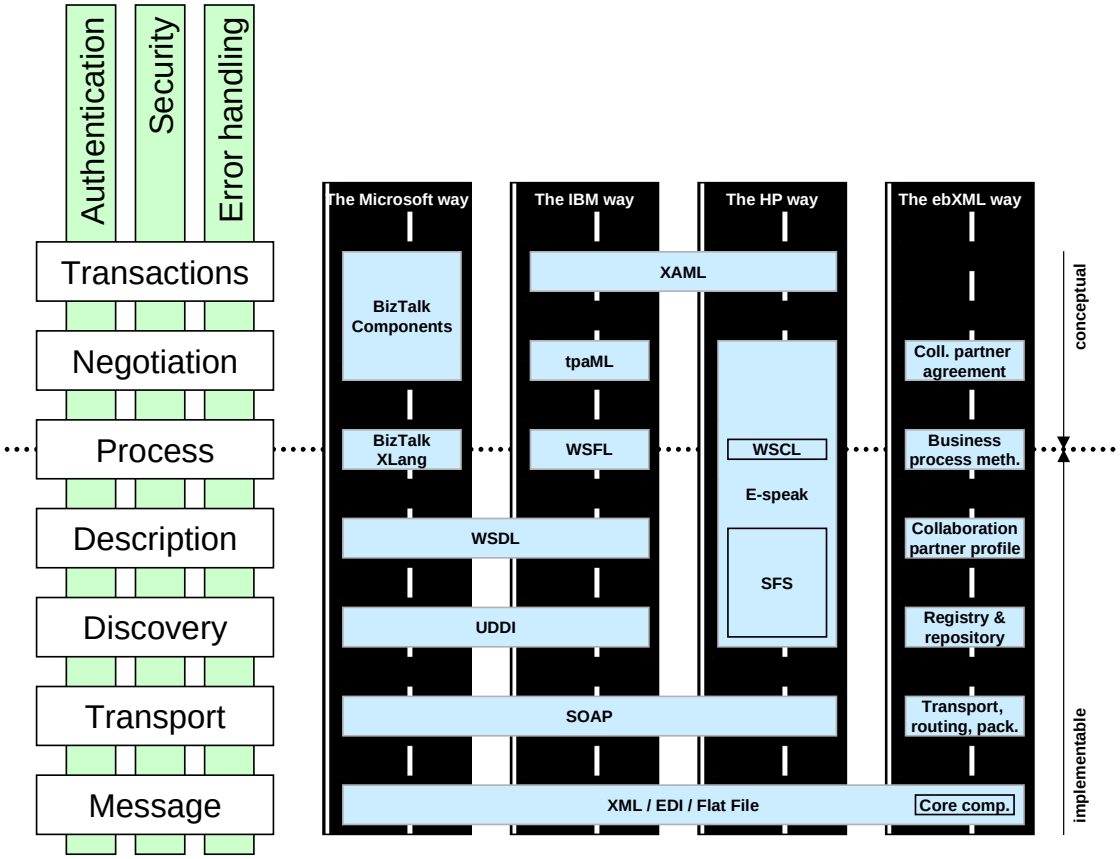

Figure 2-6: B2B integration and the standards used by Microsoft, IBM, HP and ebXML.

2.4.1 SOAP: Simple Object Access Protocol

According to the W3C note, SOAP is a lightweight protocol for the exchange of information in a decentralized, distributed environment. It is an XML-based protocol that provides a specification for:

- an envelope that defines a framework for describing what is in a message and how to process it (the header provides contextual information about the message),
- a set of serialization and encoding rules for expressing instances of application-defined data types,
- a convention for representing remote procedure calls and responses,
- binding to HTTP and other first transport layer protocols
- exception processing rules and formats

SOAP can potentially be used in combination with a variety of protocols; however, almost all available documentation describes SOAP in combination with HTTP, although SOAP is by no means restricted to the HTTP binding.

It can be seen as both an advantage and a potential danger of SOAP that, since HTTP requests are usually allowed through firewalls, SOAP method calls pass through the firewall. It is an advantage because this is the way to communicate with programs anywhere via a

method call; it is a potential danger because unwanted method calls may pass through the firewall without the firewall being able to inspect the SOAP message. The firewall only uses the designated port mechanism: a binary accept/reject mechanism per defined port.

The power of SOAP is that it refrains from implementation details on any side of the connection, and it does not invent what already exists (be it CORBA, DOM, EJB or anything). The only requirement is a mapping from SOAP to the existing component model of the application. This mapping will be done by the middleware layer as depicted in Figure 2-7, but the way how this should be done is not standardized, and highly implementation dependent.

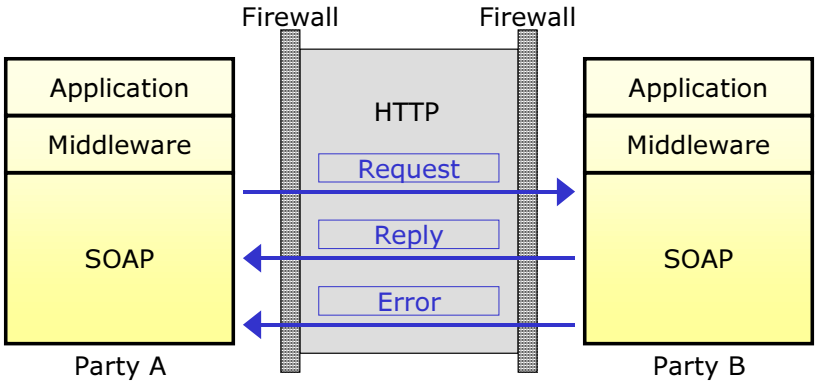

Figure 2-7: The anatomy of a SOAP call

The Envelope is the top element of the XML document representing the message. The element may contain namespace declarations as well as additional attributes. If present, such additional attributes must be namespace-qualified. Similarly, the element may contain additional sub elements. If present these elements must be namespace-qualified and must follow the Body element. An envelope has a header section and a body section. The header may have the attributes actor, encodingStyle, or mustUnderstand, and is encoded as the first immediate child element of the Envelope element. The body provides a simple mechanism for exchanging mandatory information intended for the ultimate recipient of the message. Typical uses of the Body element include marshalling RPC calls and error reporting. An example of a SOAP message requesting information regarding the whereabouts of a specified TI employee.

```
<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:WhereIs xmlns:m="www.telin.nl/whereis">
         <m:employee>Chris Vissers</m:employee >
         <m:format output=”longitude,latitude” />
      </m: WhereIs >
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The SOAP response message might look like

```
<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:ThereIs xmlns:m="www.telin.nl/whereis">
         <m:longitude>52.50</m:longitude>
         <m:latitude>4.95</m:latitude>
      </m:ThereIs>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Xmethods (http://www.xmethods.net/) maintains a list of SOAP-enabled services.

2.4.2 UDDI: Universal Description, Discovery, and Integration

UDDI is intended to be an Internet standard for creating an online business registry in a platform-independent way, and was introduced in August 2000 by Microsoft, IBM and Ariba. In May 2001, Ariba resigned and HP became a new UDDI operator. At the high level, the standard defines White Pages (general business information), Yellow Pages (business categories, taxonomies), and Green Pages (services offered by a business). Figure 2-8 shows how these white, yellow, and green pages are intended to evolve over time, until UDDI will be handed over to an open standardization body at the end of 2001.

Businesses can list and describe themselves in a registry that is maintained by a number of different companies. Microsoft has begun beta testing its implementation of the UDDI Business Registry, the same goes for IBM, and for HP it is not exactly clear. Organizations can register now at http://uddi.microsoft.com/register.asp, which promises no-cost registration and access to the UDDI repository. Using the registry, service requestors are able to discover businesses and their services, and to integrate these services.

Currently, a service requestor is able to find a business using business name, business location, or service name, but also using the different identifications and classifications (taxonomies). The identifications include the DUNS number, Thomas Supplier ID, or a custom identifier, and a business or service can be identified via its identifierBag.

The classifications include the North American Industry Classification System (NAICS), Universal Standard Products and Services Codes (UNSPSC), Standard Industrial Classification (SIC), IS0 3166 Geographic Taxonomy (country level), GeoWeb geographic classification (city level), and business or service can be classified via its categoryBag .

This way, the service requestor is able to—both manually and automatically—find a business situated in the Netherlands, selling products classified as voice recognition software, and offering a catalog service.

Version	1	2	3
White pages	Business Units	Corporations	Associations
Yellow pages	3 Taxonomies	More Taxonomies	Custom Taxonomies
Green pages	Descriptions of Services	Layered Services	Workflow
Due	September 2000	March 2001	December 2001

Figure 2-8: UDDI's white, yellow and green pages

The other promises of UDDI are that registration enables a company to

- Conduct business in any industry, around the world, using any platform.
- Look up the Web Services interfaces of business partners and potential partners.
- Discover technical details on working with other Web Services, and post details for its own services.
- Remove the discovery barriers to participating in the global Internet economy.

The core information model used by UDDI registries is defined in an XML Schema which defines four core types of information.

1) Business Entity (white pages): Information about the party who publishes information about a service. Contains businessKey, name, description, businessServices, categoryBag, and identifierBag
2) Business Service (yellow pages): Descriptive information about a particular family of technical services. Contains serviceKey, businessKey, name, description, bindingTemplates, categoryBag,
3) Binding Template (green pages): Technical Information about a service entry point and construction specifications. Contains bindingKey, serviceKey, description, and accessPoint.
4) tModels : Descriptions of specifications for services or taxonomies. These form the basis for technical fingerprints. Binding template data contains references to tModels. These references designate the interface specifications for a service. Contains name, description, overviewDoc, categoryBag, and identifierBag.

A new standard, named WS-inspection, has been proposed by Microsoft and IBM to allow an application to interface with a Web site in order to find out what Web Services a particular organization offers. Microsoft and IBM are touting WS-Inspection as complementary to Universal Description, Discovery and Integration, and in a sense it is, but it is also a decentral competitor of the central UDDI approach. For parties with existing relationships, WS-Inspection might help to discover each other's Web Services, whereas UDDI can be used to

get a listing for Web Services that match the need of a party as formulated in a query. The following figure explains how Microsoft and IBM see the relation between UDDI and WS-Inspection [WSInsp].

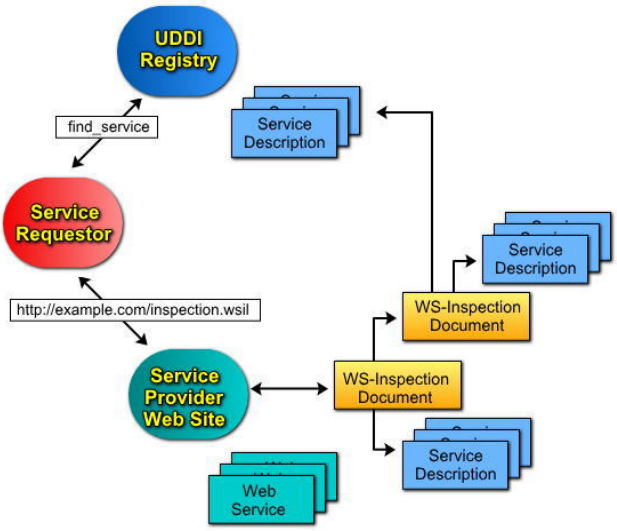

Figure 2-9: Relation between UDDI and WS-Inspection

The WS-Inspection specification contains two primary functions: an XML format for listing references to existing service descriptions, and a set of conventions so that it is easy to locate WS-Inspection documents.

2.4.3 WSDL: Web Service Description Language

Once a service requestor has found an appropriate service provider in the UDDI registry, the next thing he wants to know is how the service(s) offered by the provider can be invoked. In other words, he needs an interface definition of the service specified in a well-known language. WSDL is such a language.

WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. Its specifications are the result of a joint effort from Microsoft, IBM and Ariba. WSDL is an important factor in the development of SOAP, and it facilitates the interoperability of Web Services. An increasing number of SOAP implementations support this description language. Thanks to WSDL, SOAP implementations can self-configure exchanges between Web Services while masking most of the technical details.

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 which are abstract collections of operations. The concrete protocol and data format specifications for a

particular port type constitutes 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 elements in the definition of network services:

- Types: 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.

The first five elements are described in a reusable *service interface definition*, which is typically defined by industry standards organizations such as RosettaNet, and the last two elements are described in a service-specific *service implementation definition*.

Xmethods is a service provider which offers the following services: a Barnes and Noble Quote Service (returns book price from Barnes and Noble online store, given ISBN), a Pacific Bell SMS Service (sends a text message to a subscriber on the PacBell SMS network), a Delayed Stock Quotes Service (20-minute delayed stock quotes), and a Currency Exchange Rates Service (returns exchange rates between 2 countries' currencies). Xmethods can be found using UDDI, and the UDDI registry provides a link to the WSDL file describing the currency exchange rate service. The WSDL specification is printed in the text box below. We see that this service basically offers only one simple operation named GetRate. To get the rate, a message should be send with the name getRateRequest containing the two countries between which the exchange rate is to be retrieved. Upon receipt of this message, the service responds with a message named getRateResponse containing the desired exchange rate. These two messages are encoded in SOAP RPC style. All the above is specified in the following WSDL file.

```
<?xml version = "1.0"?>
<definitions name = "XMethods Currency Exchange"
targetNamespace = "http://www.xmethods.net/tmodels/CurrencyExchangeRate.wsdl"
xmlns:tns="http://www.xmethods.net/tmodels/CurrencyExchangeRate.wsdl"
xmlns:xsd = "http://www.w3.org/1999/XMLSchema"
xmlns:soap = "http://schemas.xmlsoap.org/wsdl/soap/"
xmlns = "http://schemas.xmlsoap.org/wsdl/">
        <message name = "getRateRequest">
                <part name = "country1" type = "xsd:string"/>
                <part name = "country2" type = "xsd:string"/>
        </message>
        <message name = "getRateResponse">
                <part name = "Result" type = "xsd:float"/>
        </message>
        <portType name = "CurrencyExchangePortType">
                <operation name = "getRate">
                        <input message = "tns:getRateRequest" name = "getRate"/>
                        <output message = "tns:getRateResponse" name = "getRateResponse"/>
                </operation>
        </portType>
        <binding name = "CurrencyExchangeBinding" type = "tns:CurrencyExchangePortType">
                <soap:binding style = "rpc" transport = "http://schemas.xmlsoap.org/soap/http"/>
                <operation name = "getRate">
                        <soap:operation soapAction="http://xmethods.net/CurencyExchangeRate/#getRate"/>
                        <input>
                                <soap:body use = "encoded" namespace="…" encodingStyle = "…"/>
                        </input>
```

```
                    <output>
                        <soap:body use = "encoded" namespace = "…" encodingStyle = "…"/>
                    </output>
                </operation>
            </binding>
</definitions>
```

2.4.4 WSFL: Web Services Flow Language

So far, things can be readily implemented (more or less). The next step however is the definition of the orchestration of, flow of, process between of Web Services, and unfortunately, things are not so easy to implement here.

Several process modeling languages exist: the business process modeling language of BPMI, Microsoft's X-Lang, IBM's WSFL, HP e-speak WSCL, and RosettaNet's PIPs. However, ebXML BPSS probably tries to settle as the most generic modeling language for business processes. BPML is more focused on standardization of business-internal processes. X-Lang is focused on workflow modeling to support Biztalk server. The purpose of WSFL is to define the composition of Web Services (for WSFL, see the discussion in section 2.4.4). And RosettaNet has no formal process specification language yet, and is focused on the IT and electronic components industry. We included WSFL in the hype stack because it reuses WSDL concepts, but it is not really clear—at this moment—which process modeling scheme is best suited for Web Services. To compare the different process modeling languages, we make use of the following scheme:

Language	Initiator	Expressive power	XML syntax	Operational semantics	Public/ private	Vendor specificity	Concrete instances
WSFL	IBM	Medium	Yes	Yes	Public	Medium	Few
WSCL	HP	Low	Yes	No	Public	Medium	Some
XLANG	Microsoft	Medium	Yes	Yes	Both	High	Many
BPSS	ebXML	Medium	Yes	Yes	Public	Low	Very few
BPML	BPMI	Medium	Yes	Yes	Private	Low	No
PIP	RosettaNet	High	No	Yes	Public	Low	Many

Table 2-1: Business process modeling language comparison scheme

The scheme shows per business process language the initiator, its expressive power, whether or not the process definition is specified in XML, whether or not the operational semantics are described, if it describes public or private business processes or both, how vendor specific it is, and finally, how many concrete instances of process description we found in this language. For a more complete comparison of these languages we refer to [Lankhorst02].

The Web Services Flow Language (WSFL) is an XML language for the description of Web Services compositions WSFL considers two types of Web Services compositions:

- The *usage pattern* of a collection of services. This describes the way in which these services together realize a particular business goal, and specifies the execution sequence of the services' functionality. In WSFL, this usage pattern is known as a *flow model*.

- The *interaction pattern* of a collection of services. This specifies how the individual Web Services are connected to one another. In WSFL, the model describing the interconnection of services is known as a *global model*.

By specifying both the internal flow and the external connections between Web Service interfaces, WSFL neatly complements WSDL, which describes the service interfaces themselves, and their protocol bindings. WSFL also relies on an envisioned "endpoint description language" to describe non-operational characteristics of service endpoints, such as quality-of-service properties.

WSFL is still in its infancy, but deserves our attention since orchestration and aggregation of Web Services are a hot topic to build Web Service based applications.

2.5 BPSS: Business Process Specification Schema

In May 2001, ebXML delivered the final version of their Catalog of Common Business Processes and the Business Process Specification Schema (BPSS). Both documents are available at the ebXML web site [ebXML].

The Catalog of Common Business Processes is a list of generic business process names that can be used across various industries. Cross-references are supplied with other common industry standards including RosettaNet PIPs, Edifact, X12, and xCBL 3.0. It also includes a limited number of industry specific business processes plus cross references, and descriptions of common business processes.

In fact, the BPSS is a process specification schema, and there are no real examples of process definitions yet, but still it is worth to look in a bit more detail into the BPSS.

In the ebXML view, a business process consists of transactions and collaborations. For example, the business process procurement consists of Create Long Term Contract, Forecast Component Requirements, Send Planning Documents, Place Order, Ship Materials, and Arrange Payments. Hence, the specification schema supports the specification of business transactions and the choreography of business transactions into business collaborations. Each business transaction can be implemented using one of many available standard patterns. These patterns determine the actual exchange of business documents and business signals between the partners to achieve the required electronic commerce transaction. Figure 2-10 depicts the BP layer in relation to the other layers in the ebXML stack.

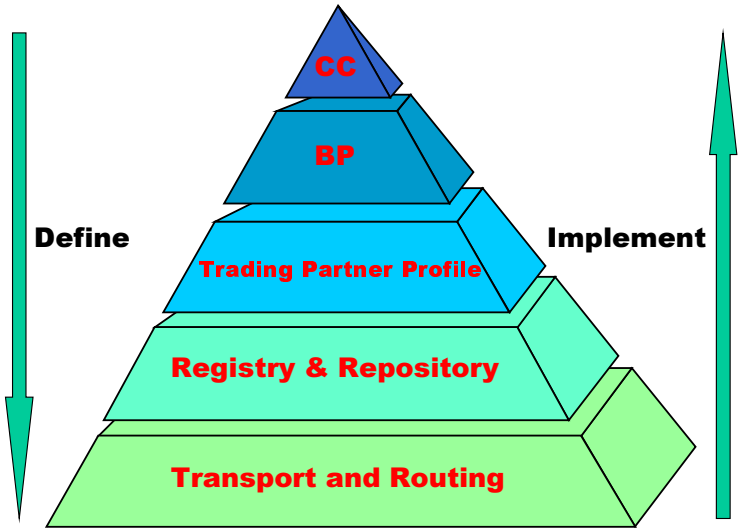

Figure 2-10: ebXML Business Processes (BP) in relation to the other ebXML layers (CC stands for Core Components, the standard patterns of e-business)

To give an impression of how business processes are modeled in a specification scheme, Figure 2-11 depicts the global ebXML context for business processes.

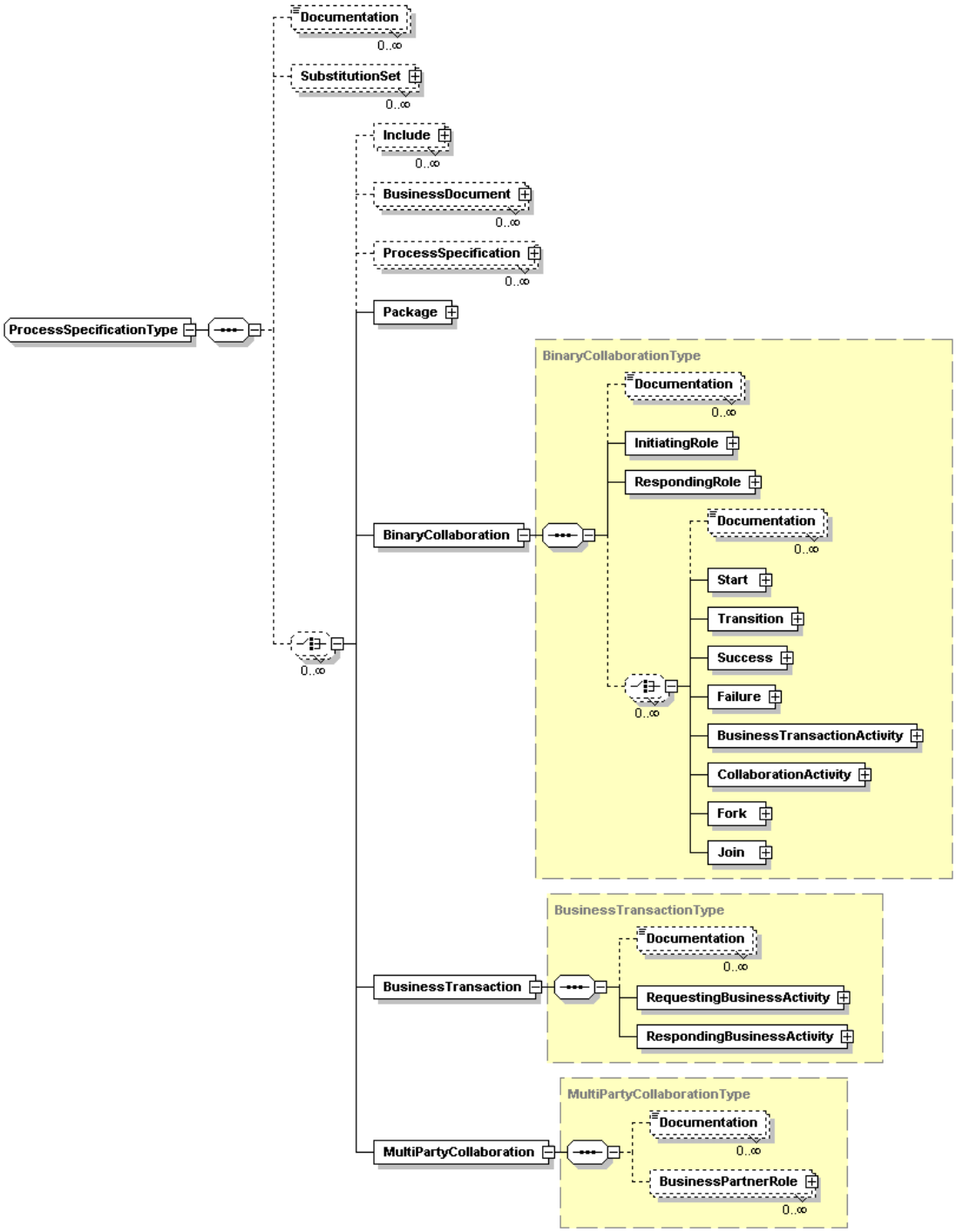

Figure 2-11: global structure of the business process specification scheme.

2.6 TPAML: Trading Partner Agreement

In January 2000, IBM submitted a specification for defining and implementing electronic contracts to OASIS, a vendor-neutral standards body. The XML-based specification, called tpaML (Trading Partner Agreement Markup Language), now is being handled by ebXML

Figure 2-12: The main functions provided by tpaML.

Figure 2-12 displays the main functions of a trading partner agreement. Overall properties of the TPA include its name, starting and ending dates, and similar global parameters. The role section provides the means to define a TPA in terms of generic roles such as airline and hotel and to produce a specific instance of the TPA by substituting specific parties for the role parameters. The identification section specifies the organization names of the parties and various contact information such as e-mail and postal service addresses. It also optionally specifies an outside arbitrator to be used for settling disputes. Communication and security properties include communication protocol (e.g. HTTP, SMTP), communication addresses, authentication and non-repudiation protocols, certificate parameters, etc.

For each party that can act as a server, there is an action menu which lists the actions that the other party can request, and various characteristics of those actions. Sequencing rules specify the order in which actions can be requested on each server. Error handling rules are various conditions related to error conditions, such as the maximum waiting time for the response to a request.

It seems that especially the lower layers of the tpaML (communications and actions) have been overtaken by rapid developments in the SOAP, UDDI and WSDL camp. It looks like there has been not so much activity around tpaML since the promising start in mid 2000. The important ideas and concepts have been adopted by ebXML's CPP and CPA (Collaborating Protocol Profile and Agreement, respectively).

2.7 XAML: Transaction Authority Markup Language

The Transaction Authority Markup Language defines the transactional interaction among Web Services, based on interfaces as defined by XA (an interface used to support global transactions across different transaction manager domains) and by JTA (the JAVA Transaction API). The initiative was taken by Bowstreet, HP, IBM, Oracle and SUN in October 2000. A business transaction usually requires the co-ordination of the Web Services of multiple companies (e.g. a vendor, an insurance company, and a carrier) and the single business transaction requires the commitment of the three underlying Web Services before

the transaction can be executed. XAML provides a mechanism for co-ordinating and processing the underlying Web Services to complete the entire transaction [XAML].

In the XAML white paper, XAML is presented as the missing link for plug and play e-commerce, which will precipitate a tsunami of e-commerce activity. However, the XAML specification is still being written in May 2001, whereas the promise was to deliver the specification in January 2001, so not much of a tsunami in delivering the specs. When the specs are finished they will be handed over to a—not yet determined—open standardization body, but it is not clear how much activity is still going on in XAML.

2.8 Wrapping up

Web Services are an exciting new concept in building and integrating applications. Except for the top layer of the protocol stack (see section 2.4), which defines the orchestration of Web Services, all other layers have developed to a level at which they can be readily implemented. So, from a technology point of view a version 1.0 of Web Services is under reach. Still, a number of (mainly non-technical) hurdles remain to be tackled before Web Services can really take off. These hurdles are the subject in the next chapter.

3 Web Services: benefits and obstacles

3.1 Benefits of Web Services

We will now review the benefits of Web Services in more detail. Certainly, each benefit also has its counterpart, but these are set aside and will be presented in section 3.2.

Promotes interoperability by minimizing the requirements for shared understanding
An XML-based service description (WDSL, section 2.4.3), which may be based on other e-business XML standards (e.g. RosettaNet, section 1.5), and a protocol of collaboration and negotiation are the only requirements for shared understanding between a service provider and a service requester. By limiting what is absolutely required for interoperability, collaborating Web Services can be truly platform and language independent, and can be implemented using a large number of different underlying infrastructures.

Enables just-in-time, dynamic integration
Collaborations in Web Services are bound dynamically at runtime. A service requester describes the capabilities of the service required and uses the service broker infrastructure to find an appropriate service. Once a service with the required capabilities is found, the information from the service's WSDL document is used to bind to it. Dynamic service discovery and invocation (publish, find, bind) and message-oriented collaboration yield applications with looser coupling, enabling just-in-time integration of new applications and services. This in turn yields systems that are self-configuring, adaptive and robust with fewer single points of failure.

Reduces complexity by encapsulation
A Web Services may be the aggregation of a collection of other Web Services. What is important is the type of behavior a service provides, not how it is implemented. A WSDL document is the mechanism to describe the behavior encapsulated by a service. Encapsulation is key to

Coping with complexity. System complexity is reduced when application designers do not have to worry about implementation details of the services they are invoking.

Flexibility and scalability. Substitution of different implementations of the same type of service, or multiple equivalent services, is possible at runtime.

Extensibility. Behavior is encapsulated and extended by providing new services with similar service descriptions.

Enables interoperability of legacy applications
By allowing legacy applications to be exposed as Web Services, the Web Services architecture easily enables new interoperability between these applications. Also existing directory technologies, such as LDAP, can be wrapped to act as a Web Service. A service-

oriented architecture would greatly facilitate a seamless integration between heterogeneous systems. New services can be created and dynamically published and discovered without disrupting the existing environment.

Moves from software to services.
Web Services will also change people's perception of how software should be paid for. Rather than paying an annual license fee to use an application or a component on an intermittent basis, people will become more comfortable using a pay-as-you-go model that is akin to the model used in the utilities industries.

Bridges the gap between implementation and definition
For the first time, the skill sets of the business manager and the application developer will converge. The manager will be enable to define service interface definitions and business processes, which can be used directly by the application developer as input for the implementation definition.

3.2 Hurdles for Web Services

Whereas interoperability, modularity and reuse of components are three of the most prominent advantages of Web Services, the corresponding drawback is the high interdependency between essentially black box components. Other hurdles to overcome are version management, performance, and reliability. This section elaborates on these hurdles for the application of Web Services in real-world scenarios.

Interdependency
Since the components of an aggregated Web Service can again be (smaller) Web Services, the aggregated Web Service depends on the proper functioning of the smaller Web Services. When unexpected results are encountered from the Web Services, it will be a difficult task to trace the origin of the problem. A Web Service is essentially a black box for which the user has no way to control or inspect, maybe only by requesting the service and see if it delivers the desired output.

In software development, tools are available for debugging purposes, but development and maintenance of distributed Web Services requires a complete new range of debugging and monitoring tools. For example, when a Web Service X unexpectedly stops its operation, the services relying on X must have ways to detect that service X is down, rather than trying to use X and conclude that something went wrong.

Malfunctioning may be caused by a server which is down for maintenance, by an unexpected change in configuration of the Web Service, by a change in the underlying database structure, by a new version of a component which is inadvertently installed on the server, et cetera.

To some extent these problems are relieved by dynamic service discovery which enables finding a replacement service for the inactive or malfunctioning service. Dynamic service

discovery systems will be self-configuring, adaptive and robust with fewer single points of failure, but how to detect that a service is malfunctioning is still an open question.

Error handling is a very important issue in Web Services. The focus is too much on what can be won when everything operates smoothly, and less on what can be lost if something goes wrong in the highly distributed Web Service environment.

Semantics
Compared with the choice for a transport protocol or a service definition language, the choice for a messaging standard is a rather complex one for the service provider. He has the choice between offering different interfaces for the most important messaging standards in his business context, or offering only one interface for a specific messaging standard and leaving the semantic mapping effort to the service requestors. But in either case, mappings have to be made between the in-house format and the public messaging standard, which can require substantial implementation efforts for complex standard, such as RosettaNet.

Version and Change Management
New and very clear policies must be developed with respect to version management of Web Services. A new version might result in new configuration details (see the previous paragraph), but this is not necessarily the case. Even when no changes are expected in the offered functionality, a new way must be found to communicate the new version to the service user, to indicate how long older versions of the service will be supported, et cetera.

Quality Management of the Registries
Ideally, the quality level of the information stored in the registries must be high to enable well-functioning Web Services. In practice however, the UDDI registry is not supervised and any party can add entries to the registry. For example, it is not guaranteed that the business's listing and technical details regarding HP are really supplied by the HP management. The situation is comparable to the early registry of Internet domain names: anyone could claim www.hp.com. In our opinion, the UDDI registries must introduce some kind of supervision with the intention to check if a party is entitled to add information to the registry, and to validate the quality of the offered information.

The uddi.microsoft.com site will monitor and audit all publication activity at its site. A level 1 publisher account is used by individual businesses and organizations registering at the uddi.microsoft.com site. As required by the UDDI operator specifications, these accounts have restrictions placed upon them in terms of the number of registry entries that may be published by an individual account. Each level 1 publisher may publish 1 Business Entity with 10 Service Types, maximum. Verification and maintenance of the accuracy of the information registered within the UDDI registry is the sole responsibility of the publisher.

Performance
Currently, companies can get a handle on their performance issues. Bottlenecks are typically code that could be optimized, a need for more servers, or a need for a faster Internet connection. Additionally, companies have come to know their business traffic by the

occurrence of special events, time of year, time of the month, time of the day, etc. Many companies make plans so that when the heavy periods of traffic arrive, they will have sufficient resources to handle the load.

But Web Services will introduce new bottlenecks. At the moment, the Internet offers no guarantees for available bandwidth at a certain moment in time. However, new versions of the Internet protocol will include the notion of quality of service. Still, there is no way to predict when heavy times will hit for a specific Web Service. Beforehand, it is unknown what the upper capacity limit of a Web Service might be. Benchmarking and evaluation tools may have a new opportunity here, as well as monitoring tools, which report violations in the service level agreement of a Web Service.

Reliability and trust

When Web Services become popular for outsourcing certain pieces of functionality to dedicated service providers, it will be unavoidable that the first failures get a lot of attention. A service requester will start to balance the advantages of outsourcing (no implementation costs, low maintenance costs) against the reliability and flexibility of a Web Service. So these two elements are essential for a successful long-term operation of Web Services. The first Web Services are already mushrooming, but the focus is mainly on presence. Reliability and flexibility are only a second goal, whereas they should be key issues.

Payment

Who is going to pay for Web Services? Solutions include the micro-payments model for small Web Services (might be difficult to implement), the advertising model (only applicable for a special range of Web Services), the subscription model for heavy users, or the package model for Web Services which are part of a larger business service. Banks providing their own financial Web Services to charge a bank account with the service costs, would be a promising step to a solution [Kuebler01]. But the question how to make Web Services profitable will be a hard one to tackle. We expect that now business oriented web services emerge, e.g. MySap.com, the subscription model will be a feasible one for these early adopters.

4 Conclusion

To conclude, we will summarize why we think that Web Services will take off, and what the most important barriers are for Web Services to become a global success.

4.1 Pro

- Web Services hide the implementation details of the underlying business function. New promising standards have been developed for service description, discovery and invocation.

- Integrate (legacy) applications by wrapping them as Web Services. This way, Web Services might contribute in the hard-to-tackle problem of legacy integration.

- Web Services make real distributed applications possible via open, web-based standards including XML, SOAP and WSDL. For example, Microsoft is betting the farm on Web Services, and .NET is the platform that supports these and aims to integrate all Microsoft's products. And that is certainly a sign to follow the developments in these areas with more than average interest.

- Speed of development of Web Service technologies. More and more open standards are agreed upon and ready to be applied in Web Services. The time line for the hype stack technologies is shown in Figure 4-1. It shows that most of the Web Service technologies were conceived in the last three years.

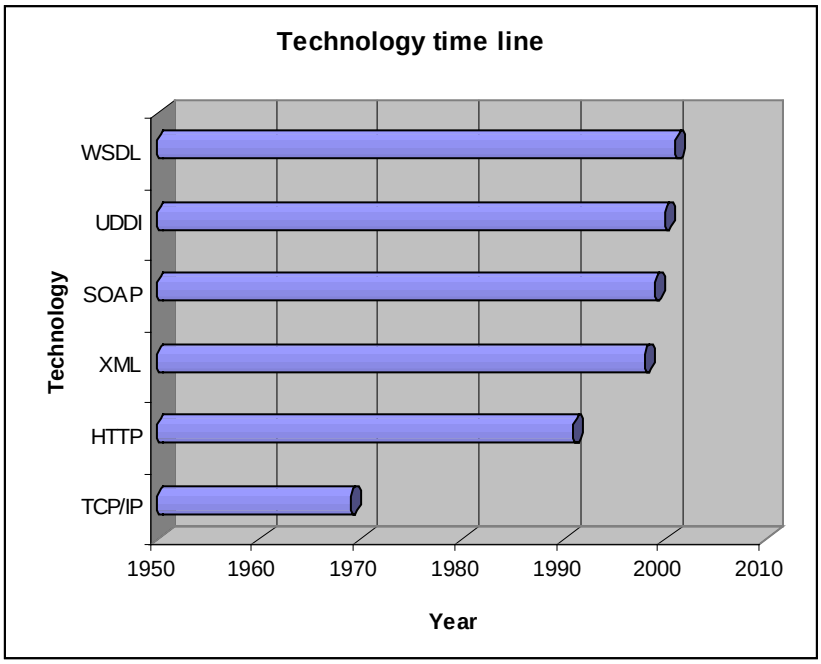

Figure 4-1: Year of introduction of technologies in the Web Services hype stack.

4.2 Con

- A distributed world is also a world full of interdependencies. This might be alleviated by the fact that Web Services are invoked dynamically, and can be substituted for each other.

- Although the technology is maturing, ‘social’ aspects are hard to predict. Too low a quality of data, performance, or reliability are potential dangers for Web Services.

- Standardization bodies should present service interface definitions for their messaging standards. At this moment there are no reusable interface definitions available, so that application developers have to do this their own way, with all possible consequences.

- Definition of Web Service processes is still an open issue. Quite a number of process definition standards are available, but none of them is really suited for application in Web Services at this moment.

- Web Services struggle with the all-for-free doom of the web. Payment of Web Service will be more and more an issue as the value of the offered business functions increases. The old advertisement model of the web does no longer suffice for application to application communication, so other payment models are needed.

References

Web links

[Aberdeen]	http://www.aberdeen.com/
[Babelfish]	http://babelfish.altavista.com/
[ebXML]	http://www.ebxml.org/
[GigaTS]	http://gigats.telin.nl/
[RosettaNet]	http://www.rosettanet.org/
[SOAP]	http://www.w3.org/TR/SOAP/
[TPAML]	http://www-106.ibm.com/developerworks/library/tpaml.html
[UDDI]	http://www.uddi.org/
[WebMethods]	http://www.webmethods.com/
[WSDL]	http://msdn.microsoft.com/xml/general/wsdl.asp
[WSDL2]	http://www.oasis-open.org/cover/wsdl.html
[WSInsp]	http://www-106.ibm.com/developerworks/webservices/library/ws-wsilover/
[XAML]	http://www.xaml.org
[xCBL]	http://www.xcbl.org/
[XML]	http://www.w3.org/TR/2000/REC-xml-20001006
[XML02]	http://www.xml.org/xml/industry_industrysectors.jsp
[XP]	http://www.w3.org/2000/xp/
[XSL]	http://www.w3.org./Style/XSL/

Literature and presentations

[Clark01]	J. Clark, *Comparison of SGML and XML*, W3C, http://www.w3.org/TR/NOTE-sgml-xml
[Colan01]	Mark Colan, *Introducing Web Services*, IBM, March 2001, http://www-106.ibm.com/developerworks/speakers/colan/
[Dan01]	A. Dan et al., *Business-to-business integration with tpaML and a business-to-business protocol framework*, IBM Systems Journal, Vol. 40, No. 1, 2001.
[Gilpin01]	M. Gilpin, *Integration: Enabling E-Business*, 1999 http://eai.ebizq.net/enterprise_integration/gilpin_1.html
[Janssen01]	W. Janssen, M. Stefanova, J.W. Koolwaaij, *XML: hype or hope?,* GigaTS/D2.2.4, Telematica Instituut, 2000
[Janssen02]	W. Janssen et al., *State of the art in e-business services and components*, GigaTS/D2.1, Telematica Instituut, 2000

[Koolwaaij01] J.W. Koolwaaij, P. van der Stappen, *ERP, XRP and EAI in virtual marketplaces*, GigaTS/D2.2.8, Telematica Instituut, 2000

[Kirtland01] Mary Kirtland, *A Platform for Web Services*, Microsoft, January 2001, http://msdn.microsoft.com/library/techart/websvcs_platform.htm

[Kuebler01] D. Kuebler and W. Eibach, *Metering and accounting for Web Services*, IBM, 2001.

[Lankhorst01] M. Lankhorst et al., *Mapping the e-business landscape*, GigaTS/D2.2.10, Telematica Instituut, 2001

[Lankhorst02] M. Lankhorst, *RSD and Web Service concepts*, GigaTS internal, Telematica Instituut, 2001.

[Linthicum01] David S. Linthicum, *Enterprise Application Integration*, Addison-Wesley, 2000

[Peltz01] Chris Peltz, *Interacting with services on the web*, HP, March 2001, http://devrsrc.external.hp.com/devresource/Docs/TechPapers/Eservices/e-services1.pdf

[Wilkes01] L. Wilkes, Business Integration, A CBDi Forum Report Produced for IBM, May 2000.

Appendix A - Web Service platforms

XMLbus (Iona)

IONA XMLBus is a complete and easy to use platform for Web Services development and deployment. This comprehensive Web Services infrastructure enables integration of applications over Internet, Intranet and Extranet; plus interoperability among distinct programming platforms such as J2EE and .NET.
http://www.iona.com/products/xmlbus_home.htm

GLUE (The Mind Electric)

GLUE is a platform that simplifies and unifies traditional distributed computing with the emerging world of Web Services. Using GLUE, you can build, deploy and invoke networked services. And because GLUE is based on standards such as HTTP, XML, SOAP, WSDL and UDDI, GLUE interoperates with Microsoft .NET, IBM WSTK, Apache SOAP and other Web Services platforms.
http://www.themindelectric.com/products/glue/glue.html

CapeConnect (CapeClear)

The CapeConnect Web Services Platform automatically and dynamically generates Simple Object Access Protocol (SOAP) eXtensible Markup Language (XML) from any J2EE, EJB, or CORBA component. The generated XML can be exposed as a Web Service on the Internet. CapeConnect also enables the resulting Web Service to be easily customised and composed with other Web Services.
http://www.capeclear.com/products/capeconnect/index.shtml

.NET My Services (Microsoft)

As part of the Microsoft® .NET initiative, Microsoft is introducing a user-centric architecture and set of XML Web Services, code named HailStorm and the formal name .NET My Services, which is oriented around people, instead of around a specific device, application, service, or network. They put users in control of their own data and information, protecting personal information and providing a new level of ease of use and personalization. The HailStorm services take advantage of the .NET technologies and architecture that make it possible for applications, devices, and services to work together. These services make user consent the basis for who can access user information, what they can do with it, and how long they have permission.
http://www.microsoft.com/myservices/

WASP (Idoox)

WASP Advanced contains modules that are necessary for enterprise Web Service development and deployment. It significantly eases the corporate wide implementation of the Web Services Architecture. WASP Advanced serves as the secure runtime engine with support for underlying legacy technologies (EJB, CORBA etc.). Many of these runtimes may be indexed and referenced by corporate Web Service master index engine - the WASP UDDI.
http://www.idoox.com/products