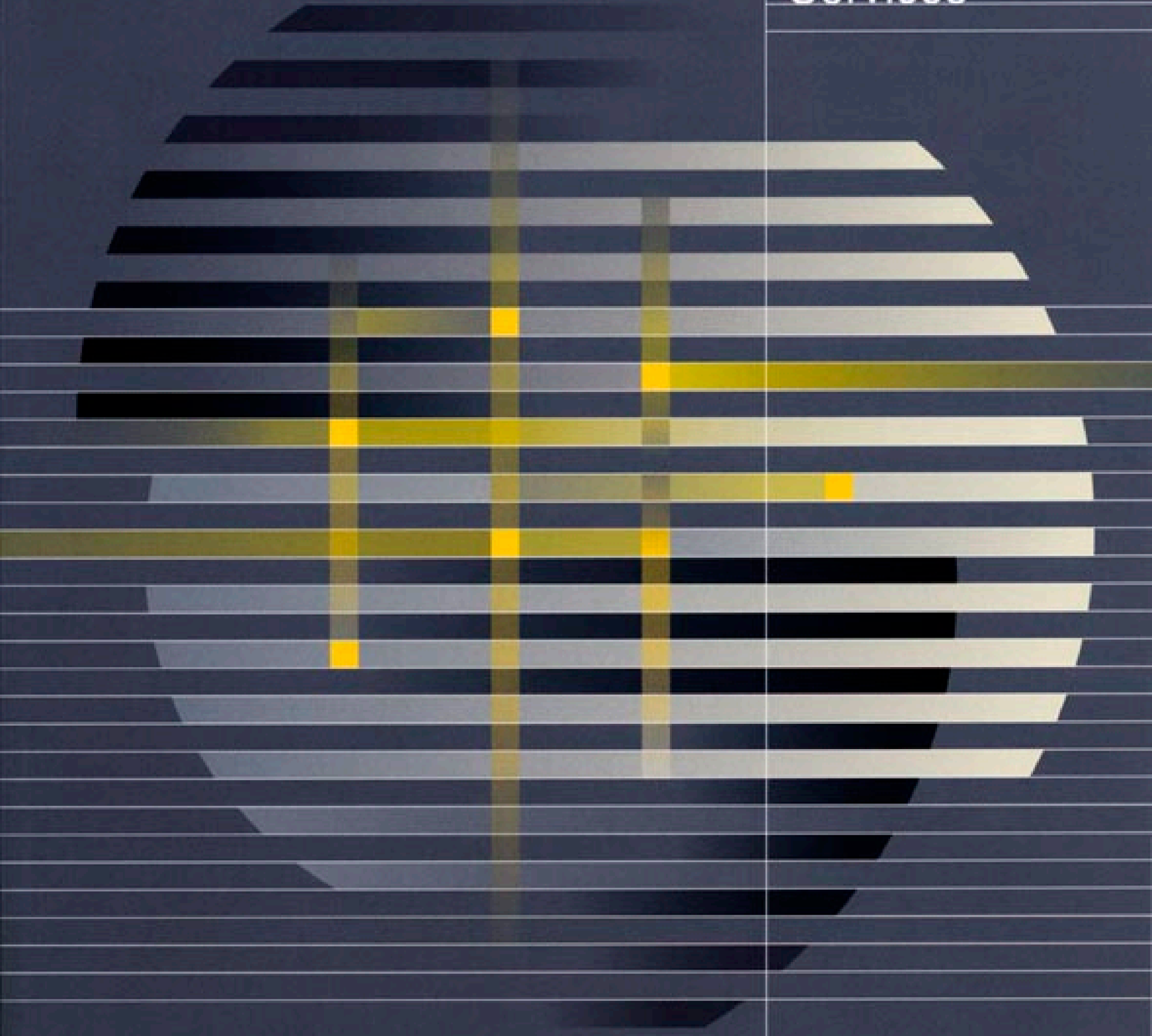
Telektronikk

4.2002

XML Web Services



Telektronikk

Volume 98 No. 4 – 2002
ISSN 0085-7130

Editor:
Ola Espvik
Tel: (+47) 913 14 507
email: ola.espvik@telenor.com

Status section editor:
Per Hjalmar Lehne
Tel: (+47) 916 94 909
email: per-hjalmar.lehne@telenor.com

Editorial assistant:
Gunhild Luke
Tel: (+47) 415 14 125
email: gunhild.luke@telenor.com

Editorial office:
Telenor Communication AS
Telenor R&D
NO-1331 Fornebu
Norway
Tel: (+47) 67 89 00 00
email: telektronikk@telenor.com

Graphic design:
Design Consult AS, Oslo

Layout and illustrations:
Gunhild Luke and Britt Kjus,
Telenor R&D

Prepress and printing:
Optimal as, Oslo

Circulation:
3,750

Contents

XML Web Services

telenor

Guest Editorial

DO VAN THANH

Do Van Thanh

When people say that they go on the net they usually mean the World Wide Web. For most people the Internet is usually associated with the World Wide Web, which is only one of the applications of the Internet, although a really popular one. Just before the current depression it was quite easy to get a job with decent salary with only a fair knowledge of HTML, the HyperText Markup Language of the World Wide Web. Everything related to the World Wide Web became hype. The most recent phenomenon is the XML Web service. All the vendors and analysts are talking about them. XML Web Services are becoming the next “new” and big thing in the Internet. It is considered as a new vision for distributed computing on the web offering an XML-based access via transparent Internet protocols (HTTP), relieving distributed computations from the need to be based on specialized middleware platforms, thus enabling the use of services over company borders, firewalls, different suppliers, infrastructures and technologies.

Nevertheless, most ironically, nobody seems to agree about the definition. So, what is an XML Web service? Does it require any particular transport like HTTP or SMTP? If so, can you use others, like MSMQ? Does it mandate the use of XML and SOAP or can other content types such as MIME, JPG, MP3, or URL-encoded data in a query string be used as well? However, the most basic questions are: What has XML to do with services? Is it just an extended Markup Language? Many questions are left without answer.

Anyway, XML Web Services should have some potential since big companies like Microsoft, IBM, Sun, HP BEA, Oracle, etc. have invested quite a lot of effort and money in them. Microsoft is betting and promoting fiercely the concept of Web Services. After its shift to supporting the Internet a few years ago, Microsoft is preaching Web Services through its .NET initiative. Bill Gates claimed that “The Power of the XML Web Services model is amazing”. In answer to Microsoft’s .NET, Sun announced the Sun ONE initiative that includes Web Services support in Sun’s Forte for Java Tools and in iPlanet’s Web, application and integration servers. IBM supports Web Services in its Websphere which consists of the Websphere application Server, the Web Services Development environment for building and deploying Web Services.

The XML Web Services concept is also gaining momentum in the Telecom world. Indeed, telecom operators started many years ago to look for a safe and efficient way of exposing and offering their network capabilities to third parties through standardisation activities in Parlay/OSA. The XML Web Services concept could be the ideal solution since it does not let itself stopped by firewalls or software incompatibility problems as distributed computing middleware like Corba. Projects aiming at investigating the feasibility of the XML Web Services concept and establishing testbeds for Web Services have been initiated by both the EU Commission and EURESCOM, the European Institute for Research and Strategic Studies in Telecommunications. In fact, some of the articles in this *Telektronikk* issue result from their projects. At Telenor, a workshop “XML Web Services: the Services of the Future” organised in June this year attended by Web Services experts from both academia and industry has gained a lot of attention and participation.

The main goal of this *Telektronikk* issue is to present a clear overview about XML Web Services, what they are, how to build and use them, what their benefits are, etc. It is also aiming to create an opportunity for worldwide experts to express their opinions and experiences about XML Web Services. The issue has some articles that cover the fundamentals of XML Web Services and can serve as an introduction to the concept. Other articles are aiming to provide a general understanding of Distributing Computing and explaining XML Web Services in the context of Distributed Computing. The major Web Service products and vendors

Front cover:
XML Web Services

The artist Odd Andersen sees various providers of XML Web Services as distinct parts of global market areas acting within the physical boundaries of bandwidth and network. Today they offer particular solutions – later they may co-operate or merge their provision into a common solution – here visualised by extending particular global areas into each other. The in- and outgoing lines – also crisscrossing within the web – indicate the user’s access to service components needed to fulfil his/her demand.

Ola Espvik, Editor in Chief

are also summarized in an article, such that a reader wishing to build Web Services can get started easily.

No matter how wonderful is the XML Web Services concept, it can never be a success without security, which is also covered in an article. How XML Web Services fit in the evolution of the telecom service creation is also explained. The business aspects of XML Web Services surely play a decisive role in their success.

The XML Web service scenarios for telcos are treated thoroughly. There are also articles that consider Electronic Commerce, Electronic Business and Electronic Gateways. This *Telektronikk* issue concludes with three applications of the Web Service concept, namely an infrastructure for electronic collaboration, a self-service channel and a user profile Web Service. Finally, I hope that the reader will find this issue both pleasant and useful.

Enjoy your reading!

Thanh

杜文成

What is an XML Web Service?

ERIK VANEM AND DO VAN THANH

XML Web Services are considered by many as the services of the future, but what exactly are Web Services? Is it simply services delivered via the World Wide Web? And what about XML, is it just a new and improved version of HTML and what does it have to do with services? This article will try to give a basic overview of XML Web Services and describe its most important features in a simple and comprehensible way. Before studying the concept of Web Services, the article introduces XML and describes how XML documents are created and parsed. It then proceeds with Web Services and how they are used. Finally before the conclusion, some more details about the standards associated with XML Web Services are given.

Erik Vanem (30) received his MSc in Physics from the University of Oslo in 1996. Before joining Telenor R&D in 2000 he worked as a geophysicist at PGS Reservoir, as a research assistant at the Norwegian Defense Research Establishment and as a physics teacher at the Oslo University College. Since 2000 he has worked with user centric services, mobile applications and services, Voice over IP and mobility management in next generation wireless networks. Recently he has been working with distributed computing and XML Web Services in the PANDA group (Personal Area Networks and Data Applications) at Telenor R&D.

erik.vanem@telenor.com

Do Van Thanh (44) obtained his MSc in Electronic and Computer Sciences from the Norwegian Univ. of Science and Technology (NTNU) in 1984 and his PhD in Informatics from the University of Oslo in 1997. In 1991 he joined Ericsson R&D Department in Oslo after 7 years of R&D at Norsk Data, a minicomputer manufacturer in Oslo. In 2000 he joined Telenor R&D and is now in charge of PANDA (Personal Area Network & Data Applications) research activities with a focus on SIP, XML and next generation mobile applications. He also holds a professor position at the Department of Telematics at NTNU in Trondheim. He is author of numerous publications and inventer of a dozen patents.

thanh-van.do@telenor.com

1 Introduction

Most people today are well familiar with the World Wide Web. As the name indicates it is a global web of servers (web servers) that are interconnected and that offer each other access to their files. In this way people may easily and rapidly exchange information from anywhere in the world. Traditionally, the WWW has mainly contained static information or content intended for the human user and presented to him through a web browser. The information accessible on the web servers has typically been contained in HTML documents that dictate how the actual presentation of the data will be and that contain links to other documents contained on other web servers. Lately however, the WWW have become more dynamic with frequent information updates and relocations. The WWW concept is also extended to contain not only content but also functionality; i.e. one can not only access information available on the WWW but also get certain functions performed. These functions can be spread out through the whole web, and be accessed by other applications with Internet access. To put it simple, the web is evolving from being basically static content available to human users to include also functionality or application components available to other applications.

Different solutions for distributed computing, allowing applications to communicate with other applications across computer boundaries, have been around for a while already. However, they all have a lot of shortcomings that limit their range of use. Object frameworks such as COM and CORBA coupled with wire protocols (IIOP, DCOM and RMI) were introduced to allow applications to talk to each other over a network. However, these solutions were not interoperable and even though there have been some efforts in the industry to amend this, they have not been widely successful. Being non-interoperable as these solutions are, it basically means that both sides of the communication link need to use the same distributed object model. This works fine in most cases where both sides belong to the same LAN and are controlled by the same organisation that can decide which systems to use. However, with the emergence of the Internet, the distributed networks became very large and very decentralized and it is generally no longer possible for anyone to control both ends of the communication link. This makes application-to-application communication and distributed computing over the Internet quite challenging with these protocols. Another difficult issue is firewalls. When communicating over the Internet, across company borders, one needs to pass through firewalls that generally do not have many open ports. Very often, the widely used ports for HTTP and SMTP are the only ones that are open.

So XML Web Services has emerged to remedy this situation and allow applications to communicate with other applications in a well-defined way. The various applications may reside on different machines anywhere on the Internet or within the same intranet and the communication should be able to traverse both company borders (firewalls) and technological borders (when the two applications are implemented on different platforms, with different language, etc.). One should keep in mind, however, that the XML Web Services proposal is not merely a substitute to replace the existing solutions for distributed computing. It rather supplements them with the exposition of services on the Web. A likely Web Services scenario will be made up of two levels. The first level is an internal system that implements the applications based on traditional platforms and communicating internally with traditional distributed computing technologies or some proprietary solutions. These applications can then be exposed externally using XML Web Services so that others are able to utilize them.

2 What is XML?

XML is one of the cornerstones of XML Web Services and the specifications associated with XML Web Services are all based on XML. In

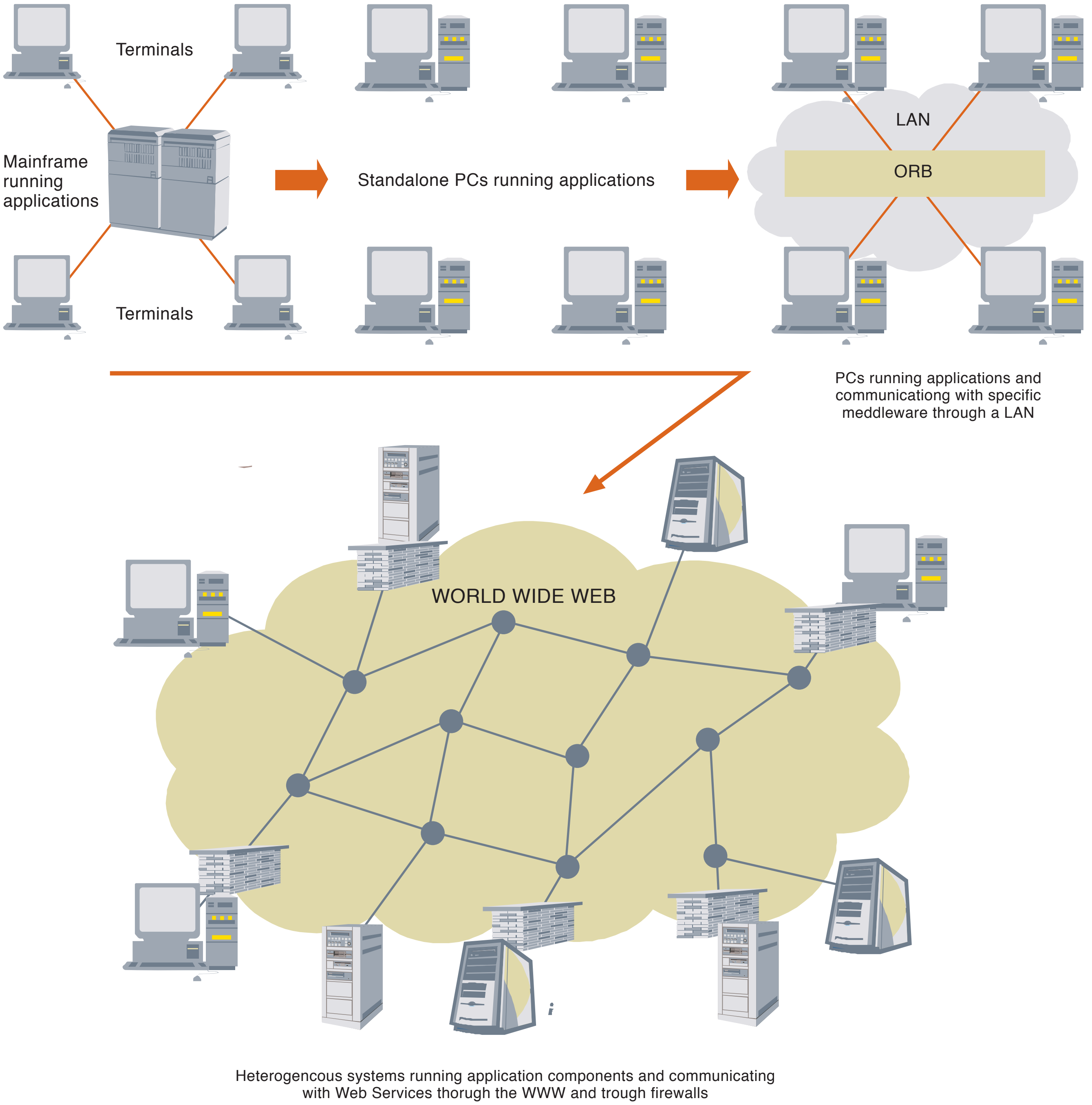

Terminals
Mainframe running applications
Terminals
Standalone PCs running applications
LAN
ORB
PCs running applications and communicatiing with specific meddleware through a LAN
WORLD WIDE WEB
Heterogencous systems running application components and communicating with Web Services thorugh the WWW and trough firewalls

Figure 1 Web Services as the next step in the evolution of distributed computing

this section the main features of XML will be outlined and it will be indicated why XML is a well-suited tool for Web Services. The W3C XML recommendations are available at [1].

What is XML? Since XML can be used in a variety of different areas of data computation and communication, different opinions exist of what XML is. So to keep things clear and avoid confusion, let us first say a few things about what XML is not.

- XML is not a programming language;
- XML is not a database;
- XML is not the next generation of HTML;
- XML is not specific to any horizontal or vertical market segment.

XML stands for eXtensible Markup Language and is a specialisation of SGML (Standard Generalized Markup Language), which has been an ISO standard since 1986. It is a clearly defined, system-independent way of representing data. With XML, all conceivable kinds of data can be structured, described and interchanged. An XML document is in text format and this makes it readable to both machines and human beings. Human being, of course, should not have to read the XML files, but it can be useful when there is a need. Finally, XML is non-proprietary and licence free so that anyone can build their own software around it without paying anything to anybody.

```
DTD
(Document Type Definition)

<!ELEMENT priceList (car) +>
<!ELEMENT car (name, price) >
<!ELEMENT name (#PCDATA) >
<!ELEMENT price (#PCDATA) >
```

```
XML

<?xml version=”1.0”?>

    <priceList>
    <car>
        <name>Ford</name>
        <price>150 000</price>
    </car>
    <car>
        <name>Nissan</name>
        <price>145 000</price>
    </car>
    <car>
        <name>Opel</name>
        <price>140 000</price>
    </car>
</priceList>
```

Figure 2 A simple XML example

The data in an XML file is enclosed in tags, much like HTML files. In the HTML specifications, however, it is specified what each tag and attribute mean, and one is limited to using only those tags that are predefined. XML on the other hand is extensible, meaning that everyone is allowed to write their own tags that describe the content. Another difference from HTML is that the tags in XML relate to the actual meaning of the enclosed textm whereas in HTML the tags most often define how the data will be presented by a browser. Since XML documents contain custom-made tags and attributes that may be specifically defined by the author of the document, there must be a way for others to know their meaning and thereby know how to interpret the enclosed text. This is done with an XML schema language, e.g. Document Type Definition schema language or XML Schemas 1 or 2, that define the tags in an XML document. A simple XML example is given in Figure 2 which shows what an XML file might look like, and with an example DTD file that defines the tags used in the XML document.

The example shows an XML file containing a pricelist for cars. First in every XML document is the XML prolog. As a minimum, every XML document must always contain prolog with a declaration that identifies the document as an XML document. This is the first line in the example document in Figure 2. In addition, optional information can be contained in the prolog. For example, specifications of which tags are valid in the document can be declared in a DTD directly within the prolog, or a pointer to some external specification file can be placed in the prolog.

Following the XML prolog is the document’s actual content or data, sometimes referred to as the Root Element. The root element is the highest-level element and this can again contain other elements in a hierarchical structure that is defined in the DTD. In the XML file this means that all other tags will appear between the <rootElement> and </rootElement> tags. The example file has <priceList> as its root element, and according to the DTD, this element contains one or more (indicated by the plus sign) <car> elements. Every <car> elements must in turn contain one <name> element and one <price> element in that order. The <name> and <price> elements are again the actual data to be parsed or analysed by the XML parser – PCDATA. In this simple example, DTD is used to illustrate how to define the tags or elements. A significantly more powerful, but also more complex language for doing this is XML Schema. For more details on XML Schemas, see [2].

With XML, the data is separated from the formatting instructions. To specify how documents are presented on a screen, in print, or how they are pronounced separate stylesheets are used. XSL is a language for expressing such stylesheets and consists of three parts; XSLT for transforming XML documents (e.g. XML data into an HTML/CSS document), Xpath to access or refer to parts of an XML document, and XSL-FO an XML vocabulary for specifying formatting semantics. For more details on XSL, see [3].

2.1 XML Data for Multiple Applications

XML documents can contain data that are meant for different applications that will need the data for different purposes. An important aspect of XML is that it is self-describing and that different parts of the data are identified. In the example above the parts of the data that contains the name of the cars are easily distinguished from the price. In this way the different parts of the data can be used in different ways by different applications. The applications have to agree on the tag names, of course, but if they have the right DTD (or XML Schemas), they can process the documents according to specified rules.

An XML document can also contain processing instructions of the following format:

<?**target instructions**?>

These processing instructions give information to an application that processes the XML data, where **target** is the name of the application and **instructions** contain the information. Because the instructions are application specific, XML files can have multiple processing instructions that tell different applications to do similar things in different ways.

XML is modular, meaning that one is allowed to define a new document format by combining and using other formats. When doing this, however, there is a possibility that the two formats have elements or attributes with the same name. For example if combining the XML example above with a format describing customers or drivers, it is likely that these will also contain a <name> element. To avoid mixing two different elements with the same name but different meaning, XML provides a namespace mechanism that is used to identify the different elements in a document. XML namespaces use prefixes to the elements to achieve this and in the above example, one can use the namespaces car and customer to distinguish between the different <name> elements. The qualifying name would then contain a namespace-prefix followed by a colon and the local name: <car:name> and <customer:name>. However, one must make sure that the prefixes one uses are globally unique and not used by anybody else; i.e. that two different namespaces never have the same name. This is obtained by using URIs as namespace names. These URIs could be controlled by an organisation so that others do not use them. In the above example, the following names could be globally unique:
<http://www.cardealer.com/car:name> and
<http://www.cardealer.com/customer:name>.
There will then be one XML schema bound to each specific namespace that defines all elements and attributes belonging to that namespace. For more about XML namespaces, see [4].

With a basic understanding of XML and how it can be used by multiple applications in different ways, it is about time to proceed and consider the Web Service concept. More interesting literature can be found at [5, 6].

3 What is a Web Service?

What then, is a Web Service? Several definitions of Web Services exist, and the following are just some examples:

- *Web Services are self-contained, modular applications that can be described, published, located, and invoked over a network, generally, the Web* – IBM [7].

- *A Web Service is a unit of application logic providing data and services to other applications* – Microsoft [8].

- *A Web service is a software application identified by a URI, whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via internet-based protocols* – W3C Web Services Architecture Working Group [9].

- *Web Services are modular, self-describing applications that can be published, located and invoked from just anywhere on the Web or a local network* [10].

- *Web Services are loosely coupled, reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over standard Internet protocols* – The Stencil Group [11].

Others may offer alternative definitions of Web Services and although the various definitions are not identical, they all share the same general view of what Web Services are.

Web services can be said to describe any computational functionality that can be found and invoked over any network in a component based model. These functionalities are reusable software components that allow other developers than the ones that have created them, to reuse them as building blocks of code and to assemble them and extend them in new ways. In this way Web Services continue the rising of object-oriented design in software development, and developers do not need to write a complete set of instructions from start to finish.

A Web Service represents self-describing and self-contained application meaning that it encapsulates discrete functionality that performs a single task. It describes its own inputs and outputs in such a way that other software can determine what it does, how to invoke it and what to expect as a result in return. Web Services are designed to be used by other programs rather than humans, so they do not have a graphical user interface. However, even though they are not designed for direct human interaction, they might be incorporated into software designed for human interaction. For example, a Web Service can be accessed by an application that generates an HTML page to be displayed in a browser.

The different Web Services software components are loosely coupled and do not depend upon a tight interconnection. The Web Services developers thus do not need to have full control over both ends of the connection and a much simpler level of coordination is required than with traditional design. The fact that the components are loosely coupled also means that more flexible reconfiguration is allowed. Finally, Web Services are distributed and can be accessed via transparent widely adopted Internet protocols like HTTP, SMTP, FTP, etc. By using these protocols, the same as traditional web content, Web Services can communicate through most firewalls, and can thus be used across company borders.

Once a Web Service is deployed, it can be found and accessed by other applications and other Web Services. A Web Service can be mixed and matched with other Web Services in a value chain allowing construction of more complex services out of multiple simple Web Services. This is illustrated in Figure 3.

The actual service logic of the different Web Services in Figure 3 can be implemented on different platforms and with different programming languages. They can also belong to different organisations or domains as the invocations and results are communicated via e.g. HTTP that can traverse firewalls.

3.1 XML Web Services

Web Services as described above should be universally accessible programmatically, regardless of what platform the Web Service runs on or what programming language it uses internally. This means that any application that is made to interact with a certain type of Web Services should be able to exploit any of those Web Service that is exposed on the Internet without any need for extra programming or adjustments. In order to achieve this, standard protocols for Web Services are important. XML has been most important as a way to standardise data formats and exchanging data and most of the standard protocols for Web Services use XML as its foundation. Having seen various definitions of Web Services, the following can be used as a definition of XML Web Services:

> *XML Web Services are Web Services that use XML as a means of standardising data formats and exchanging data.*

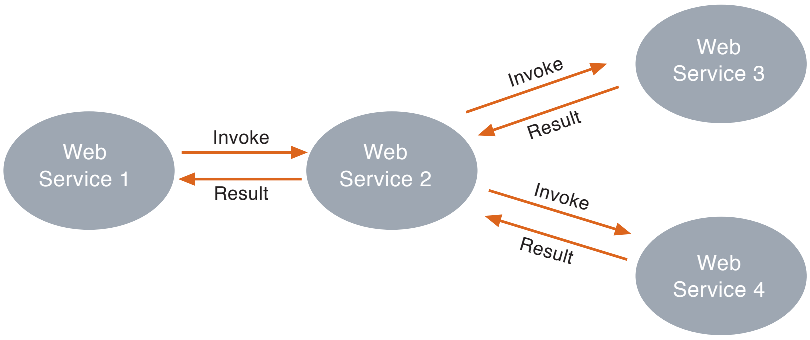


Figure 3 A value chain of Web Services

3.2 Three Distinct Roles in a Web Services Scenario

In an XML Web Services scenario, three distinct roles that interact with each other can be identified. These roles are the service provider, the service requestor and the service broker. Figure 4 illustrates these roles and their interactions.

The service provider creates the service and makes it accessible for clients over the Internet. The service can be created in any language on any platform as long as it supports the publish and bind interactions. A Web Service provider differs from an application service provider (ASP) in that the offered Web Services are distributed components. The ASP on the other hand delivers entire applications from a central hosting location, which are generally not extensible.

A service requestor is the client that uses the Web Service. This client can also be written in any language and on any platform as long as it is able to find and bind to a Web Service. A Web Service can itself take the role of a service requestor when it takes advantage of other Web Services as illustrated in Figure 3.

The service broker is the one that brings the service requestor and the service provider together by providing an interface between the **publish** and **find** interactions.

The interactions between these roles are the following:

Publish: The service broker offers a **publish** interface where the service provider can tell the service broker about the

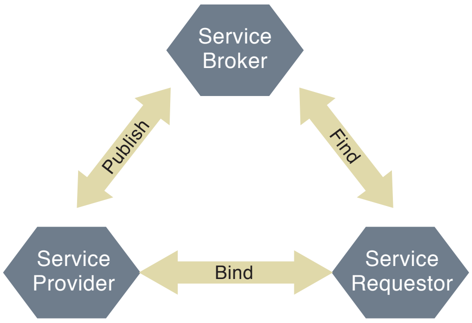


Figure 4 The three roles in a Web Services scenario and their interactions

services it provides. The published information should include data about the service itself, its input and output, and where it can be accessed. In addition, an **unpublish** interface is offered that lets the service provider remove advertisements of published services.

Find: In addition, the service broker offers a **find** interface where the service requestor can inquire about a particular Web Service or a category of Web Services, where services can be searched for by specifying a variety of search parameters.

Bind: The **bind** interaction represents the actual invocation of the Web Service. Binding in this context can be compared to a function call, where parameters are passed to the function and a return value is received as a result.

Neither the **find** nor the **bind** operations need to be performed every time a service requestor wants to invoke a service, however. If the service requestor has found a suitable service once, the results can be cached and used every time it needs to **bind** to this specific service. Only in cases where the **bind** operation fails, for example when the service has been moved to a different location or modified, will the service requestor need to inquire the service broker again and refresh the cached information.

3.3 Main Uses of Web Services

The three roles in Web Services scenarios identified above can all exist within the same enterprise domain or they can belong to different domains within the Internet. This results in a wide range of different areas where Web Services can be used. Basically, the main uses of Web Services can be divided into three categories:

- Application integration Web Services
- Business integration Web Services
- Commercial Web Services

Normally, the first type of Web Services an enterprise would implement is application integration Web Services (Web Services will be a suitable technology for Enterprise Application Integration, EAI). These are Web Services used internally within an Intranet to integrate different business applications that run on different platforms. Often, different systems like for example Unix and Windows coexist within an enterprise, and Web Services can enable them to communicate with each other. Legacy applications can expose part of their interfaces as Web Services to allow integration with other business applications in a heterogeneous environment without the need to rewrite a huge amount of code.

The next step is usually business integration Web Services that integrate applications between company borders. A few key partners outside the company can be chosen and Web Ser-

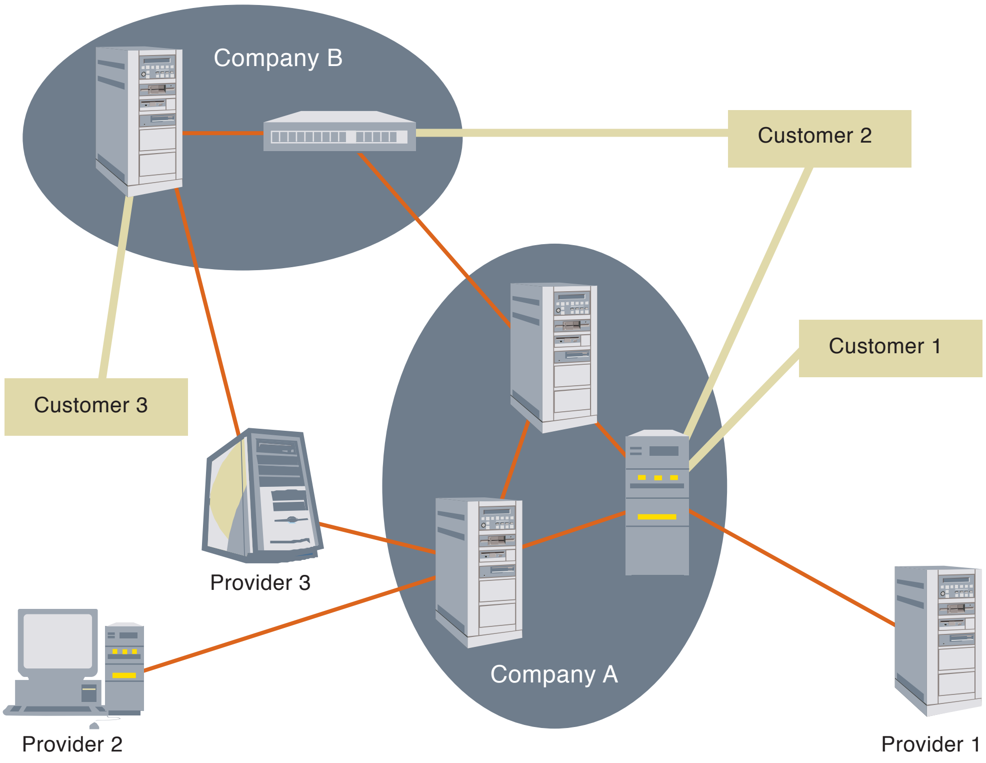


Figure 5 Business ecosystem that benefits from different types of Web Services

vices are used to ensure interoperability between the partners' applications across the public Internet. Due to the lack of widely adopted specifications, however, the companies must agree upon the technologies used to develop these interoperating Web Services. This category of Web Services is ideally suited in a B2B context, i.e. for electronic business administration between a buyer, a seller and possibly a third party.

As the final step, companies might want to extend their services to reach more partners and customers in a commercial Web Service. They could use Web Services to sell various content or business services over the Internet to potential customers anywhere on the web in a pay-per-use manner or through subscriptions. Examples of such services could be location specific weather forecasts, currency exchange, authentication services, etc.

These different categories of Web Services can coexist in a system that connects internal systems together, expose services to a number of external partners and customers as well as utilizing Web Services provided by others in a business ecosystem as illustrated in Figure 5. In this figure, company A and company B both use application integration Web Services to integrate the internal applications within their enterprise. They use business integration Web Services to integrate with each other's applications as well as with other business partners' applications and finally, they offer commercial Web Services to a number of different customers.

3.4 Web Services Requirements

A number of basic requirements must be fulfilled if such interoperable, ubiquitous Web Services as described above are to become a reality. As a minimum, there is a need for widely adopted specifications that are neutral to both platform and programming language for the following:

- Publication of services
- Discovery of services
- Description of services
- Invocation of services

In other words, a commonly agreed mechanism for publishing services so that others can find them is required as well as a well-defined way to search for services providing certain functions. Without these mechanisms, only a very restricted number of clients that are informed directly by the service providers will be able use the services. In addition, the published services must be fully described so that a service requestor can develop a client that knows how to invoke it. Finally, a protocol that allows invocation of the service over the Internet must be specified.

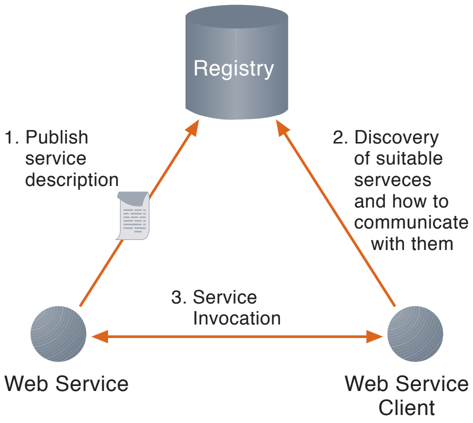

Registry
1. Publish service description
2. Discovery of suitable serveces and how to communicate with them
3. Service Invocation
Web Service
Web Service Client

Figure 6 Basic Web Services architecture

To what extent Web Services will be a success depends on whether these basic requirements, which basically all deal with interoperability between different implementations, are fulfilled or not. Assuming they can be fulfilled, the basic Web Services architecture will look like the illustration in Figure 6. In this figure, it should not matter if the Web Service client and the Web Service are implemented on different platforms, as long as the communication between them follow well specified rules. In the next section, the different standards associated with XML Web Services that address these requirements will be discussed.

On top of the basic interoperability requirements that are needed to make the Web Services architecture work in the first place come other requirements such as stability, reliability and efficiency, security, scalability, availability, manageability, accountability, etc. A thorough discussion of these requirements is considered beyond the scope of this article, but it should be sufficient to mention that the degree of success and the range of use of Web Services will depend on how these are met as well.

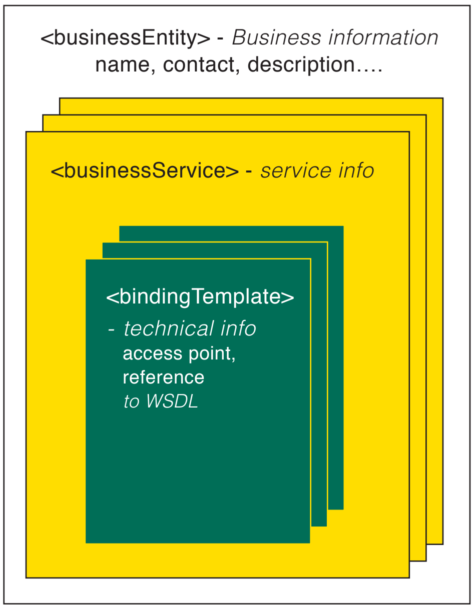
4 Standards Associated with XML Web Services

In order to meet the four basic requirements discussed in the previous section, major industry actors have joined forces and proposed several XML based solutions that are now more or less commonly agreed upon. Three of these initiatives, UDDI, WSDL and SOAP will be discussed in more detail in this section.

4.1 Publishing and Discovering Web Services – UDDI

In order for Web Services to work satisfactorily, there is a need for a standard way to publish and discover Web Services. The Universal Description, Discovery and Integration (UDDI) specifications [12] proposed by UDDI.org define a way for businesses to list themselves and their services on the Internet (or intranet) that is commonly accepted to be the standard XML Web

Figure 7 An UDDI entry containing white pages, yellow pages and green pages


<businessEntity> - *Business information*
name, contact, description….
<businessService> - *service info*
<bindingTemplate>
- *technical info*
access point,
reference
to WSDL

Services way of doing this. All information contained in the UDDI registries is formatted in XML.

The UDDI registries provide two groups of APIs: Publishing APIs that allow creation and deletion of entries in the registry, and inquiry APIs that allow search for entries in the registry by different search criteria. The APIs are invoked by sending appropriate SOAP messages to the registries, so each UDDI registry must have some kind of server process that receives and responds to SOAP messages. In fact, UDDI itself can be thought of as an XML Web Service.

A UDDI registry contains a number of entries with each UDDI entry consisting of three parts:

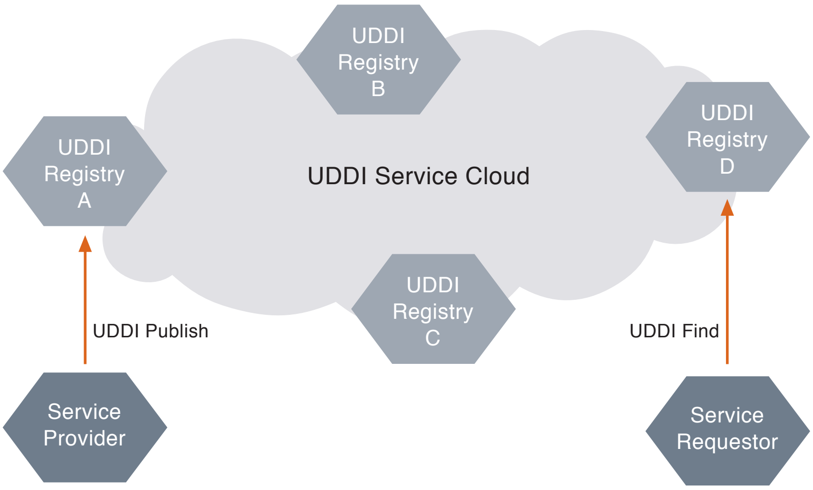
- The white pages
- The yellow pages
- The green pages

In analogy with traditional telephone books, the white pages describe the company offering the services. It contains information such as company name, address, telephone number and other contact information. The yellow pages include information about the type of business and the industry categories that the company belongs to; i.e. it shows the services a certain business provides. The green pages contain the information about the specific services that are offered. Here, the service should be described in enough detail for someone to be able to write an application that uses or consumes the Web Service. Figure 7 shows an example UDDI entry containing white pages, yellow pages and green pages.

A UDDI entry is an XML document with root element <businessEntity>. This corresponds to the white pages where the business information is contained. This root element can contain zero or more <businessService> elements that correspond to the yellow pages and that each represents a family of technical services. The <businessService> thus shows the services a certain business provides. Besides a description of the services, each <businessService> contains a number of <bindingTemplate> elements. These elements provide references to technical information about a service (e.g. a WSDL document), access point for the service, etc.

As already mentioned, two groups of UDDI APIs exist that allow publishing and searching for services. The publishing APIs permit creation and deletion of all kinds of entries. One can publish a whole new business entity or add new service categories or individual services to an existing business. Generally, a userID and password would be required to invoke the publishing APIs. The Inquiry APIs provides ways to find services and to get more details about services. They allow searching for all kinds of entries, in the white, yellow or green pages, based on different search criteria. One can search for businesses based on location or name, businesses and services based on industry categories or types of services, or one can search for specific services.

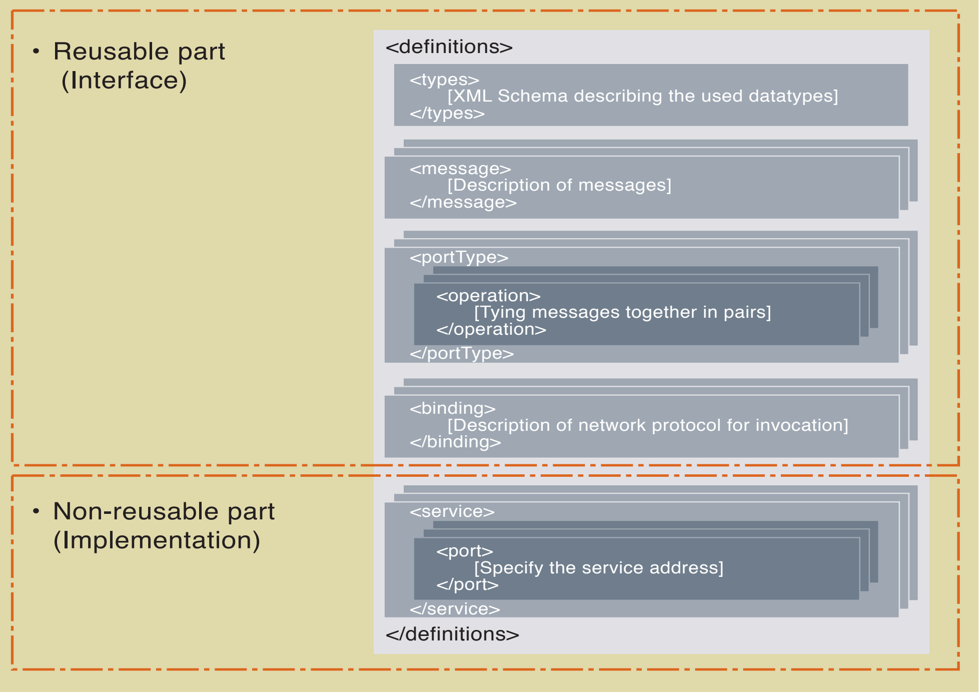
Figure 8 The UDDI service cloud



It is a goal to establish a global network of UDDI registries resembling the Domain Name System (DNS). All these registries should exchange and share their information, so that accessing one registry should provide all information contained in all registries. Already, there is a number of UDDI registries, and the number of registries is expected to grow in the future and form a hierarchical structure. The goal of a network of UDDI registries is illustrated in Figure 8.

4.2 Describing Web Services – WSDL

After a service is found, one needs to know what this particular service can do, where it is located and how to invoke it or format messages to it. From the service provider's point of view, there



• Reusable part (Interface)
• Non-reusable part (Implementation)
<definitions>
<types>
[XML Schema describing the used datatypes]
</types>
<message>
[Description of messages]
</message>
<portType>
<operation>
[Tying messages together in pairs]
</operation>
</portType>
<binding>
[Description of network protocol for invocation]
</binding>
<service>
<port>
[Specify the service address]
</port>
</service>
</definitions>

Figure 9 A WSDL document

is a need to be able to give information to others about what their service can do and how to make a client that can use it. The WSDL specifications [13] specify how this can be done. The specifications were originally developed by IBM and Microsoft and have now been submitted to W3C to become a standard.

A WSDL document is an XML based description of services to make them programmatically accessible to other applications. It must contain enough information for others to be able to write a client program that uses the services it describes. WSDL defines the data types and messaging as well as the specific location of the service. It is in itself independent of underlying protocols, but in the real world it mostly uses XML Schemas to define data types and SOAP (over HTTP) for messaging. Simply put, WSDL specifies the What?, Where? and How? of the services it describes.

A WSDL document contains two parts: A reusable part, sometimes referred to as the interface part, and a non-reusable part, also called the implementation part. The reason for this is the assumption that certain services can be standardised in the future. The interfaces of these services can then be described in a generic way without specifying its exact location. One can then deploy the service at multiple locations without having to duplicate the interface description. An example WSDL document showing all its elements is given in Figure 9.

The root elements of a WSDL document are called definitions, and these again contain a <types> element, one or more <message> elements, <portType> elements, <binding> elements and <service> elements. The non-reusable part of the document is the part containing the <service> elements.

At the highest level, there is a need to define the data types used by the described Web Service and this is done within the <types> element. Usually, the data types are described as an XML Schema. In the <message> element, the different messages that are sent to and from the service are described. Each message can again contain zero or more <part> elements that correspond to the parameters of the message. An operation corresponds to a method call tying the messages (defined in <message>) together in request-response pairs. An <operation> element contains <input> and <output> messages that refer to corresponding messages. One or more operations build a <portType> element, which basically represents a set of functions that the Web Service can process. The following is an example of what this could look like in a part of a WSDL document:

```
<message name=”Request”>
   <part name =”Astring” type=”xsd:string”/>
   <part name=”Bstring” type=”xsd:string”/>
</message>

<message name=”Response”>
   <part name=”Result” type=”xsd:float”/>
</message>

<portType name=”GettingResultService”>
   <operation name=”GetResult”>
      <input message=”Request”/>
      <output message =”Response”/>
   </operation>
</portType>
```

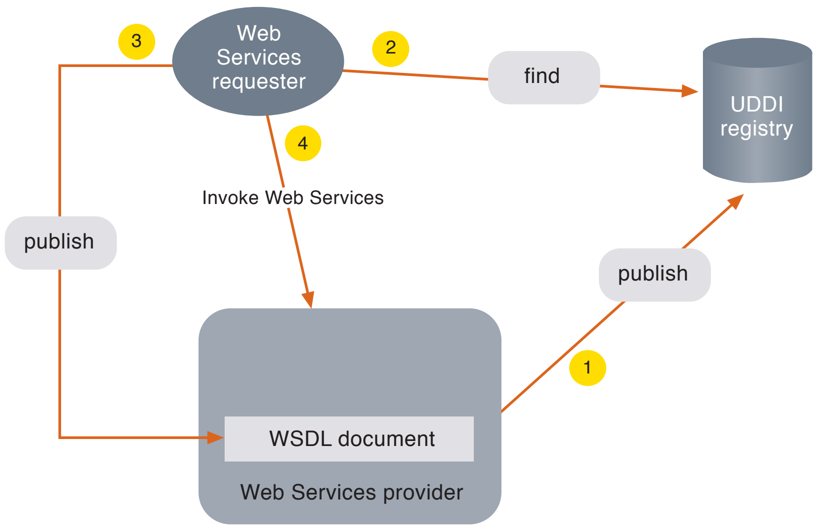


Figure 10 How WSDL works

In this example, the GettingResultService contains an operation that takes two strings, Astring and Bstring, as input and returns a float called Result. In general, it should be noted that each <portType> could contain several operations.

Next, the <binding> element describes the message protocol with which the service can be reached. There can be multiple different bindings for the same <portType> in a WSDL, one of which can be SOAP over HTTP.

In the non-reusable part of the WSDL document, <service> elements are used to describe the services. Inside the <service> elements are <port> elements that specify the actual network endpoints of the service. A service can be made accessible on many ports, for example it can be available via both SOAP and plain HTTP GET. In this case two <port> elements should be defined, each with a different name. One port should specify the SOAP address, i.e. the address of the actual SOAP server handling the request, and the other should specify the HTTP address.

The following gives an overview of how WSDL works: When a service is deployed, its description and a link to it is published in a UDDI registry. When someone finds and wants to use it, they can query the WSDL file to find out the location of the service, how to access it and which function calls to use. The information in the WSDL file is then used to generate a client that can call the Web Service. This is illustrated in Figure 10.

4.3 Invoking Web Services – SOAP

With well-defined ways of publishing Web Services, finding Web Services and describing Web Services, the one basic requirement that is missing is a standard way to invoke Web Services. First, consider the generic Web Services architecture in Figure 11.

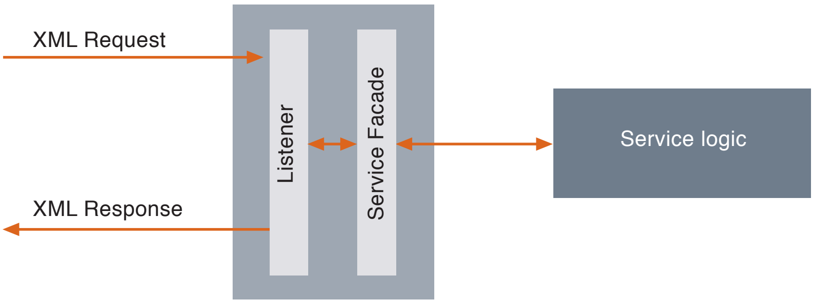


Figure 11 Generic Web Services access

The Web Service is merely a layer that offers programmatic access to a service. The server logic itself can be implemented on traditional platforms using other kinds of middleware and using any kind of programming language. The access to the service consists of a façade that exposes the operations or methods supported by the business logic and a listener that is unaware of the service logic. This listener will typically be a SOAP application as SOAP is emerging as the most widespread way to invoke Web Services over the Internet. One must not necessarily use SOAP for Web Services invocation, but as it is the choice of most vendors, its acceptance as a standard is growing.

The SOAP specifications [14], like the WSDL specifications, were first released as a joint effort by IBM and Microsoft and have also been submitted to the W3C organisation to become a standard. It defines the structure of SOAP messages, a model for exchanging SOAP messages and how SOAP messages over HTTP can be used for Remote Procedure Calls (RPC).

SOAP is a lightweight protocol that defines a uniform way of passing XML encoded data. In doing so it allows code of any kind, on any language and on any platform to cross-communicate, also between parties with no prior knowledge of each other or each other's platforms. It is essentially a one-way messaging protocol, meaning that a sent message does not necessarily lead to a response message (unless something went wrong while sending it), but two SOAP messages can be combined in for example an HTTP request-response pair. Each SOAP message contains one XML document with root element <Envelope> and a <Header> and a <Body> element. Figure 12 shows the structure of a SOAP message.

The <Envelope> element serves as a container for the other elements of the SOAP message, and for this reason a SOAP message is often referred to as a SOAP envelope. The <Header> element is optional and can add different fea-

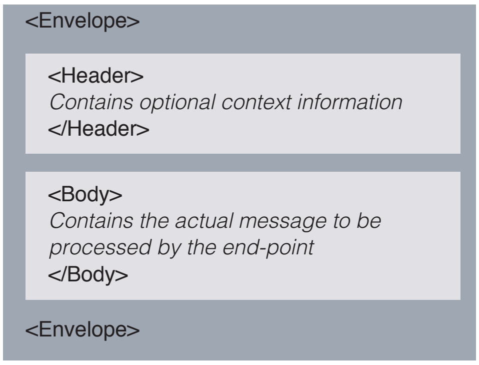

<Envelope>
<Header>
Contains optional context information
</Header>
<Body>
Contains the actual message to be processed by the end-point
</Body>
<Envelope>

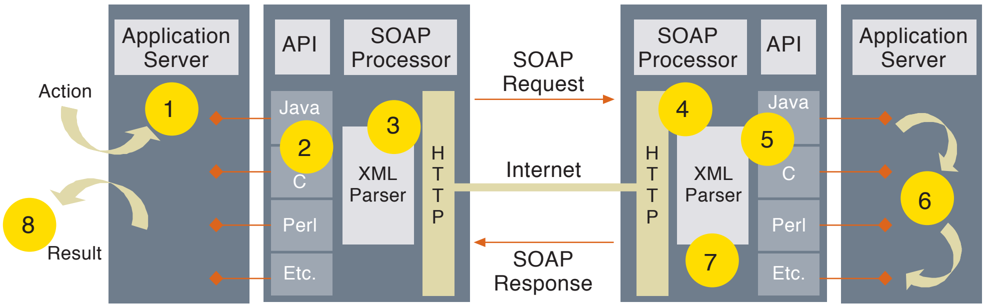
Figure 12 The structure of a SOAP message

tures to the SOAP message without affecting the payload. Authentication is an example of such an extension. An authentication server can process the header entries, independent of the <Body> element, and remove the information it used to validate the signature. The rest of the message will be passed on to the SOAP server that will process the body of the message. The payload of a SOAP message is the <Body> element that contains the actual application specific data that is to be processed by the end-point.

SOAP can be used in combination with a number of other protocols. In the SOAP specifications, however, the only bindings that are defined describe how SOAP can be used in combination with HTTP. It is also described how to use SOAP messages for (RPC).

In a typical SOAP message exchange model, there are three basic components: A SOAP client, a SOAP server and the actual service. The SOAP client creates an XML document, the SOAP message, with the information needed to remotely invoke a method over a network. This SOAP message is then typically sent over HTTP, allowing it to traverse almost any firewall. On the server side, a SOAP server listens for SOAP messages and acts as a distributor and interpreter of these. It then translates the XML structure in the SOAP message into a language that the actual service can understand. It also translates the response from the service into a SOAP response that is sent back to the SOAP client. In Figure 13, a SOAP message exchange example is given.

The exchange of SOAP messages does not have to follow a traditional client-server model, as SOAP supports message chains or intermediates. A logical entity that performs some processing of a SOAP message is referred to as an endpoint. An endpoint can function both as a sender and a receiver and this allows for processing chains to be created. These endpoints, which act as both sender and receiver, are called intermediates and can sit on the path a request takes from a sender to a receiver. The endpoints can use the <Header> elements to determine which parts, if any, of the message is addressed to them. If the endpoint is an intermediate, it can remove the parts of the message addressed to it before sending it through to the next endpoint.



1 A command is executed that generates a process witin the application. The result arrives in the application interface.

2 The message is translated into XML format and sent to the Web server.

3 The XML parser checks the coherence of the XML document and sends the SOAP message via HTTP.

4 The XML parser checks the validity of the message using the HTTP and XML headers and accepts or rejects it.

5 The message is routed to the relevant application server and translated so that it is meaningful to the application.

6 The target application executes the task and a result is produced.

7 The return is done in the same way. Translated to XML and sent by HTTP.

8 The result is returned. The result may be displayed in a browser, access to a database, some actions, and so on.

Figure 13 A SOAP message exchange example

4.4 Web Services Process Flows

Once Web Services are commonplace on the Internet, new services can be composed by aggregating existing services, like illustrated in Figure 3. Usually, the different input and output operations must be performed in an exactly predefined order or sequence, and this operation sequence needs to be defined. To do this in a Web Services environment would require technologies that are able to describe Web Services compositions or workflows within newly created applications and even between companies. Appropriate usage pattern of a collection of Web Services to achieve a particular business goal should be specified, e.g. a description of the business process. The interaction pattern of a collection of Web Services should also be specified resulting in a description of the overall partner interactions. For systems spanning enterprise boundaries and systems made from different technologies, an XML based process flow and pattern description would be useful. XLANG [15] and WSFL [16] are initiatives from Microsoft and IBM respectively, that addresses these issues often referred to as orchestration.

5 Conclusion

This article has introduced and explained the Web Services concept. It has outlined the most important technologies normally used for XML Web Services and basically given a high level understanding and answer to the question: "What is an XML Web Service?".

XML Web Services are a new brand of services that can be expected to have a big impact on the World Wide Web. As put by the XML Protocol Working Group at W3 in their charter [17]:

> *Today, the principal use of the World Wide Web is for interactive access to documents and applications. In almost all cases, such access is by human users, typically working through Web browsers, audio players, or other interactive front-end systems. The Web can grow significantly in power and scope if it is extended to support communication between applications, from one program to another.*

The Web Services model, continuing the history of distributed computing, is a promising way to achieve such a universal application-to-application communication. It can lead to better cross-business integration, improved efficiency, cost savings in application development (due to faster application development) and closer customer and vendor relationships. The three basic building blocks in the Web Service model are invocation, description and publishing/discovering of services. A number of different technologies can represent each of these building blocks, but some specifications emerge as the dominant ones and are therefore more important than the others, simply because they are commonly accepted as standards and agreed upon by major actors. These dominant technologies are SOAP for invocation of Web Services, WSDL for describing Web Services and UDDI for publishing and discovering Web Services.

All these specifications are based on XML making XML one of the cornerstones of XML Web Services. XML is important as it defines a standardised way to format and exchange data, and it thus enables applications created with different technologies, platforms and languages to communicate with each other. Web Services use standard Internet protocols like HTTP and this allows ubiquitous access from across company borders and firewalls. Various types of devices, from personal computers and servers to different kinds of smart devices will be able to collaborate seamlessly.

The utilisation of XML Web Services technology can help bring the Internet to a new level, taking it from the traditional information-based Internet to the emerging service-based Internet. In the service-based Internet, machine-to-machine communication will be commonplace and service-based application development will be prevailing. New opportunities for en even richer and more meaningful collaboration between businesses and people will flourish and those who take advantage of these opportunities can benefit greatly from it.

References

1 *Extensible Markup Language (XML) 1.0 (Second Edition)*. (26 June 2002) [online] – URL: http://www.w3.org/TR/REC-xml

2 *XML Schema Part 0: Primer*. (27 June 2002) [online] – URL: http://www.w3.org/TR/xmlschema-0/

3 *The Extensible Stylesheet Language (XSL)*. (27 June 2002) [online] – URL: http://www.w3.org/Style/XSL/

4 *Namespaces in XML*. (27 June 2002) [online] – URL: http://www.w3.org/TR/REC-xml-names/

5 Harold, E R. *XML Bible*. Foster City, CA, IDG Books Worldwide, 1999. (ISBN 0-7645-3236-7)

6 Goldfarb, C F, Prescod, P. *The XML Handbook*. Upper Saddle River, NJ, Prentice-Hall, 1998. (ISBN 0-13-081152-1)

7 *IBM developer Works: Web architecture: Web Services architecture overview*. (27 June 2002) [online] – URL: http://www-106.ibm.com/developerworks/library/w-ovr/

8 *XML Web Services*. (27 June 2002) [online] – URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/wsspecsover.asp

9 *Web Services Architecture Requirements*. (27 June 2002) [online] – URL: http://www.w3.org/TR/wsa-reqs

10 Cauldwell, P et al. *Professional XML Web Services*. Birmingham (UK), Wrox Press, 2001. (ISBN 1-861005-09-1)

11 *Defining Web Services*. (27 June 2002) [online] – URL: http://www.stencilgroup.com/ideas_scope_200106wsdefined.pdf

12 *UDDI*. (10 July 2002) [online] – URL: http://www.uddi.org/specification.html

13 *Web Service Definition Language (WSDL)*. (10 July 2002) [online] – URL: http://www.w3.org/TR/wsdl

14 *Simple Object Access Protocol (SOAP)*. (10 July 2002) [online] – URL: http://www.w3.org/TR/SOAP/

15 *XLANG*. (11 July 2002) [online] – URL: http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm

16 *Web Services Flow Language (WSFL 1.0)*. (11 July 2002) [Online] – URL: http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

17 *XML Protocol Working Group Charter*. (12 July 2002) [online] – URL: http://www.w3.org/2000/09/XML-Protocol-Charter

Acronyms

API	Application Programming Interface
ASP	Application Service Provider
B2B	Business-To-Business
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CSS	Cascading Style Sheets
DCOM	Distributed COM
DNS	Domain Name Service (Domain Name System)
DTD	Document Type Definition (Document Type Description)
EAI	Enterprise Application Integration
FTP	File Transfer Protocol
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IIOP	Internet Inter-ORB Protocol
LAN	Local Area Network
PCDATA	Parsed Character Data
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SGML	Standard Generalized Markup Language
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Indicator (Universal Resource Identifier)
URL	Uniform Resource Locator (Universal Resource Locator)
WSDL	Web Services Definition Language (Web Services Description Language)
WSFL	Web Services Flow Language
WWW	World Wide Web
W3C	World Wide Web Consortium
XML	eXtensible Markup Language
Xpath	XML Path Language
XSL	eXtensible Stylesheet Language
XSL-FO	XSL Formatting Objects
XSLT	XSL Transformations

Distributed Processing Platforms: Transparencies

JAN A AUDESTAD

Jan A Audestad (60) is Senior Advisor for Telenor Corporate University. He is also Adjunct Professor of telematics at the Norwegian University of Science and Technology (NTNU). He has a Master degree in theoretical physics from NTNU in 1965. He joined Telenor in 1971 after four years in the electronics industry. 1971 – 1995 he did research primarily in satellite systems, mobile systems, intelligent networks and information security. Since 1995 he has worked in the area of business strategy. He has chaired a number of international working groups and research projects standardising and developing maritime satellite systems, GSM and intelligent networks.

jan-arild.audestad@telenor.com

1 Why Distributed Processing?

Distributed processing simply means that computers in different locations cooperate in order to perform common tasks. Therefore, distributed processing is nothing new in telecommunications: in automatic telecommunications systems distributed processing has always been required in order to establish calls in the network. It is not so obvious that operations of telecommunications network require even larger and more complex interconnected computer resources in order to obtain information for charging and billing customers, collect performance statistics required for evolution and maintenance of networks, perform remote management and surveillance of switches, routers, satellite earth stations, SDH multiplexers and other equipment, and execute a number of other complex tasks. The operations system is also one of the most heterogeneous systems that exist being composed of all imaginable types of computing devices having been manufactured during the last twenty years. Not only is the physical topology complex but the software functions these computers execute are written in an admixture of most modelling and programming languages developed during the last thirty years or more.

Installations such as satellite earth stations, telephone exchanges, large databases, and local area data networks consist of clusters of several computers. Power plants, oil refineries, flight ticket booking systems, and signal systems of railways also consist of clusters of cooperating computers. Distributed processing is required in the production of almost all services and goods. The common denominator of all these systems is that they are heterogeneous.

All the above systems belong to a single organisation and are usually manufactured by one manufacturer – or if several manufacturers are designing different parts of the system such as in telecommunications networks, the design is in accordance with precise interface and interoperation specifications developed by the owner of the system. This makes it easier to handle heterogeneity. When such systems are distributed over many administrative domains and employing a multiplicity of technologies, we are again facing immense problems related to heterogeneity. Now we are also facing the additional, and by no means simpler problem, of agreeing to develop a single technological standard.

The reasons to develop standards for distributed processing were several. The owners of large systems want to reduce the market power of the equipment manufacturers by specifying open software interfaces so that different manufacturers can compete to deliver software and hardware components for the same application. This was the driving force behind the Information Networking Architecture (INA) developed by Bellcore between 1985 and 1995. Their intention was followed up by the development of the Telecommunications Information Networking Architecture (TINA) by an international consortium consisting of telecom operators and manufacturers of ICT equipment and systems. This work took place between 1992 and 2000. While INA should primarily offer distributed processing in telecommunications systems only, the intention of TINA was to develop a general platform suitable for both operating the telecom network and processing services implemented in this network.

Some systems have become so large and so dynamic that they need the support of platforms shielding the application from the effects of software and configuration errors. There are complex global systems for passenger transfer in international air transport, for monetary settlements between airlines, and for rebooking of flight tickets. The CORBA (Common Object Request Broker Architecture) platform was developed by the Object Management Group (OMG) in order to offer a possible solution to these challenges. A slightly modified CORBA platform is the baseline architecture of TINA. Though the CORBA platform is commercially available from several manufacturers, it never became the commercial success that OMG expected.

None of the platforms developed so far have become commercial successes. However, the need to develop such platforms has become more urgent. The reason is the large number of cooperative systems now being explored. These include platforms for peer-to-peer communications and anarchistic networks, international systems for road payment, platforms for medical monitoring, swarms of interacting sensors, platforms of wearable computers, and many more sophisticated applications. Most of the new applications also include mobile interfaces with strong data-security protection requirements making it more urgent to standardise distributed processing platforms.

Detailed descriptions of the different platforms are found in the specifications developed by standardisation organisations and industrial groups (ITU, ETSI, ISO, OMG, TINA Consortium, IEEE). The general principles for distributed processing were developed by ISO under the heading Open Distributed Processing (ODP). The ODP standard defines the underlying principles of specifying and designing distributed platforms: it is not concerned with specifying any particular type of platform. Because of its general nature, the ODP standard is still the most important reference to distributed processing.

The description of the transparencies given below builds on the experience with ODP, CORBA and TINA. The reason for choosing these standards as the basis for this article is that they are specifications of large, universal systems for distributed processing. The ideas were developed in large groups with continuous review of the solidity and applicability of the ideas both from a technological and a commercial viewpoint. Future platforms should build on these experiences in order to generate new ideas and to save time by not reinventing the wheel.

2 Baseline Principles of Distributed Processing

One major problem concerning the design of computer platforms for distributed processing is the definition of the functions that the platform should offer. These include runtime management of the platform, storing and retrieving permanent and temporary data, organisation of the software modules, and management of the distributed processing itself.

The most important principle of designing a distributed computer application is that first the actual configuration of the system is ignored, that is, the design is based on the assumption that the computer system consists of one machine and one user. This means that the application is defined in the same way as for mainframe computer-applications in general. After this has been accomplished, the application is distributed on the computers making up the system. The purpose of platforms for distributed processing is then to enable the system itself to ensure interoperability of the software objects making up the application irrespective of how the objects are distributed over the individual computers of the system.

In this way, the design of the application is independent of the actual configuration of the system: how many computers it consists of, which computer does what task, where the computers are located, how the computers communicate, and how the hardware and software configuration changes over time. The advantage of this design method is, of course, that the actual complexity of the system is hidden so that the designer of the application software need not take into account how the final system is configured when writing the software.

The mechanisms for managing interoperability of the software objects are the transparencies described below.

Figure 1 shows what the configuration of the distributed system looks like. It consists of the application software interfacing the platform software in each computer. The distribution platform then interfaces the runtime system of the computer in such a way that it looks as if the platform is common for all the computers of the distributed system.

The distributed processing platform connects together all the computers in the system – or the computing nodes as they are denoted in ODP. The platform software is contained in each computer, and it is written in accordance with the interface towards the operating (or runtime) system of the computer. This interface is machine dependent so that different versions of the platform software must be written for different operating systems: Microsoft windows, Linux, Unix and so on. However, the platform ensures that computers with different operating systems can interoperate in order to execute the distributed application.

The software of the distributed platform is also called *middleware* since it acts as a mediator between the application and the runtime system of the computer.

The application software objects are written in a standard language such as Java or C++ that can be compiled at each computer. The Application Programming Interface (API) ensures that all software objects are written using a common syntax such that objects designed by different

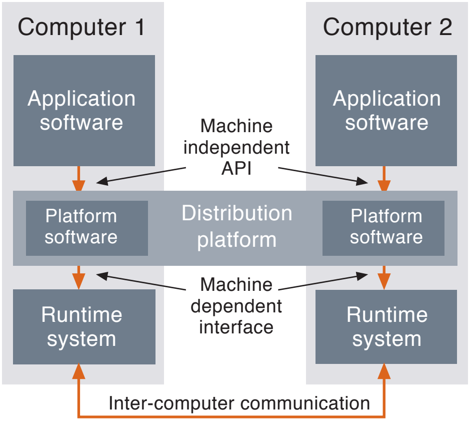


Figure 1 Distributed computation

manufacturers can be purchased and implemented on the same platform.

The basic requirements of platforms offering distributed applications are as follows.

- *Concurrency*. Many applications require processing of concurrent (or simultaneous) processes in a single machine, in a complex of machines at one site, and in machines at remote locations. In this context, concurrent processes are independent processes sharing the resources of the computers in some predefined way. Distributed platforms for telecommunications applications and database applications must offer concurrency. The transaction network for money transfer between banks is a concurrent system.

- *Parallel processing*. Parallel processing comprises two algorithmically different methods: distribution of a single algorithm on several computers where each computer executes parts of the problem (e.g. factoring large numbers, searching over large databases for illegally decoding a cryptogram, or decoding the human genome), and complex programs consisting of many tasks that can be executed at the same time but where execution of the different tasks must be synchronised in order to produce the final result. Most distributed systems are concerned with problems of the latter kind.

- *Timesharing*. Timesharing is quasi-parallel processing within a single processor. This means that each concurrent process is assigned cyclically occurring timeslots where it uses the resources of the computer alone. Between each timeslot, the state of the computation; that is, the set of intermediate results, is stored in memory. In this way, the processing time is evenly distributed between the concurrent applications.

- Timesharing may sometimes be combined with *priority* where the processing timeslots allocated to each concurrent process depend on type of process, its real-time requirement, or other information.

- Many distributed processes require *real-time* processing; that is, the processing has to be completed within strict time limits. If it cannot be completed within this time, the processing has to be terminated in a controlled manner.

- *Weak coupling* means that only a small number of messages are exchanged between any pair of software objects involved in a parallel process. This implies that distributed applications should be designed in such a way that software processes requiring heavy exchange of information should, if possible, be contained within the same object. If a process or machine is overloaded, it is often better that it ignores new requests and leaves it to the requesting process to detect and act upon lack of response. Similarly, if an acknowledgement that a message has been received is not strictly required, it should not be sent in order to reduce processing load and protocol capacity. When specifying and designing systems, much work should be put in weakening synchronisation, removing unnecessary acknowledgements and defining procedures for autonomous fault handling. This is probably the single most important activity when designing a system.

- *Information hiding* is a concept that applies to both software objects and systems. Objects are designed in such a way that only the knowledge (syntax and semantics) of the interface of the object is required in order to access the services offered by the object. This is one of the most important aspects of object-oriented (OO) design. Systems are specified and designed in a similar manner: the only information ensuring cooperation between systems is the syntax and semantics of the protocol interconnecting them. Information hiding is thus the most important mechanism there is in order to handle heterogeneity. This is so obvious that it is often overlooked.

- *Weak synchronisation* means that a system, process or object has no *a priori* knowledge of when it will be invoked by another process, which process is initiating the request, or whether the invocation is remote or local.

- *Heterogeneity* implies that the platform must be such that it can be implemented in different types of equipment ranging from microprocessors of mobile phones to mainframe computers, equipment of different vendors, and equipment spanning a considerable range of age, say ten years or more.

- *Fault management*. Each system, process or object must be able to react to faults on its own, including autonomous processes for fault discovery, fault diagnostics, fault recovery, isolation of faulty software and hardware, redundancy management, and backup memory management. Autonomous fault recovery is complex and expensive but is one of the standard features of telecommunications systems. The reason why this is so important for telecommunications systems is the enormous impact even a short outage may have on society. Failure of several of the new applications mentioned above will have similar severe impact on businesses, public management and society as a whole.

The purpose of the transparency is to implement these and several other objectives. However, before describing the transparencies, let us have a look at what telecommunications networks and applications look like.

3 General Model of Telecommunications Systems

Any type of terminals can be connected to the telecommunications system as shown in Figure 2. The network consists of two parts:

- The network infrastructure offering access connections to the user equipment, and transport and routing of bits (including mobility) between access points.

- The operations system responsible for all management of the telecommunications system, including recording of usage information and statistics, performance monitoring, fault supervision and recovery, reconfiguration management, quality of service and security management, and handling of subscription profiles. The operations system is built on complex, distributed platforms meeting most of the requirements listed in Section 2 above.

Note that in this model the user equipment is not part of the network.

Figure 3 shows the same configuration except that now platforms are included. There is one platform supporting the distributed processing of the network infrastructure and the operations system, and there is another platform implemented on the terminals offering distributed processing to them. The platform supporting the network and the platform existing on the user terminals may be the same type of platform or be different. The functionality of the two platforms is not interconnected but is serving two independent purposes.

Figure 4 shows how the Internet is configured. The reasoning that follows applies to all applications except telephony[1] since all new applications are implemented on the Internet or a network with similar functionality. (We may safely assume that the basic principles of the Internet will be implemented in all telecommunications networks in the foreseeable future.)

The Internet offers IP at the interface to the user terminal. The IP protocol does not offer service capabilities beyond routing, mobility and a few other capabilities; for example, predetermined routing for support of real-time applications such as telephony and video. This is true both for IP version 4 and IP version 6[2]. IP does not offer simple integrity services such as secure delivery

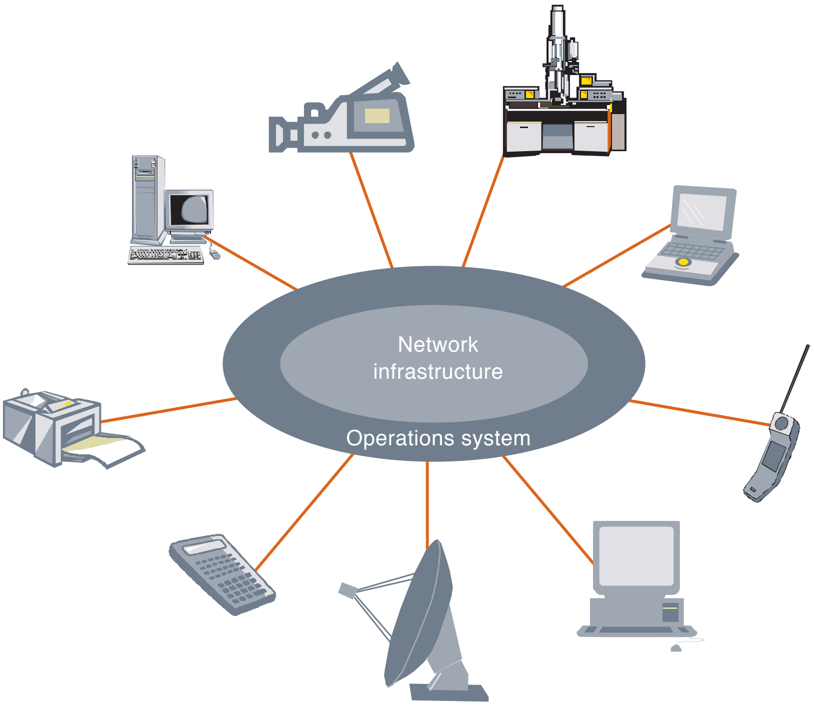
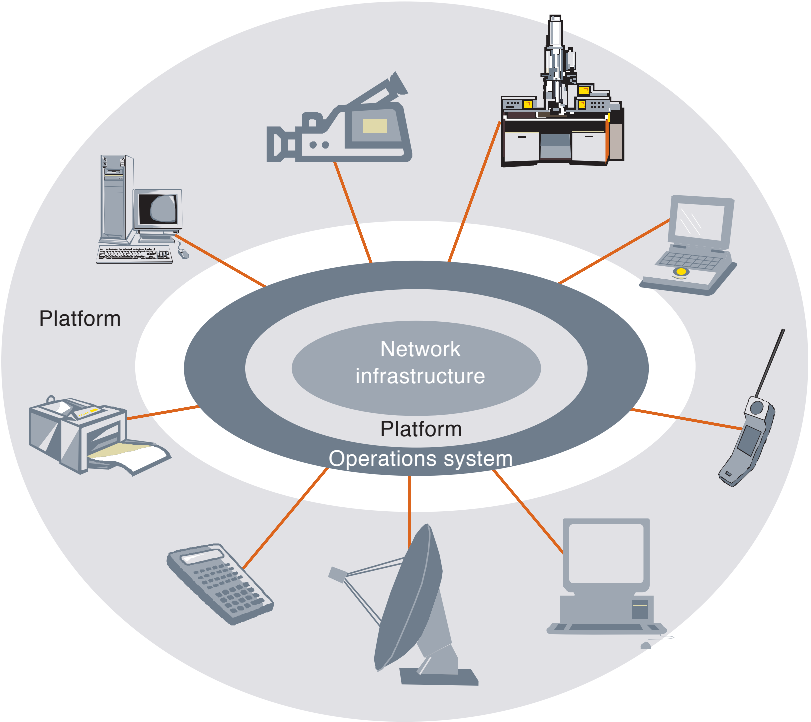


Figure 2 Telecommunications network

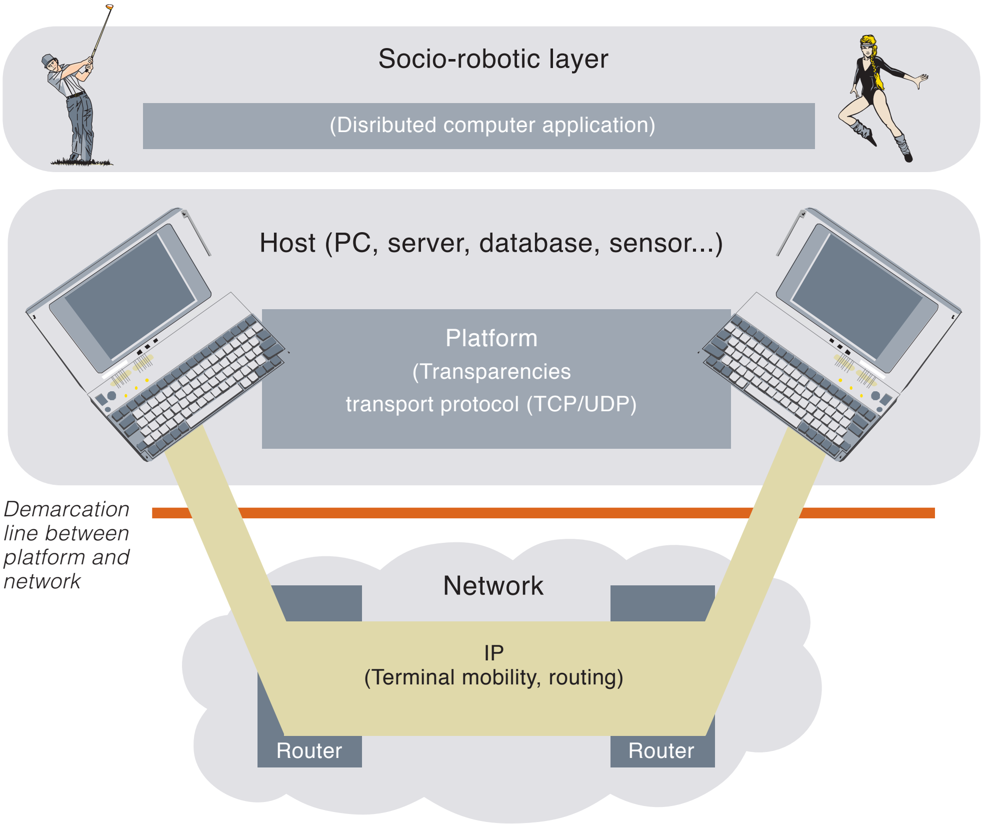
Figure 3 Processing platform



[1] *The telephone network is different because the telephone apparatus does not offer significant functionality beyond that of making telephone calls. The network then contains all functions required for intelligent management of services. In the Internet, this management can be done in the terminals.*

[2] *Capabilities beyond routing and simple mobility support are not really implemented in IP version 4 although several additional capabilities are specified in the standard. This is why IP version 6 was developed.*

Figure 4 Network, platform and applications


Socio-robotic layer
(Disributed computer application)
Host (PC, server, database, sensor...)
Platform
(Transparencies
transport protocol (TCP/UDP)
Demarcation line between platform and network
Network
IP
(Terminal mobility, routing)
Router
Router

of information and recovery of information lost in the network. IP offers a transparent interface to TCP, UDP or any other transport protocol.

TCP and UDP are end-to-end protocols; that is, they offer only interconnection of software processes existing in terminals. TCP and UDP are transparent vehicles for transporting information between these software processes. The information contained in the information field of a TCP or UDP packet can be structured (e.g. a remote procedure call) or unstructured as in the transfer of a video picture or a file. There is no way in which you can tell which type of information is transferred by simply looking at the TCP/UDP header. It is only the sending and receiving terminals that know what type of information is being transferred and how it is to be interpreted. For example, in file management the transfer may alternate between structured information (management of file parameters) and unstructured information (transfer of file content).

The network operator may well offer access to services that require TCP/UDP as the vehicle for transferring information. If so, these services are accommodated in equipment connected to the network in the same way as any other terminal. In other words, the network operator has no advantage of offering services above TCP/UDP compared to a service provider connected to the network. Regulations even prohibit the network operators from offering such access to themselves at a lower price than to other providers offering equivalent services. Depending upon how clever the operator is, the service offer may be better and cheaper than that of the competitors. This is not an advantage gained by owning the network but by being cleverer in marketing and pricing the service.

A platform supporting distributed computing is thus offered on top of TCP/UDP. This platform may offer distribution of processing and storing of information, enhanced mobility such as moving sessions between terminals, transaction services, and transparent data security services such as access control, authentication and confidentiality.

The applications are implemented on top of the platform as explained above. In Figure 4 this is called the socio-robotic layer; socio because it includes communication between people, and robotic because it includes communications between machines. The socio-robotic layer corresponds to the application shown in Figure 1.

4 Transparencies

4.1 Introduction

The remainder of this paper is concerned with the transparencies. Formally the transparencies can be defined as the set of functions that map the design from a mainframe description onto a distributed configuration. The principle is illustrated in Figure 5. The transparencies thus formalise the principle stated at the beginning of Section 2.

4.2 Access

Access transparency hides from the applications message formats and protocol details required for operating in the heterogeneous environment. The application will thus see a standardised interface to its surroundings independent of where (e.g. remote or in the same machine) the cooperating application is located and on which type of computer hardware it is operating. The transparency is the most important element in solving the core issues of heterogeneity: different types of computer hardware and operating systems, different data rates and transfer protocols offered at different network interfaces, and different capabilities of the terminals. Therefore, every distributed system must support this transparency.

In Figure 5, there may be different protocols between the different computers in the distributed environment: one protocol can be TCP over WAP, one UDP over Bluetooth and one TCP over IP.

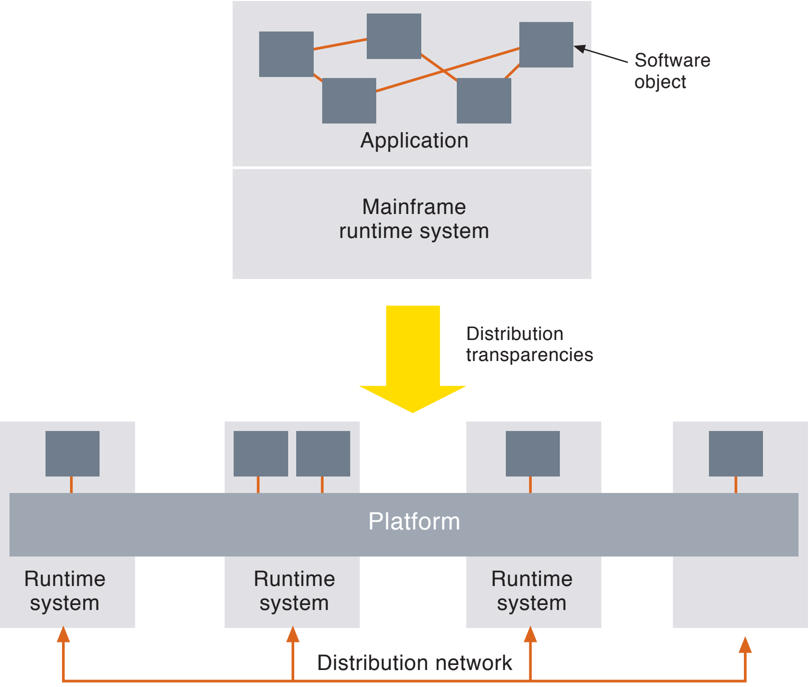


Figure 5 Mapping from mainframe to distributed system

The platform contains various software objects implementing the transparency. One such set of objects adapts the general protocol offered at the API to the system-dependent protocol implemented on the connection. The transparency also ensures binding; that is, establishing the connection to the right peer object, retaining this relationship as long as required so that messages are not misrouted, and removing the connection when it is no longer needed. In some cases, it may even be necessary to install protocol software or set protocol parameters at the initiation of the connection involving negotiations with the binder object of the peer system in order to agree on a common protocol. The application is ignorant of these activities taking place.

The access transparency makes it possible to install the same software at fixed locations operating at 100 Mbps and mobile phones offering 9600 bps. The protocols at these locations will have different characteristics but the platform will ensure that the software object installed at the different locations will interface the protocol in the appropriate way.

4.3 Location

Location transparency hides the location of an application from other applications. The transparency ensures that an object can be inserted in an arbitrary computer and still be found by any other object wanting to communicate with it. The location address of cooperating objects need then not be included in the program code of the objects: it is enough to indicate in the program code which other type of object is needed in order to complete the task. The transparency is realised by two simple functions: when a new object is registered in the system, the identity and the location of the object (that is, in which computer it is installed) together with other relevant information such as the services it offers are announced to one or more address directories. When another object wants to use the services offered by the object, it passes its request to the binder of the access interface. The binder then searches for the address of a suitable object offering the service in the address directories.

The implementation of location transparency seems simple, but it is not. It is a simple and unproblematic task to store the address of the object in the computer where it is located and in computers placing remote calls to this object regularly. It is more difficult to agree on a worldwide structure consisting of address directories accessible to anyone. Some questions are: who should own these databases; should it cost anything to access the directory and to register information in it; and who is responsible for the correctness of the information contained in the directory? These aspects are not technical but legal and commercial, and are therefore much harder to solve.

4.4 Concurrency

Concurrency transparency or *transaction transparency* coordinates the interactions arising when several applications interact with one another at the same time. The transparency preserves the consistency of an object being used concurrently by several other objects. The concurrency mechanism is built into the platform and not included in the individual objects making the objects simpler and more universal. The mechanism in the platform serialises and completes individual transactions on an object in accordance with the ACID rules. ACID stands for the following:

- *Atomicity*. A transaction is completed either fully or not at all (partial transaction results do not exist).

- *Consistency*. The action of the transaction always leads to a result that is in agreement with the specifications of the transaction service.

- *Isolation*. A transaction in a concurrent system is performed as if it was alone (serialisation).

- *Durability*. The data can only be changed by transactions: only a new transaction can alter the result of the previous transaction.

In addition to the ACID requirements, the platform must also protect against deadlock; that is, the platform must always be able to terminate a transaction (for example by cancelling it) also when the transaction fails to terminate on its own. This task is particularly challenging if the operation consists of a complex set of linked and nested transactions between several objects.

The concurrency transparency also hides to an object that several objects are concurrently using the other object.

4.5 Migration and Relocation

Migration transparency is an extension to location transparency making it possible for the system to change the location of an application without the application knowing that it is moved. Such relocation may take place even while other applications are interacting with the process being relocated without disrupting ongoing interactions. Migration may be required in order to reconfigure the processing load of computers, move the computation session from one user terminal to another, or do system maintenance.

When an object is migrated, the following activities may take place:

- The processing of the object is stopped, and the processing state of the object and addressing information required for reconnecting the objects to its peers are stored in a backup memory. The processing state consists of a pointer indicating exactly which program statement is the next to be executed and the temporary results of the computation stored at that instant. This is also called checkpointing and is used for other processing purposes also (maintenance and fault recovery). Connections to other objects are suspended until the object has been relocated.

- Then the data of the processing state is transferred to the other computer so that the object can be reinitiated at the correct state in the new computer. Either the complete program listing of the object (compiled or uncompiled) or a reference to the template in which the program listing can be found is also sent together with the data.

- The new computer installs the object and initiates it with the processing-state data.

- Finally, the connections with other objects are re-established. This is in itself a complex process involving the binders of the access system.

Migration of objects may also make use of synchronised replicated objects (see below) where a replica of the object to be moved is first initiated in the new location. Then the replicated object is synchronised to the original object, and when this has been completed, the original object is terminated. Synchronisation makes use of checkpointing and transfer of the checkpoint data to the replicated object.

The location of the new object may then be stored temporarily in databases so that the object can be found if needed by other applications.

This capability of hiding that relocation takes place is called *relocation transparency*. Migration transparency and relocation transparency are therefore two aspects of the same mechanism: migration transparency allows applications to be moved while the relocation transparency hides such events from other applications bound to the application being moved.

4.6 Replication

Replication transparency hides the replication of one object from the objects using it. Replication means that two or more objects are doing the same task and that each object is a replica of the same object type. The replicated objects may run in different computers. The platform takes care of synchronisation and other processing dependencies of the replicated objects.

The replicated objects may be fully synchronised, that is, they do exactly the same processing at the same time. The platform synchronises the objects and ensures the integrity of the data they contain. This type of replication is used in order to create security groups where the objects act as hot standbys for one another: if one object fails, the other objects can still execute the task. Synchronous replication is also used when the same information is required at different places at the same time and where it is more practical that two objects are used instead of one, e.g. to reduce processing load and processing delay.

One object may interface one, several or all of the replicated objects of a set. The platform takes then care of how the communication is arranged and managed.

Mobile agents may be asynchronous or synchronous replicated objects sent to different computers where they are executed. The process sending out the mobile agents is not aware of how many replicas exist and where they are located. Processes in the platform manage the multiple binding to all the agents and organise the information exchange between home base and agent.

The replication and migration transparencies are important tools for managing processing load.

4.7 Persistence

Persistence transparency implies that activation or deactivation of objects is not explicitly visible to other objects. Hence, the transparency supports evolution of the system in the sense that new software may be added or old software removed without having impact on other parts of the system.

The transparency also hides from other objects that the software of the cooperating object is compiled and instantiated when an operation towards it is invoked; i.e. the object is not persistent but created whenever someone uses it.

The program code of the objects can be loaded down from specification repositories possibly subject to commercial rules. The platform must then contain the software (the factory software) capable of loading down the template and the compiler that compiles the program code of the object. The actual object compiler may then be associated with the template and not be stored in the computer where the object is initiated. This allows more flexible computer configuration since the computers need not be prepared *a priori* for all the compilers and languages in which the objects are written.

4.8 Failure

Failure transparency hides failure of one object from other objects and prevents other processes from failing as a secondary effect. The transparency also hides encapsulation of failed objects, recovery actions taken to restore normal operation, and any subsequent rearrangement of the system.

The automatic restart systems of telephone exchanges and satellite earth stations are examples of implementations of failure handling of the type to be implemented in distributed processing platforms.

4.9 Security

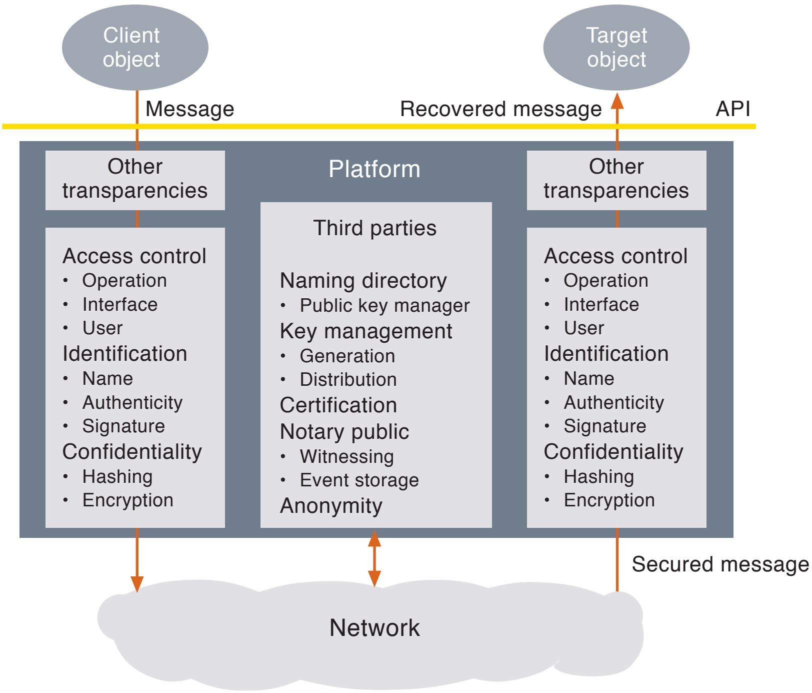
Security transparency comprises several complex functions such as access control, authentication, certification, confidentiality and data integrity, non-repudiation, notary public services, anonymity management, and non-traceability. Each of these functions can in fact be viewed as a separate transparency because the implementation of each of them requires its own network functionality. The transparencies are also difficult to implement because they require access to specialised equipment including public key and naming directories, key generation and distribution systems, databases containing access control information and access decision algorithms, certification management systems, and systems for producing and storing non-repudiation information.

Examples of platforms offering automatic authentication and encryption without involving the call handling functions are the GSM platform and the UMTS platform. These transparencies are simple. However, transparent access control and notary public services are very complex and difficult to implement. Figure 6 shows how security features can be built into the platform.

The example shows two communicating objects: the client object initiating the interaction and the target object. The security system consists of software in the end systems, that is, the sending and the receiving computers, and in a number of third parties offering support to the end systems.

The end systems may contain the following functions:

Figure 6 Security transparency



- Access control provided for each operation, access control associated with each object interface, and access control associated with each user. The sending system will check whether communications with the target object is allowed, whether the parameters of the operation is within allowed limits, and insert access information that the receiving system can use in its analysis. The receiving system will analyse whether the user is allowed to access the system (only my wife and I have read access to our joint bank account but we do not have write access), whether the interface is allowed to service the access request (the read interface cannot be used to change the amount of money on the account), and whether the invoked operation is allowed (my wife and I can invoke read operations and payment service operations). Access control may in addition depend on other parameters or events such as type of authentication, certificates provided by third parties, message confidentiality and integrity, and the time when the operation was received.

- Identification including secure naming services, retrieval of public keys from third parties, authentication of client and target objects, and provision of electronic signatures to be attached to messages. The keys used for authentication may also be secret keys generated by a third party. Retrieval of public keys and generation of secret keys may require certification by a third party.

- Confidentiality services include encryption and integrity protection, for example, by hashes, timestamps or nonces. Third parties may generate and distribute encryption keys and nonces.

Third parties may support capabilities beyond those just described. This may include anonymity where aliases replace plaintext addresses over unsecured connections (GSM offers this feature over the radio connection), and notary public services offering non-repudiation. The latter includes witnessing that events take place and storing data associated with the event. The data can be retrieved later in order to settle disputes concerning whether or not an event took place at a certain time.

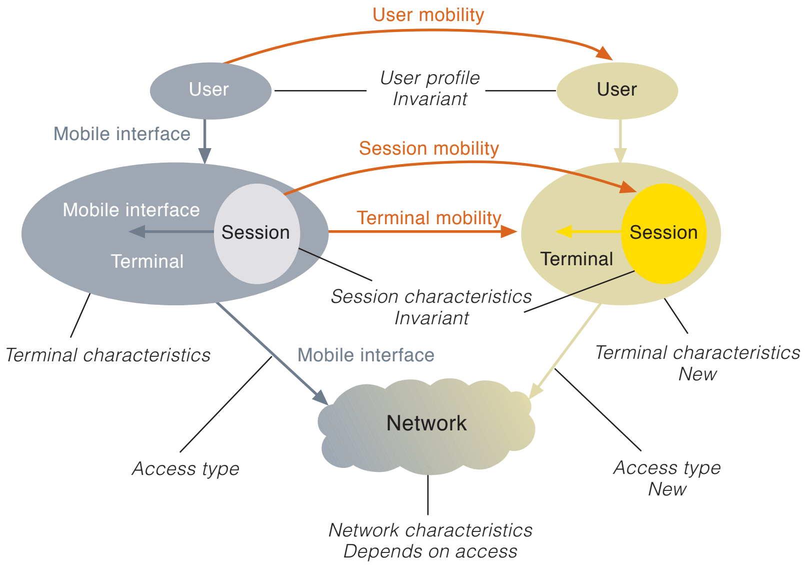
4.10 Mobility

Mobility transparency enables design of software processes independent of whether they are run on mobile or fixed systems. The problem associated with mobility and transparency is illustrated in Figure 7. The figure shows a *user* running a *session* at a *terminal* connected via an *access* to a *network*, that is, involving five elements each having its own characteristics: user profile, session characteristics, terminal characteristics, access type, and network characteristics. There are three mobile interfaces: between the user and the terminal, between the session and the terminal, and between the terminal and the network, the latter being managed by the network. The platform implemented above TCP must support user mobility and session mobility. This implies also that the relationships between user and user profile and between session and session characteristics are retained during mobility. However, how the user profile and the session characteristics are to be interpreted, depends on the terminal characteristics, the access type and the network characteristics.

User mobility implies that the user moves from one terminal to another. The major concern here is to retain the user profile and adjust it in accordance with the new terminal characteristics, the new access type, and the new network characteristics associated with the new access. The user profile is concerned with which applications the user is allowed to initiate (files, programs, operating systems, e-mail, Internet), which access rights are assigned to him as a user, and which security features are required for different types of access. This means that the capabilities available to the user may be different if the access is inside a firewall or outside the firewall. The mobility transparency shall take care of these functions.

Session mobility makes use of the migration and relocation transparency in order to move parts of the session. In addition, session mobility requires management of the session characteristics and adjusts it in accordance with other system characteristics similar to user mobility. When a session is moved, for example, from one side of a firewall to the other, this is likely to require reestablishment of the security features.

Figure 7 Mobility



4.11 Processing Dependency and Grouping

Processing dependency and grouping transparency hides how timesharing and other dependencies between applications are realised. It also hides all types of grouping of objects into processing entities and the reason for forming such groups (commercial packaging, load optimisation, or timing constraints). Note that the same application may be organised differently in different machines and by different organisations but that the modelling of the application should be independent of this.

In CORBA the processing dependency transparency supporting timesharing is implemented as threads in the computing node software. Other dependencies such as handling of internal procedure calls versus remote procedure calls and shared memory are also implemented in organising objects (groups and clusters).

4.12 Federation

Federation transparency implies that the distributed system may be modelled and designed independently of business relationships and ownership of applications over systems spanning several owners. This is particularly important when designing telecommunications systems.

A distributed system may be divided into domains representing ownership, technology and other aspects varying over the system. Such subdivisions are then not visible to the users of the platform.

4.13 Scalability

Scalability transparency means that the system may grow in size and incorporate new functionality without redesigning the system architecture and software objects already installed. The location transparency takes care of the flexible addressing and identification required for smooth scalability. The access transparency partly solves the heterogeneity issues. The federation transparency takes care of heterogeneity associated with ownership and administrative responsibility so that the platform can span several organisations and businesses. However, this is not enough to ensure that new computers containing the distribution platform can be installed in the system without extensive management activities involving already installed computers and systems such as reconfiguration of the architecture and re-initiation of the platform.

The scalability transparency must support smooth introduction of new platform capabilities, facilitate easy installation of new hardware (user terminals, databases, servers, mainframe computers and so on), and rearrangement and evolution of the platform topology.

The Internet is the best example of systems with extreme agility with regard to scalability. The most important feature there is that routers are charged with the autonomous task of figuring out what their environment looks like. This includes also topological information such as which other routers are in the environment. The traffic handling capacity of the resulting network is not system optimal since each router makes its own independent decision. The advantage is, of course, that the network does not need management resources in order to update topological data: the identity of the nearest neighbours, which routers are operational and which are faulty and taken out of service, instantaneous maps of the traffic distribution, and tables indicating how calls shall be forwarded at each router.

However, the type of platforms considered in this paper are platforms interfacing TCP, UDP or another end-to-end transport protocol. Therefore the scalability of the Internet will have no impact on the scalability of these platforms. Scalability must be implemented once more in a different part of the system.

5 Conclusion

We have described twelve transparencies that should be included in platforms for distributed processing. The most complex platforms, including CORBA and TINA, only offer access transparency and location transparencies as mandatory platform elements. However, even these transparencies do not offer full support of all the characteristics required for flexible service implementation.

Some of the transparencies described above are included not as transparencies in CORBA and TINA but as other service features of the platform (persistence, processing dependencies and grouping, and simple security services).

The reason why even these advanced platforms do not offer more is that it is difficult and expensive to implement the transparencies.

Platforms for distributed processing have existed as commercial products for not more than six or seven years. The specifications of some of the platforms are still not complete and will never be finished. It will probably take at least ten more years before platforms with the desired characteristics are available for general commercial use. The platform software is complex and huge. Therefore, the platform can only be accommodated in future generations of microprocessors. However, the technology must be developed now in order to be available within the next decade or so.

Several of the transparencies are much more complex and commercially important than the other transparencies. These are particularly concurrency, security, mobility and scalability. These are also the transparencies that are the most urgent to implement. The reasons are:

- Concurrency or transaction handling is required in large applications such as banking, finance and e-commerce. Banking includes diverse applications such as transfer of money between banks, remote management of users' accounts, electronic payment of services and goods, and automatic payment of road toll, parking charge and energy usage charge. Such transactions will soon be required by computers everywhere: the PC at home, cash registers in shops, e-money databases, the electricity meter, the road payment chip, the parking meter, and so on. The transparency will therefore be used in a number of diverse applications. To be more general: whenever information is stored in or retrieved from any device even remotely resembling a database, transaction services are involved.

- Security is hardly being implemented in any system yet. The transaction services described above rely heavily on security: confidentiality, e.g. in order not to disclose secret keys exchanged between payer and merchant, data-integrity hashing and timestamping to protect against manipulation of content, non-repudiation to prove that a transaction has taken place, authentication to establish secure identity, and access control to protect against fraudulent manipulation of any type of information, including bank accounts. Security will also, for the similar reasons as for transactions, become a key feature in remote access to business systems, in cooperative work, and in distributed management systems of government and businesses. The urgency of implementing security on a grand scale is also evident from the fact that the number of security attacks on computer systems at least doubles every year. This is the highest form of e-persecution mania!

- Mobile access both on radio and fixed lines is ubiquitous. Presently only terminal mobility is generally supported (radio systems and mobile IP). This type of mobility is well handled by the network. Personal mobility allows people to access the same processing session at different terminals with different capabilities and characteristics. Session mobility moves processing sessions between different computers and between different users. Personal mobility and session mobility are still not solved in a satisfactory way. These advanced forms of mobility also require strict security.

- Scalability must be built into the platform from the beginning. Platforms are too often designed for a single application, employing a single technology, fitted to the needs of a single user, or installed in a single system. Road payment platforms are designed for the single purpose of collecting road toll, are employing passive radio beacon technology, are specified and used by a single road authority, and are implemented as a system consisting of a small number of toll stations, a financial institution handling the administration of collected money, and the management system of the road authority. Such systems are certainly not scalable in any interpretation of that word.

There are several reasons why standardised platforms are not becoming the expected commercial success. The platforms are complex and expensive to build so that most of the sales price depends on the number sold. Therefore, the building and marketing of new platforms is a risky business. The platform may not offer what the users want, the reason being that the specification is done in large groups run by organisations having diverse motives for implementing platforms. Such platforms consist often of sets of compromises satisfying as many compromises as possible. The result is that the platform is not attractive to anyone; it contains too much of what is not needed and too little of what is needed. It takes much time (ten years is rather the rule than the exception) to finish the specifications. The platform design is then facing a different reality when it is finished than when it was conceived.

These problems do not imply that joint efforts to specify platforms is a waste of time and should be abandoned. Platforms are more needed than earlier so that the efforts should go on; sooner or later the process converges and platforms that are commercially viable are a reality.

However, platforms developed by a single organisation may not reach the market either. The OLE platform of Microsoft is an example of this.

If the reader is involved in designing or purchasing distributed platforms, he or she is referred to the vast amount of available literature on platforms. Detailed data concerning ODP, CORBA and TINA can be found on these Internet addresses:

www.iso.org and search for Open Distributed Processing
www.omg.
www.tinac.com

Web Services – An Evolution of the Distributed Computing Model

TRON SYVERSEN

Tron Syversen (30) works as sales executive and specialist on mobile solutions for iAnywhere Solutions, a subsidiary of Sybase Inc. His job consists of sales, promotion and marketing of mobile solutions. Through iAnywhere Solutions, Sybase offers customers and partners a complete product portfolio of mobile software. Tron likes to work with tailored solutions based on mobile databases and synchronization technology. He has a technical background and has been working with introducing Sybase mobile technology on the Nordic market for the last three years.

Tron.Syversen@sybase.com

Today, the Internet's two most popular services are e-mail and the World Wide Web. However, as the Internet evolves over the next few years, we are likely to see Web services become equally popular since they, ultimately, will enable companies to conduct e-Business more effectively and at less cost. Web services are the next-generation component model for distributed computing and a natural extension of integration technology. The promise of Web services is the ability to enable services to freely communicate over the public network, as well as create a service-oriented approach to computing.

Web services are used for deploying, providing, and orchestrating access to business functions over the Web. Web services are built on existing technologies, such as XML, so they can be implemented easily and incrementally. Yet the efficiencies they are capable of bringing about could dramatically change the way e-Business is conducted.

In contemplating the adoption of Web services, it is important to remember that whenever a company seeks to provide services and goods using Internet technologies, it can find itself involved in large integration projects that lead to business, technical, and political challenges. Not surprisingly, when such projects fail, the consequences are disastrous. Therefore, Web services and successful integration go hand in hand.

Origins in Component Architecture

The historical trend that has led us towards Web services has its roots in component architecture of the 1980s. Components were originally developed in the context of graphical user interfaces (GUIs) and allow the re-use of existing application code. Within this architecture, code is compiled into several independent units instead of into a single entity. These units or components communicate via an infrastructure provided by the operating system.

But component architecture allowed developers to do even more than simply build GUIs. Soon its use grew from managing communication between components on the same computer to communicating with component infrastructures on other computers across a network. Distributed component architectures enabled the rapid development of complex, distributed applications. This led to the specification of CORBA (the Common Object Request Broker Architecture) in 1991 and Microsoft DCOM component architecture, as well as the development of the Sun J2EE distributed architecture.

Distributed component architectures have one serious limitation: generally they can only be used across a tightly managed network, such as a corporate intranet. They do not play well in an open, fragile environment, such as the public Internet. Hence the need for Web services in an Internet world. In the same way that component middleware allows one piece of software to make use of the functionality contained in another piece of software on another computer, Web services use the Internet's protocols to provide a component infrastructure for the development of distributed components that function on the public net. Web services are modular applications that can be described, published, located, and invoked over a network through standardized XML messaging. They are defined by new standards – like Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Discovery Description and Integration (UDDI). Web services offer a new model for creating e-Business applications from reusable software modules that are accessed on the Web.

Benefits of Web Services

From a technical standpoint, Web services offer an easier way to develop applications that need to be accessed over the Web. It is important to note that Web services do not solve all integration requirements, only the ability to communicate with other software modules over the public net. Additional integration technologies are still needed to handle integration between data, applications, and business processes. Also needed are the enterprise-class capabilities to surround the Web services, enabling them to be secure and scalable.

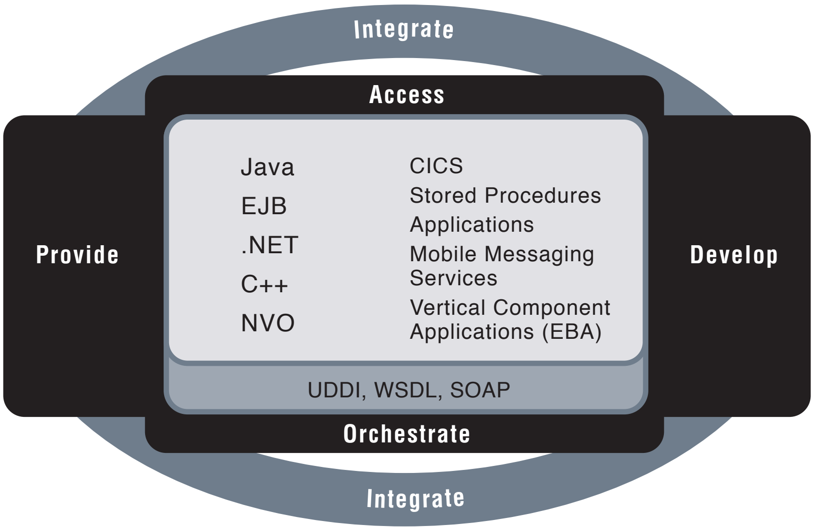
From a business standpoint, Web services allow a company to concentrate development efforts on computing assets that drive revenue. Business models and relationships are evolved as necessary, the costs of integration reduced, interactions with marketplaces established more efficiently, and business functions delivered to a broader set of customers and partners. Web services technology enables the outsourcing of services that provide no business benefit. Because Web services decouple applications and infrastructure, a company can quickly compose and deploy solutions based on reusable components from the lowest-cost provider, whether internal or external. These solutions can change the target, and even the nature of interactions, in response to changing business conditions. Organizations can leverage a flexible and dynamic business model, which maximizes their reach to customers, partners, suppliers, and marketplaces while minimizing their costs and time to market.

Sybase's Solution for Web Services

Over the last 18 years, Sybase has been delivering enterprise-class infrastructure software aimed at helping customers with their distributed computing needs. This software, which couples integration with distributed computing, is Sybase's heritage. We were the first vendor to introduce a database for distributed computing, the first to release replication and distributed data access in a heterogeneous environment, the first to handle both Java and COM components as well as to adopt J2EE technology, and the first to bridge mobile users with the enterprise.

Most recently, we extended the integration capabilities of most of our software infrastructure to include the developing, accessing, providing, and orchestrating of Web services. From within this infrastructure, customers can expose existing software components from multiple vendors' technologies and have the unique advantage of being able to quickly adapt to Web services. This is accomplished in two ways. First, Sybase Web services can expose a wide variety of existing software components, including Java, EJBs, database stored procedures, CICS components, and mobile wireless services.

Second, we help customers adapt to Web services by providing a solution that lets them adapt Web services incrementally; that is, when appropriate and at different times. In the evolution of distributed computing, it will be many years before Web services are mainstream, so their full adoption will occur in a number of steps. For example, a company may itself adapt Web services but its partners and suppliers – as well as different divisions within that company – will proceed on a different schedule. In such cases, Sybase Web services can bridge heterogeneous technologies at a logical level and act as a central control above the other services. We can also act as a hosted service, allowing customers to connect quickly to their partners using the technology of choice and eliminating the need to install and maintain multiple gateways and connectivity software.



Sybase is providing a comprehensive portfolio of open-integration for Web services and delivering them in four ways:

- Development of Web services – software that supports the development and automation of Web services

- Providing Web services – software that hosts the Web services

- Accessing Web services – software that connects to a service and delivers the value of that service to its intended user(s)

- Orchestrating Web services – a set of software services that are designed to help coordinate the activities of services while they are being used.

An example of a Sybase enabled Web services product is our Enterprise Application Server (EAServer) version 4.1. EAServer supports the open standards and technologies necessary to develop, publish, and provide Web services-based applications, including UDDI, SOAP, J2EE, WSDL, and the ability to interface with a public UDDI registry.

Mobile Web Services

As a part of Sybase's comprehensive Web services strategy iAnywhere Solutions, a subsidiary of Sybase INC, will enable delivery of Web services to mobile and wireless users. This includes an integrated software platform for extending the reach of e-Business applications, messages, enterprise data, and Internet content to mobile and wireless devices, including smartphones, PDAs and laptops. By exposing this functionality as Web services, other applications and servers can easily tap into this platform to leverage mobile-specific features. For example, an application can access a Web service running on a middleware platform to send an SMS message to a user's phone.

Conclusion

Sybase believes that the next generation of e-Business will be “services-oriented” because of the increasing need to decouple applications and infrastructure for the rapid construction and deployment of flexible solutions. Standards-based Web services – teamed with integration technology – open the door to more effective and efficient e-Business. These services support massive automation and integration of applications and business processes, allow rapid acceleration of the e-Business innovation-integration cycle, and reduce infrastructure costs and complexity. The underlying technology is still evolving, but through the efforts of Sybase and others, companies will be able to adapt to Web services quickly, achieving business value today and being well positioned for tomorrow. To encourage standards that will make Web services a more valuable part of the Internet, Sybase is now working through two major organizations: the Web services Interoperability (WSI) Organization and OASIS (Organization for the Advancement of Structured Information Standards).

INTEGRATE
Develop
Provide
Access
Orchestrate
Web Services

SYBASE WEB SERVICES ENABLED

Company profile

Sybase is a leading provider of enterprise software products and services, including enterprise portals, vertical solutions and mobile technologies. Such products and services enable companies to build robust e-business infrastructures for integrating, managing, and delivering information anywhere it is needed. With nearly 5,350 employees and 2000 revenues of $960.5 million, Sybase is one of the largest independent software companies in the world. Today Sybase’s end-to-end solutions are defining the requirements for enterprise solutions, Web computing and Mobile computing. These solutions currently focus on four principal vertical markets: financial services, telecommunications/media, healthcare, and the government/public sector.

Sybase’s headquarters are located in Dublin, California, and operates in over 60 countries. The company’s stock is traded on the NYSE under the symbol SY.

iAnywhere Solutions, a subsidiary of Sybase, Inc., is the market-leading provider of mobile and wireless solutions that enable anywhere, anytime access to enterprise information. More than 6 million users at 10,000 customers sites worldwide rely on iAnywhere Solutions to power solutions that help to improve productivity, streamline operations and create new revenue streams. With more than a decade of experience, iAnywhere Solutions provides a one-stop source for organizations’ m-business needs across key markets including financial services, healthcare, government, utilities, transportation and retail.

Web Service Development Explained

ANNE MARIE HARTVIGSEN, LUIS ARTURO FLORES AND DO VAN THANH

This paper explains in generic terms how Web Services and Web Service clients are developed.

Anne Marie Hartvigsen (25) finalised her Master in Information Systems at Agder University College in June 2002, with the thesis "Studying Emerging Technologies in Telenor – Using Web Services to Provide Universally Accessible User Profiles". She also holds a Cand.Mag. in social sciences from the Norwegian University of Technology and Science (NTNU). Formerly a visiting researcher at Telenor R&D, working in the PANDA (Personal Area Network and Data Applications) research group, she is now employed as a systems developer at the Telenor spin-off Xymphonic Systems.

anne.hartvigsen@xymphonic.com

Luis Arturo Flores (24) obtained the Computer Systems Engineering degree with Honors distinction from the Monterrey Tech (ITESM) in Mexico, complementing his studies with university programs in the University of Melbourne, Australia; and the Washington College at Maryland, USA, where he is a member of the Honor Society. In 2002–2001 he participated in Information Systems projects in Mexico and the United States before joining Telenor R&D in June, 2001. At Telenor he performed research focused on the Session Initiation Protocol and XML Web Services, together with Dr. Prof. Do Van Thanh.

al766023@mail.mty.itesm.mx

1 Introduction

This paper explains Web service development in generic terms – without discussing implementation details on different platforms. The first part focuses on the process of developing a Web service, while the second part concerns client development. Finally we list useful sources of information in regard to Web service development.

2 How A Web Service Is Made

This chapter tries to explain the generic process of creating a Web service, regardless of platforms and other implementation details. It assumes the following definition of a Web service:

> *A Web service is application logic that is made available on the Internet using the following standards: XML for data description, SOAP for message wrapping, WSDL for service description and UDDI for service discovery.*

We describe two scenarios for developing Web services: Starting from scratch or starting with an existing service.

2.1 Developing a Web Service from Scratch

Starting from scratch means creating all the functionality in a service from the bottom up, before exposing the desired functionality to Web service clients. The process is as follows:

1 Create service functionality
2 Generate Web service interface
3 Deploy service
4 Publish service

2.1.1 Create Service Functionality

The service logic must be developed according to the goals of providing the service. The platform and internal workings of the service are irrelevant in a Web service perspective, since Web services depend on the model shown in Figure 1.

In this way Web services allow for cross-platform interoperability in a way that makes the platform irrelevant. Internally the service may connect to a database, an Enterprise Java Bean, a COM object, a legacy application, etc., but towards the Web service client all communication will be done using higher level, universal protocols. This is the same principle that allows a Web browser and a Web server to be implemented with completely different technologies but still communicate without problems using the common, universal standards HTTP and HTML.

Once the functionality is in place, the next step is to expose it as one or more Web services.

2.1.2 Generate Web Service Interface

The functionality created in the first step needs to be given a Web service interface in order to be a Web service. The tool the programmer is using should implement the Web service layer automatically if the developer specifies what functionality to expose as a Web service. But what does it mean that the tool generates a Web services interface? There are two steps involved in exposing functionality as Web services:

a) Create SOAP interface
b) Create WSDL service description

2.1.2.1 Create SOAP Interface: the Web Service Proxy

For a Web service to be called by remote clients, a service listener and a service proxy need to be in place, as shown in Figure 2. This will all be taken care of by the tools/platform you choose to build your Web services on, but it is still important to have a general idea of what is going on.

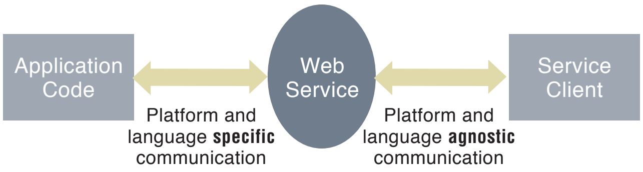

Application Code
Web Service
Service Client
Platform and language **specific** communication
Platform and language **agnostic** communication

Figure 1 Web Services abstract implementation details

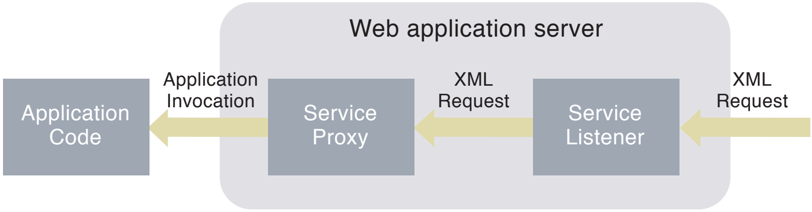


Figure 2 Generic Web Services architecture

Do Van Thanh (44) obtained his MSc in Electronic and Computer Sciences from the Norwegian Univ. of Science and Technology (NTNU) in 1984 and his PhD in Informatics from the University of Oslo in 1997. In 1991 he joined Ericsson R&D Department in Oslo after 7 years of R&D at Norsk Data, a minicomputer manufacturer in Oslo. In 2000 he joined Telenor R&D and is now in charge of PANDA (Personal Area Network & Data Applications) research activities with a focus on SIP, XML and next generation mobile applications. He also holds a professor position at the Department of Telematics at NTNU in Trondheim. He is author of numerous publications and inventer of a dozen patents.

thanh-van.do@telenor.com

The service listener receives incoming SOAP requests, which the service proxy parses and interprets, and decodes into calls to invoke the application code created in the first step. This proxy must understand how to handle all things that can occur in a SOAP request. When it receives a request from the listener it performs three steps:

1 Deserialise the message from XML into the native format that the application code requires;

2 Invoke the code;

3 Serialise the response to the message back into XML and hand it back to the listener for delivery back to the requester.

It is the proxy and the listener that are often referred to as *SOAP wrapping*, meaning that they provide an abstract layer between the service implementation and the calling client. Because this is merely a standardised way of wrapping existing functionality, it is quite straightforward to add, and Web services tools will do it automatically. There already exist SOAP toolkits for all the popular programming languages and environments (see http://www.soaplite.com or http://www.soapware.org for detailed product listings). The proxy component needs to know exactly which code to invoke when a particular message is handed to it, and different Web service tools have different mechanisms for doing this.

SOAP is an infrastructure technology, and once this interface is generated, SOAP works "behind the scenes" to make sure the service and a client may interoperate. Ideally the SOAP components are completely transparent to the application code, so that the code does not even know it is being invoked through a Web service.

2.1.2.2 Create WSDL Service Description: the WSDL File

Once the SOAP wrapping is in place, it is possible for clients to issue SOAP requests to the service and receive SOAP responses back. But the client is not able to send requests or handle the responses if it does not know what kind of requests the service accepts, or what kind of responses it can expect to get back. This information is provided in the service's WSDL file.

The WSDL file defines the Web service in terms of its logical and physical interface.

- A service consists of one or more ports.
- Each port is on the one hand bound to a specific URI, and on the other hand to what is called a binding.
- A binding is a combination between an operation (method signature) that can be invoked and a specific protocol that can be used to invoke it (e.g. SOAP).
- Each operation is composed of messages, which again consist of parts (data values) that have their own names (e.g. strUserID) and part types (e.g. string). The part types (data types) are defined using XML Schema.
- Operations are grouped together in logical port types.

The separation between logical elements – port types, operations, messages – and physical elements – data types, protocols, URIs – makes it easy to for example offer the same service on several different protocols (e.g. SOAP, HTTP GET, HTTP POST) and/or from several different locations (URIs).

As with the SOAP proxy, the WSDL file is created automatically, since it merely provides a formal description of an already existing service. This description is expressed in an XML document that adheres to the WSDL protocol for defining services. This makes it possible for other applications to read the WSDL file programmatically, and to generate client proxies for the Web service automatically.

2.1.3 Deploy the Web Service

The Web service must run in an environment that allows clients to access it. Usually this means deploying the service to a Web application server.

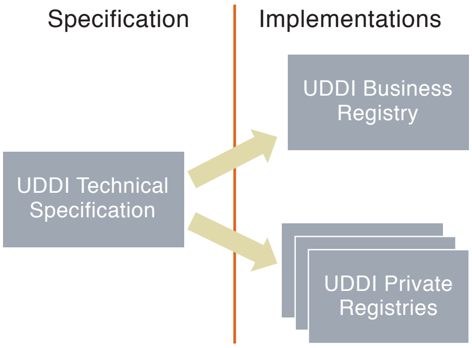


Figure 3 UDDI specification and implementations

Once the Web service layer is implemented and deployed, all that remains to be done is publishing the service so that clients may find it.

2.1.4 Publish the Web Service

The last step in creating a Web service is publishing it to the UDDI business registry. It is of course possible to have Web services that are not registered in UDDI, but that assume that suitable clients get to know about the service in other ways. The purpose of registering the service in the UDDI registry is to allow potential users of the service to find it based on its functionality.

Example: A developer is creating a Web site that requires users to be authenticated. Instead of programming this functionality herself, the developer can search the UDDI registry for Web services providing this functionality. There the developer may find several authentication services, e.g. Microsoft's Passport service. Other information also resides there, such as information about the company offering the service, and the terms for using the service (the service provider can for example require that the client pays per use, or an annual fee, or nothing at all; there may be restrictions of how many times per day the service can be invoked, etc.). By studying the information, an appropriate service can be chosen, and integrated in the Web site solution with the help of the service's WSDL file. If after some time the developer does not want to use the selected service anymore, it should be easy to find an equivalent service (presuming one exists) in the UDDI registry and switch to that instead.

UDDI is a technical specification for building a distributed directory of businesses and Web services. The data model is an XML Schema for describing businesses and Web services. UDDI also includes SOAP API details for searching existing data and publishing new data.

The UDDI Business Registry (cloud services) is an implementation of the specification, accessible through the different UDDI Business Registry Nodes (UBR Nodes). Businesses can register themselves and their services at these nodes. They each offer two interfaces. One is a Web page, which can be accessed with a regular Web browser. The other is UDDI's publishing API where different save and delete functions can be used by SOAP messages. This can be useful if you want to use software that automatically registers new Web services or if, e.g., you need an automatic method for updating binding access URLs.

As illustrated in Figure 3, there are other UDDI implementations in addition to the UBR. These implementations allow you to publish to your own private registry, which can be shared within a company or among partners.

If you are developing Web services for internal use they should of course not be published in the UBR. You should however register your services somehow, in order to make efficient use of them. One possible way is to publish them in your own, private UDDI registry.

2.2 Developing a Web Service from an Existing Service

Building a Web service from an existing service means that the functionality already exists, but there is no Web service interface to it. Except for the first step the process is the same as for building a Web service from scratch:

1 Build interface to the service
2 Generate Web service interface
3 Deploy service
4 Publish service

2.2.1 Build Interface to the Service

Web services are most useful when they are used to expose existing functionality. Whether Web services are used to tie internal systems together or to expose services to external actors, Web services should provide a fast and easy way to accomplish this. Using Web services also means that once the interface is in place it can be accessed from applications on any platform, meaning that only one interface is needed. The bottom line is that existing functionality relatively easy can get added value, leveraging prior investments.

If the functionality already exists on a platform from which one wants to offer Web services, then all that is needed is to generate the Web service interface in the same way as for building a service from scratch. If it is not possible or desirable to generate the Web service interface from this platform, the developer must first interface the functionality with the desired Web service platform, and then generate the Web service interface on that platform. The resulting architecture is shown in Figure 4.

There are generally two reasons why one would want to generate the Web service interface on another platform

- It is not possible to generate a Web service interface directly from the existing solution/ platform;
- The current platform lacks desired functionality.

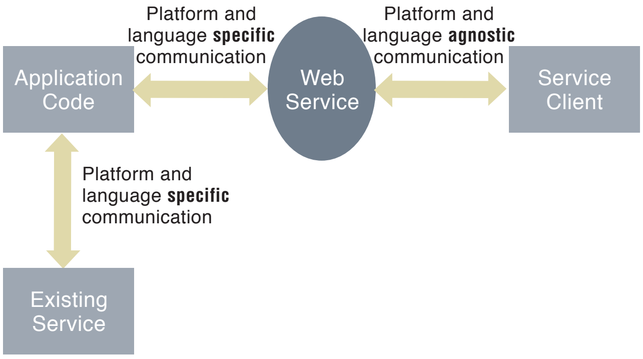
One example of the first case may be providing Short Message Service (SMS) as a Web service. This task first requires that a gateway be built between the SMS centre and the desired Web service platform (e.g. .NET or a J2EE platform). Next the Web service interface can be generated. Laird (2001) describes a project that developed an SMS Web service and also explains why this kind of functionality is particularly suitable to offer as a Web service.

An example of the second case is when one has an application running in Perl. By using SOAP::Lite for Perl one can easily offer the desired functionality as Web services, since SOAP::Lite supports SOAP, WSDL and UDDI. However, SOAP::Lite is a very simple tool – aimed at solving the problem from a programmer's view point. It might be that more features are needed than what is offered by SOAP::Lite, for example if one wants to tie together multiple Web services – then a more complex implementation, like Apache Axis, might provide a better platform. Or let us say one wants to expose the service with the aim of selling it to potential users. Then one needs to do things like keep track of who is using one's service so one can charge them, and there might be other business aspects to the Web service initiative. Maybe there is already an e-business server running that takes care of some business aspects and integrates Web services with that – then one would probably want to offer one's service from that.

When building the bridge between one's service and the chosen Web service platform, there is no formula for how this should be done. It depends on the systems at hand, available tools and skills, and the trade-off between performance versus flexibility and development costs. Depending on all these factors one could either build some proprietary interface oneself or use technologies like COM, CORBA, Java RMI, etc. One could even end up using Web services to tie the two platforms together!

2.2.2 Generate Web Service Interface, Deploy and Publish the Service

These steps are the same whether starting with an existing service or building the service from scratch. See the chapter on building Web services from scratch.



Platform and language **specific** communication
Platform and language **agnostic** communication
Application Code
Web Service
Service Client
Platform and language **specific** communication
Existing Service

Figure 4 Web Service built from existing service on another platform

3 How A Web Service Client is Made: Web Service Consumption

This chapter aims to explain the generic process of consuming an existing Web service, regardless of platforms and other implementation details. The process involves four steps:

1 Find service
2 Generate client proxy
3 Invoke service

3.1.1 Discover the Web Service

There are many ways to obtain information about a Web service that one wants to use. Typically if using Web services to tie together internal systems or to do e-business with trusted partners one would be given descriptions and specifications for the specific services to use. However, if looking for some functionality and wanting to see if there is a Web service for this, the appropriate place to search for it, according to Web service standards, is a UDDI registry. Some companies may use UDDI implementations to keep private service registries. The UDDI Business Registry (cloud services) is another implementation of the specification, accessible through the different UDDI Business Registry Nodes (UBR Nodes). This is the place to search for public services, enabling anyone to search existing UDDI data. These data are divided into three categories:

- White Pages – general information about companies;

- Yellow Pages – general classification data for companies and services;

- Green Pages – technical information about Web services (using the term in its broadest sense, including everything from Web pages and e-mail addresses up to SOAP, CORBA, Java RMI services, etc.).

Although some technical information about a Web service can be found in the registry, it will mostly also contain a link to a WSDL file and

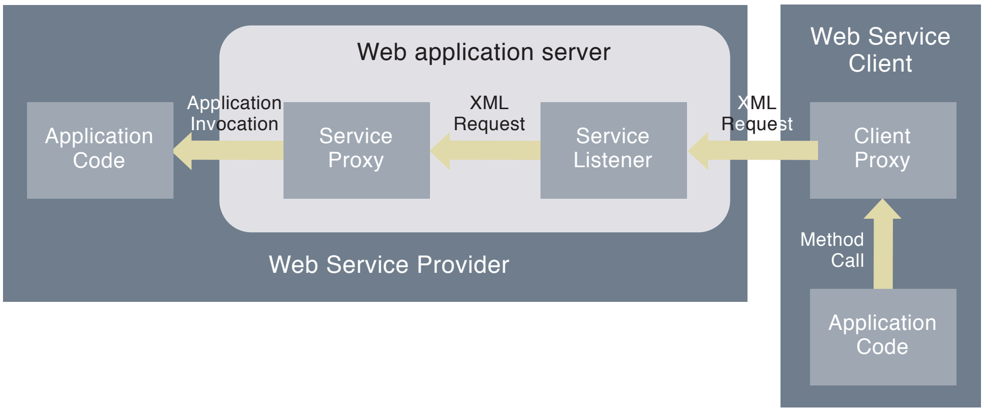


Figure 5 Web Service architecture – server and client

possibly other service descriptions located at the service provider.

Searching of the UDDI registry can be done by accessing a UBR Node. They each offer two interfaces. One is a Web page, which can be accessed with a regular Web browser. The other is UDDI's inquiry API where different find and get functions can be requested using SOAP messages. Such requests could either be sent directly or a UDDI tool could be used.

3.1.2 Generate a Client Proxy

Once the desired service and the WSDL file that belongs to it have been located, your development tool should be able to generate a client proxy based on the WSDL file. When calling Web service methods in one's application, it is actually the methods in this proxy that are being called. The proxy then generates suitable SOAP requests that it uses to invoke the service. Results from the requests are received by the proxy, which deserialises the XML and returns the answer to the relevant application on an appropriate format. Conceptually this is the same as for the server proxy, and this is shown in Figure 5.

3.1.3 Invoke Service: Sending and Receiving SOAP Messages

To invoke the Web service all that has to be done is make calls to the proxy class, and this will forward the call as a SOAP request to the Web service. The answer returned will be handled by the client proxy, and handed over to the application code in native format.

4 Summary

Recalling all that has been mentioned in this chapter, we provided one approach that we consider simple, yet effective for developing a Web service from scratch. To do so, we suggested going by the following process:

1 *Create service functionality*. This includes writing the code of the components that will constitute the application, including the functions that will be available through the service.

2 *Generate Web service interface*. This step is generally performed by the developing tools, so one should not worry about carrying out this step, but just to test whether the tools are actually doing it correctly one can e.g. check that a WSDL file has been created for the service and that the details in the WSDL description seem correct.

3 *Deploy service*. To deploy the service means to make it available on one's Web server, where service clients can access it. Depending on the development tools and the platform used, the deployment will be performed according to these. But basically, a proxy class that speaks SOAP and a WSDL file describing the interface of this class must be made available. This allows clients to reference the service and test if the actual service itself is reachable by them.

4 *Publish service*. Finally we talked about publishing the service at the UDDI business registry. This is the registry for businesses worldwide to list themselves and their services on the Internet. Of course this is only necessary if one wants the service to be available worldwide. If the purpose of the project is to create services for the internal use of the company, it may not be necessary to register the service here. But somehow the network address of the WSDL file has to be provided to the clients, or the interested developers of your company. One can even have one's own internal "myUDDI" registry, for benefit of the company's developers.

We also presented a brief description of what needs to be done to create a Web service from an existing service. This process is very similar to the previous one, except that on the first step, instead of building the service itself, only the interface has to be created. That is, generate the necessary functions or processes that can be called and that link the entering calls with the implemented functionality of the service. So, in this case, the functions themselves will work as an intermediary between the Web interface, and the existing functionality. Steps two, three, and four are just the same as when building a service from scratch.

Finally, we explained how to consume an existing Web service. The following steps constituted the approach we introduced:

1 *Find service*. This means finding the desired service to complement our application. We can search the UDDI registry, or by other means find the description of the service functionality and the network address of the WSDL file.

2 *Generate client proxy*. By means of the development environment, the service found in step one needs to be referenced. The development tool must be able to understand the WSDL file in order to be able to generate the necessary invoking method to the developer. This is called a proxy class because it is a class that acts as an intermediary between the client application and the Web service.

3 *Invoke service*. Finally, it is just a matter of calling the service. To do so, call the proxy class, and it will forward the call as a SOAP request to the Web service, and obtain the answer returned from the service.

5 Resources

In the following we list some resources that may be useful when developing Web services and clients.

5.1 SOAP

SOAP 1.1: *http://www.w3.org/TR/SOAP/*
SOAP 1.2: *http://www.w3.org/TR/soap12-part1/*

Implementations
Overview: *http://www.soapware.org/directory/4/implementations*

Apache Axis: *http://xml.apache.org/axis/*

Microsoft SOAP ToolKit 3.0:
http://msdn.microsoft.com/soap

SOAP::Lite for Perl: *http://soaplite.com*

Glue from the Mind Electric:
http://www.themindelectric.com/glue/

5.2 WSDL

WSDL 1.1: *http://www.w3.org/TR/wsdl*

WSDL 1.2: *http://www.w3.org/TR/2002/WD-wsdl12-20020709/*

WSDL validator: *http://pocketsoap.com/wsdl/*

Invocation Tools
Glue from the Mind Electric:
http://www.themindelectric.com/glue/

SOAP::Lite for Perl: *http://soaplite.com*

IBM Web Services Invocation Framework /WSIF): *http://www.alphaworks.ibm.com/tech/wsif*

5.3 UDDI

UDDI Business Registry Nodes (Operators)
- Microsoft: *http://uddi.microsoft.com/*
- IBM: *http://uddi.ibm.com/*
- SAP: *http://uddi.sap.com/*

Inquiry and Publish Interfaces
- Microsoft
  - *http://uddi.microsoft.com/inquire*
  - *https://uddi.microsoft.com/publish*

- IBM
  - *http://uddi.ibm.com/ubr/inquiryapi*
  - *https://uddi.ibm.com/ubr/publishapi*

- SAP
  - *http://uddi.sap.com/uddi/api/inquiry*
  - *https://uddi.sap.com/uddi/api/publish*

Tools
- UDDI4J (UDDI for Java) – a tool for using the UDDI APIs: *http://oss.software.ibm.com/developerworks/projects/uddi4j/*

- jUDDI: *http://www.juddi.org/*

- SOAP::Lite: *http://soaplite.com/*

- Microsfot UDDI Software Development Kit: *http://uddi.microsoft.com/developer/*

Information
- The official UDDI site: *http://www.uddi.org*

- UDDI technical newsgroup: *http://groups.yahoo.com/group/uddi-technical/*

- UDDI information and news: *http://uddicentral.com/*

5.4 Other Information and Resources

W3C Web Services Activity:
http://www.w3.org/2002/ws/

The IBM Web Services Browser:
http://demo.alphaworks.ibm.com/browser/

6 Sources

The information in this paper was extracted from the following sources:

Cauldwell, P et al. 2001. *Professional XML Web Services*. Birmingham, UK, Wrox Press Ltd.

Cerami, E. 2002. *Web Services Essentials*. CA, USA, O'Reilly.

Graham, S et al. 2002. *Building Web Services with Java : Making Sense of XML, SOAP, WSDL, and UDDI*. Indiana, USA, Sams Publishing.

Laird, C. 2001. *SMS : Case study of a Web services deployment. "Instant gratification" programming results*. July 25, 2002 [online] – URL: http://www-106.ibm.com/developerworks/webservices/library/ws-sms.html

Snell, J, Tidwell, D, Kulchenko, P. 2002. *Building Web Services with SOAP*. CA, USA O'Reilly.

Vasudevan, V. 2001. *A Web Services Primer*. O'Reilly XML.com. July 22, 2002 [online] – URL: http://www.xml.com/pub/a/2001/04/04/webservices/index.html

Major Web Service Products and Relations to Mobile Telco Services

ERIK LILLEVOLD AND DO VAN THANH

Erik Lillevold (60) obtained his MSc in Physics from the Norwegian University of Science and Technology (NTNU) in 1970 and joined the Norwegian Defense Research Institute (NDRE) in 1971. He worked there as a research scientist until 1986 when he joined Telenor R&D. He has most of his time worked in the field of messaging and application services, e.g. X.400, Internet Mail, WWW and WAP. In the last few years he has devoted his work to Open Service Platforms (Parlay, OSA, Web Services, etc.).

erik.lillevold@telenor.com

Do Van Thanh (44) obtained his MSc in Electronic and Computer Sciences from the Norwegian Univ. of Science and Technology (NTNU) in 1984 and his PhD in Informatics from the University of Oslo in 1997. In 1991 he joined Ericsson R&D Department in Oslo after 7 years of R&D at Norsk Data, a minicomputer manufacturer in Oslo. In 2000 he joined Telenor R&D and is now in charge of PANDA (Personal Area Network & Data Applications) research activities with a focus on SIP, XML and next generation mobile applications. He also holds a professor position at the Department of Telematics at NTNU in Trondheim. He is author of numerous publications and inventer of a dozen patents.
thanh-van.do@telenor.com

1 Introduction

This article presents an investigation of the major XML Web Services products that are available on the market and an overview of the Web Service arena with the dominant players. The article will also consider briefly the Telco services relevant to be exposed as XML Web Services.

The term Web Services describes a standardized way of integrating Web-based applications using the XML, SOAP, WSDL and UDDI open standards over an Internet protocol backbone. XML is used to tag the data, SOAP is used to transfer the data, WSDL is used for describing the services available and UDDI is used for listing what services are available. Used primarily as a means for businesses to communicate with each other and with clients, Web services allow organizations to communicate data without intimate knowledge of each other's IT systems behind the firewall.

When it comes to Web Service products we see that most of the vendors either base their products on Java platforms or Microsoft platforms. Due to the Web Service standardization different products will ideally be able to interoperate when needed.

2 Platforms

To develop and run a commercial Web Service it is convenient to differentiate between four types of platforms needed:

- Provider platform – the environment that does the hosting of a Web Service, for example an application server.
- Consumer platform – the environment where the consumer or client connects to a Web Service, for example a user interface that automatically accesses the remote Web Service.
- Production platform – the environment where automation of developing Web Services is provided. Support for the Web Services standards must be provided.
- Management platform – the environment that coordinates the usage of Web Services, for example providing QoS, delivery guarantees, security management, trading relationships management, etc.

A Web Service product that covers all these platforms may be regarded as complete.

The vendors have so far put a lot of emphasis into the first three platforms, but they have not got that far in the management platform area.

2.1 Provider/Consumer Platform

A Web Service will always consist of a provider and a consumer platform representing correspondingly the server application side and the client application side of the service. In most cases those two platforms are based on existing application servers of which there are mainly two different types, those based on Java 2 Enterprise Edition (J2EE) or those based on Microsoft .NET.

2.1.1 J2EE

The J2EE platform uses a multi-tiered distributed application model. Application logic is divided into components according to functions. The various application components that make up a J2EE application are installed on different machines depending on the tier in the multi-tiered J2EE environment to which the application component belongs. Figure 1 shows two multi-tiered J2EE applications divided into the tiers described in the following list:

- Client-tier components run on the client machine;
- Web-tier components run on the J2EE server;
- Business-tier components run on the J2EE server;
- Enterprise information system (EIS) tier software run on the EIS server.

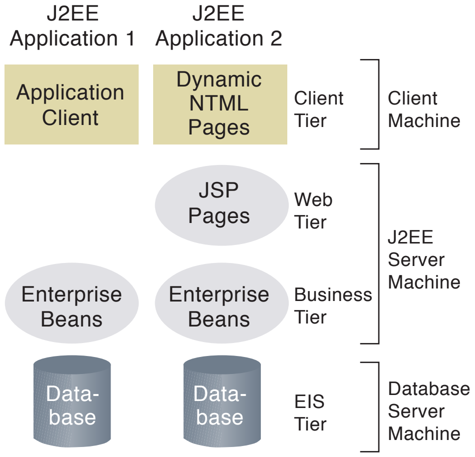


Figure 1 The J2EE platform

J2EE components are written in the Java programming language and are compiled in the same way as any program in the language. The difference between J2EE components and "standard" Java classes is that J2EE components are assembled into a J2EE application, verified to be well formed and in compliance with the J2EE specification, and deployed to production, where they are run and managed by the J2EE server. The J2EE specification defines the following J2EE components:

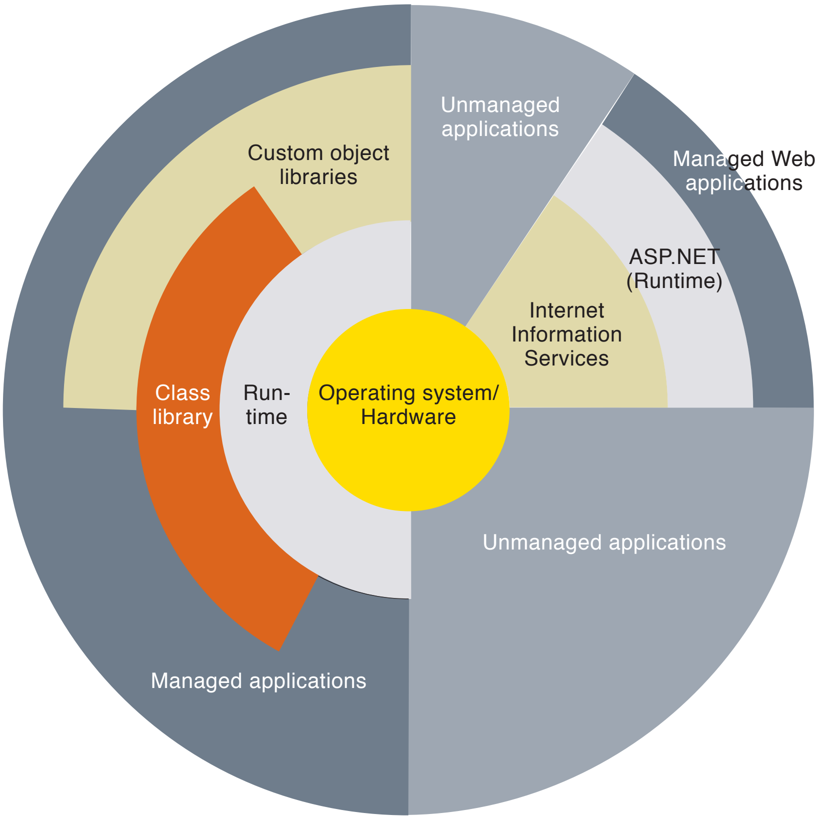
- Application clients and applets (dynamic HTML pages) are components that run on the client machine.

- Java Servlet and JavaServer Pages (JSP) technology components are Web components that run on the J2EE server machine.

- Enterprise JavaBeans (EJB) components (enterprise beans) are business components that run on the J2EE server machine.

A J2EE client can be a Web client or an application client. A Web client consists of two parts: dynamic Web pages containing various types of mark-up language (HTML, XML, and so on), which are generated by Web components running in the Web tier, and a Web browser, which renders the pages received from the server. A J2EE application client runs on a client machine and provides a way for users to handle tasks that require a richer user interface than can be provided by a mark-up language.

Figure 2 The .NET platform architecture



A Web page received from the Web tier can include an embedded applet. An applet is a small client application written in the Java programming language that executes in the Java virtual machine installed in the Web browser.

The server and client tiers might also include components based on the JavaBeans component architecture to manage the data flow between an application client or applet and components running on the J2EE server or between server components and a database.

The enterprise information system tier handles enterprise information system software and includes enterprise infrastructure systems such as enterprise resource planning (ERP), mainframe transaction processing, database systems, and other legacy information systems.

2.1.2 The .NET Platform

The .NET Framework is a new Microsoft computing platform that simplifies application development in the highly distributed environment of the Internet.

The .NET Framework has two main components: the common language runtime and the .NET Framework class library. The common language runtime is the foundation of the .NET Framework. You can think of the runtime as an agent that manages code at execution time, providing core services that manage memory, threads, remote access, security and robustness. In fact, the concept of code management is a fundamental principle of the runtime. Code that targets the runtime is known as managed code, while code that does not target the runtime is known as unmanaged code. The class library, the other main component of the .NET Framework, is a comprehensive, object-oriented collection of reusable types that can be used to develop applications ranging from traditional command-line or graphical user interface (GUI) applications to applications based on the latest innovations provided by XML Web services.

Figure 2 shows the relationship of the common language runtime and the class library to your applications and to the overall system. The illustration also shows how managed code operates within a larger architecture.

For example, ASP.NET hosts the runtime to provide a scalable, server-side environment for managed code. ASP.NET, including IIS (Internet Information Service), is a set of technologies in the Microsoft .NET Framework for building Web applications and XML Web Services. ASP.NET pages execute on the server and generate mark-up such as HTML, WML or XML that is sent to a desktop or mobile browser.

ASP.NET works directly with the runtime to enable XML Web services for example.

The .NET Framework can be hosted by unmanaged components that load the common language runtime into their processes and initiate the execution of managed code, thereby creating a software environment that can exploit both managed and unmanaged features. The .NET Framework not only provides several runtime hosts, but also supports the development of third-party runtime hosts.

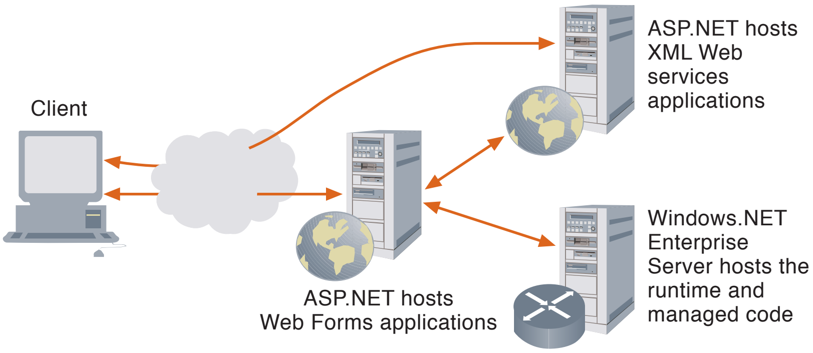

Client
ASP.NET hosts XML Web services applications
ASP.NET hosts Web Forms applications
Windows.NET Enterprise Server hosts the runtime and managed code

Figure 3 A basic .NET network

Figure 3 shows a basic .NET network with managed code running in different server environments. Servers such as IIS and SQL Server can perform standard operations while your application logic executes through the managed code.

ASP.NET is the hosting environment that enables developers to use the .NET Framework to target Web-based applications. However, ASP.NET is more than just a runtime host; it is a complete architecture for developing Web sites and Internet-distributed objects using managed code. Both Web Forms and XML Web services use IIS and ASP.NET as the publishing mechanism for applications, and both have a collection of supporting classes in the .NET Framework.

2.2 Production Platform

When developing a Web Service the developer must start to create the service functionality from scratch or use an existing service on the provider platform. The application server usually has tools that support service development. These or any other available tools can be used to make the necessary source code for the service to run on the targeted provider platform.

Most of the production platforms seem to contain similar functions:

- From source code as input is then the WSDL (Web Service Description Language) file produced. At the same time a deployment description is made and used to deploy the code into the actual runtime platform.

- If the WSDL file is already available, as it often is when the client side application is created, you could start from there instead. An existing WSDL file may also be used to create server-side code stubs of the Web Service defined by it.

- WSDL is an XML file used to create more or less automatically the Web Service interface (server-side) that exchanges and converts SOAP messages with XML code to invoke the server application and respond to the client application.

- The server-side code is deployed on the provider platform using the deployment description file.

- WSDL is also used to create the client application code stubs used to exchange SOAP messages with the server.

- The client-side code is deployed on the consumer platform.

It should be noted that the client-side and the service-side of the Web Service in theory could be produced by production platforms from different vendors. In practice, however, interoperability between Web Service clients and servers from different vendors cannot be guaranteed.

3 Major Web Service Products

Some of the most prestigious Web Service products known today have lately been tested and evaluated by Telenor R&D to some extent. These are BEA WebLogic Server 7.0 with Workshop, Apache Axis beta 3 Web Service Engine, Microsoft .NET and IBM WebSphere. The results of the findings are described in the following.

3.1 BEA WebLogic Server 7.0 with Workshop

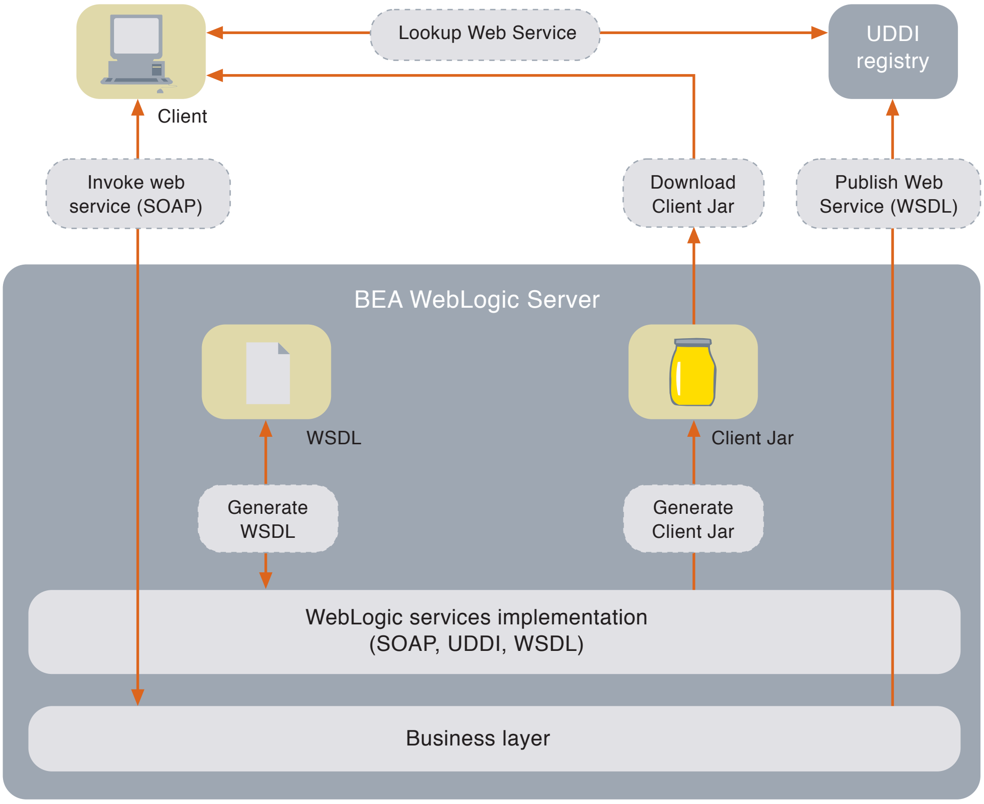
In our context is the WebLogic Server 7.0 the provider platform, while the Workshop is the production platform.

3.1.1 Overview

WebLogic Server (WLS) 7.0 is a fully J2EE-compliant application server, regarded as one of the leaders in the application server space because it has many Java-oriented features and is a reliable server. Even if SOAP and WSDL support is included in BEA WebLogic Server 6.1, they are not sufficient in this version. Compared to WLS 6.1, WLS 7.0 with Workshop extends Web service functionality and fulfils the need to integrate the developing environment with Web services support.

Workshop is the Web service creation and deployment tool for BEA's WLS 7.0. It provides

Figure 4 Web Service Architecture in BEA WLS 7.0



an integrated development framework that allows developers to build Web services that can easily be set up to interoperate with the existing resources and applications (EJBs, Java Message Service destinations, Java DataBase Connectivity connections and other Web services).

While WLS 6.1 forced developers to build part of the infrastructure, BEA Workshop provides this functionality for developers out of the box as part of the standard Web services infrastructure, encapsulating the complexities of the J2EE architecture. Workshop handles all the Java Naming and Directory Interface (JNDI) code, API calls, resource management, and exception handling, leaving the client with a simple interface for business logic calls. This allows application developers who are more business oriented than programming oriented to take advantage of Web services technology. BEA WebLogic Workshop includes two major components:

- A visual development environment that lets developers design and implement Web services. It graphically illustrates the interface with other components.

- A runtime framework that provides the Web services infrastructure, as well as testing, debugging and deployment environment for Web services applications. It creates the XML, SOAP and WSDL interface together with necessary components to implement the service under the J2EE WebLogic platform, which are EJBs, JDBC connections, etc, while the developer only worries about writing the standard Java code

Figure 4 shows what Web services look like in BEA's perspective.

3.1.2 Results

WebLogic Server 7.0 with Workshop supports large-scale development and deployment of XML Web services. The solution is totally integrated, which means that setting up and using the platform is relatively problem free.

WLS 7.0 and Workshop support SOAP 1.2, WSDL 1.1 and UDDI 2.0, and when it comes to SOAP messaging styles it supports both the RPC and document/literal styles. BEA claims Workshop Web services to be fully interoperable with .NET Web services, both when using .NET as a resource and when serving .NET clients. Our tests indicate that Workshop indeed offers true interoperability with both .NET and other J2EE services.

WLS 7.0 and Workshop constitutes a reliable, integrated platform with full support for every J2EE standard and all Web service standards. Combined with BEA's heavy attention to Web services and a nice visual development environment it is certainly a platform that should be taken into consideration when a reliable infrastructure for Web services is needed.

3.2 Apache Axis beta 3 Web Service Engine

Axis will together with a J2EE compatible application server support a provider platform, a client platform and a production platform.

3.2.1 Overview

The open-source, freeware Web services container offered by the Apache project is called Axis (a complete re-architecture of Apache SOAP), and stands for Apache Extensible Interaction System. The first (alpha) version was released less than a year ago, and the current version is 1.0 RC1 (candidate for 1.0 release).

Axis is a SOAP engine that acts as a client and server of SOAP messages with primary focus on HTTP. It can be viewed as a thin layer sitting between the business logic and the network transport. It runs as a Web application on top of an application server. It comes with a standalone server, but is most commonly used as a Web application on top of Tomcat. In our evaluation, however, we use WLS 6.1 as the underlying platform.

Axis provides extensive support for WSDL, and with Axis' utilities one can generate WSDL from Java classes implementing the actual service. Starting with a WSDL an entire package of client-side proxy and server side skeleton classes can be built. Axis also includes a Java application called "tcp-mon" that allows you to monitor the SOAP messages that are being sent to and from an Axis engine.

The Axis engine's main job is to pass SOAP messages through a set of handlers that examine and/or modify a SOAP message in order to complete its job. Figure 5 shows how the Axis engine works on the server-side, i.e. in the provider platform, and Figure 6 illustrates how it works on the consumer platform when Axis is used to run a Web service client.

3.2.2 Results

Although Axis is constantly evolving, at the time of writing some limitations exist. First of all, with Axis one cannot consume or generate services that use unsigned XML schema types, something which might raise interoperability

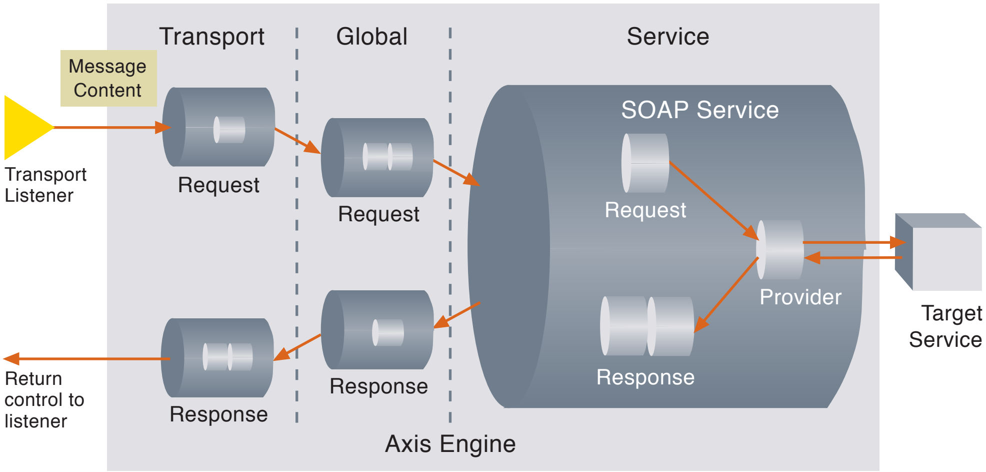
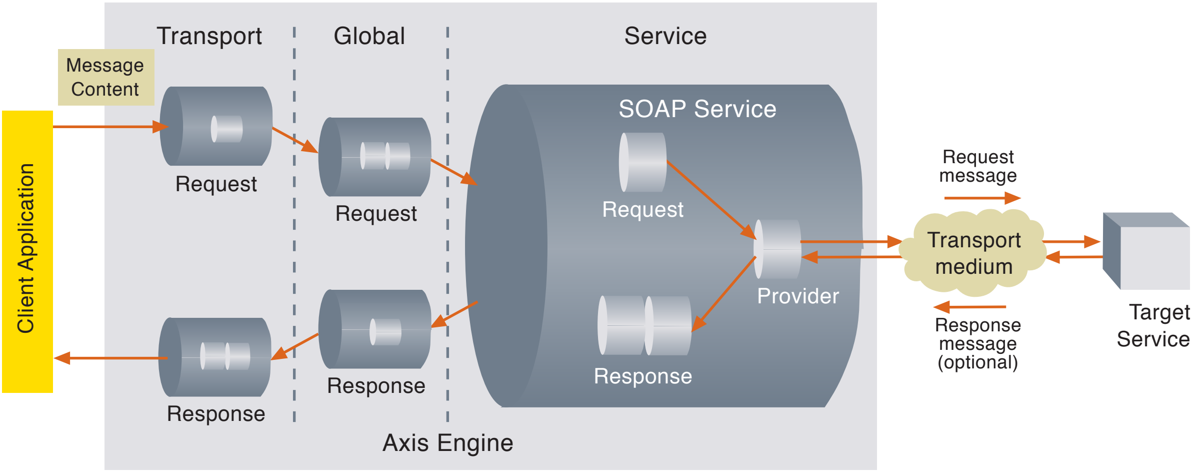


Figure 5 Axis Engine Server Architecture

Figure 6 Axis Engine Client Architecture



problems, especially when consuming .NET Web services. The reason is the lack of unsigned data types in Java. The next release of Axis will certainly include support of this.

Another thing is that you cannot send arbitrary Java objects over the wire and expect them to be understood at the remote end. Axis will only send objects for which there is a registered Axis serializer. To serve up objects in general you must build the serialization support into Axis yourself.

Because Axis developers find the SOAP specification to be unclear, Axis neither supports RPC calls in SOAP headers nor multiple RPC calls in a single SOAP message.

Axis supports WSDL 1.1, but does not support UDDI at all.

Axis supports SOAP headers, but only with limited information so that e.g. .NET interoperability may be compromised.

It also has preliminary support for the SOAP with attachments specification.

Axis did well on our interoperability tests, except that

- When publishing WSDL files which specify port numbers (e.g. 7001), the port number is for some reason not recognised when the WSDL file is imported in .NET, thus the .NET client must be edited manually to add the correct port number.

- Axis does not support unsigned data types, and it is thus difficult to make clients for services that does use unsigned data types in their services (e.g. .NET). A solution to this problem is already possible to download, and in version 1.0 it will certainly be fixed.

Our evaluation shows that Axis is a Web service engine with great potential because of its focus on specifications and its development team not belonging to one single commercial company. However, Axis users must still reconcile themselves to the fact that a lot of useful functionality is still not implemented, and that bugs will exist. Combined with poor and erroneous documentation and no professional support organisation it is not recommendable for anyone to use Axis for production of Web services.

However, skilled J2EE developers may find Axis a perfectly OK implementation that can be used for experimenting with Web services development on any J2EE platform, and for that purpose its a powerful tool.

3.3 Microsoft .NET

.NET is a product that does support all aspects of Web Services, i.e. provider, consumer and production platforms as well as management.

3.3.1 Overview

.NET is Microsoft's newest platform for developing and running applications, competing with the J2EE standard. It contains Web services functionality as an integrated part. Visual Studio is a development environment for developing applications.

.Net integrates technologies that have been evolving during the last years, such as COM+, ASP, XML and Web services, based on protocols like SOAP, WSDL, UDDI and HTTP.

.Net's architecture is structured as follows:

- Development Tools
  - A set of languages, including C#, VB.NET, C++.NET
  - A RAD developing environment: Visual Studio 7.0

- Specialized servers
  - SQLServer 2000, Exchange Server 2000, BizTalk Server, Commerce Server 2000, ISA Server.

- Web Services
  - Integration with internal or external services that can be bound together through the Internet.

- Devices
  - PCs, handhelds, mobile phones, gaming consoles.

The main limitations of .NET are the proprietary format, which makes application portability difficult and gives interoperability problems when communicating with non-.NET services or clients.

The main advantage of .NET is that it promises to provide a simple and quick service and application development, especially web based, by exploiting all of the experience with the previous tools. By the "Plug&Play" deployment of the applications, the installation and distribution will be highly simplified. At the same time, by means of the reusability paradigm, the productivity of programmers will be raised.

Microsoft's solution for implementing Web services is a complete approach for programmers who do not have much experience, or who are already familiar with languages like C++, VisualBasic, or any other language supported by the platform. Introducing Microsoft's new programming language C# (or c sharp), Microsoft is trying to compete in the inter-platform market against Sun Microsystems' Java. Microsoft presumes its new language is as powerful as Java, but it is still very early to even consider it true, although it does seem to have a large potential for gaining some terrain on the programming languages competition.

Using the .Net platform provides a big aid for developing Web services, but it is a release that is still in its early stages, so many bugs, or problems such as interoperability issues, should be expected to be discovered soon.

An additional advantage that .Net provides is a thorough on-line documentation, which can be used for everything from understanding the architecture of the platform to finding class definitions and class descriptions for referencing them. At last, .NET also includes an interface that tries to reduce, as much as possible, the necessity of writing low-level code.

3.4 IBM WebSphere

To facilitate building, deploying and enhancing e-business applications IBM has made WebSphere, which includes three product families that all together support the provider, consumer, production and management platforms for Web Services:

- WebSphere Application Server that is a scalable platform to deploy dynamic e-business applications.

- WebSphere Studio that is a set of development tools for e-business and Web Service applications based on one common workbench technology (Eclipse).

- WebSphere Host Integration that offers tools for application and data access to legacy systems.

3.4.1 Overview

There exist different editions of the Application Server that are adjusted to fit the right size of a company:

- Small companies – WebSphere Application Server, Version 4, Advanced Single Server Edition;

- Mid-size companies – WebSphere Application Server, Version 4, Advanced Edition;

- Large companies – WebSphere Application Server, Version 4, Enterprise Edition.

The more advanced the edition is, the more functions and facilities are included. However, the functionalities mentioned below do count for all editions. The main components are shown in Figure 7.

IBM WebSphere Application Server is a comprehensive Java technology-based Web application server providing integrated support for J2EE and Web services. The J2EE compatibility means that the full range of Java technology-based APIs and protocols are supported, e.g. Enterprise Java Beans (EJB) that is regarded as probably the most significant contribution.

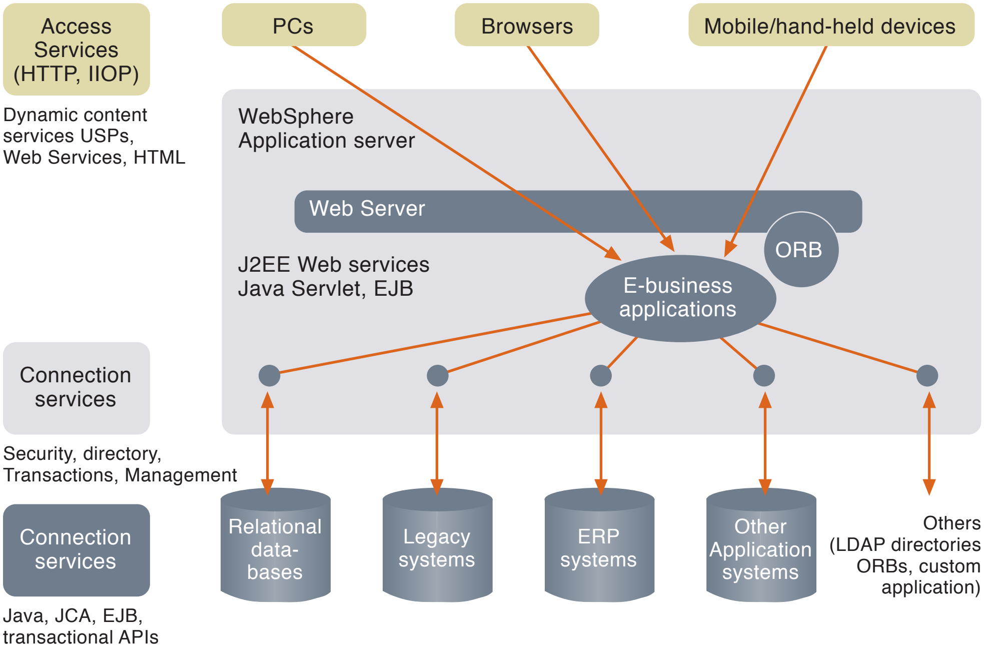

Access Services (HTTP, IIOP)
Dynamic content services USPs, Web Services, HTML
PCs
Browsers
Mobile/hand-held devices
WebSphere Application server
Web Server
ORB
J2EE Web services Java Servlet, EJB
E-business applications
Connection services
Security, directory, Transactions, Management
Connection services
Java, JCA, EJB, transactional APIs
Relational data-bases
Legacy systems
ERP systems
Other Application systems
Others (LDAP directories ORBs, custom application)

Figure 7 Components in IBM WebSphere Application Server

Due to the Web service support it is easy to build, customize and publish services based on the XML, SOAP, UDDI and WSDL standards. The extensible and open programming model and architecture of the WebSphere Application Server can enable J2EE applications to swiftly interoperate with Web service applications.

The WebSphere Application Server contains the J2EE Connector Architecture (JCA) that may be used to create resource connectors or adapters for accessing external enterprise systems. In addition, the Application Server does support Java Database Connectivity (JDBC) that enables access to a range of database servers, such as Microsoft SQL Server, Oracle and Sybase.

WebSphere Studio provides an industry-leading integrated development and runtime environment for the Application Server. In addition the Studio has a workbench based on open standards that provide plug-and-play capability for third-party application development tools.

IBM WebSphere Host Integration Solution can take legacy application to the Web quite quickly. It extends host applications to the Web and provides software for creation and deployment of new host access e-business applications, without requiring any changes to the existing applications themselves. Whether one needs a simple Web page delivery, putting a new face on a legacy application, or creating sophisticated Java solutions, the Host Integration Solution allows one quickly and flexibly to integrate critical enterprise data with the Web.

Administrators are offered a so-called *WebSphere Administrative Console* used to configure and control a WebSphere domain. A WebSphere domain is made up of one or more physical machines sharing a single WebSphere administrative database, which holds information on each of the components running on those machines. To effectively manage a WebSphere environment, the administrator should be familiar with the concepts underlying the J2EE runtime environment, e.g. containers, J2EE enterprise applications, Web modules, EJB modules, XML deployment descriptors, etc.

3.4.2 Results

IBM provides one of the foremost platforms for developing Web services today. With its strong involvement in open projects such as Apache, IBM has ensured that it uses state-of-the art technology and follows standards strongly. WebSphere has long been one of the most popular application servers on the market, and companies that want to take advantage of Web services technology should not be afraid to develop their services on this platform.

4 Web Services in Mobile Telecom Business

UMTS is the next generation of mobile systems, including networks, terminals and services. With UMTS the basic Internet protocol, IP, will be fully integrated in the mobile networks. This will open up for Internet Services in the mobile environment that technology-wise is almost the same as in the fixed environment, e.g. WAP and i-Mode used by small mobile terminals. At the same time a great interest is also foreseen in having access to pure Telco services, such as call control and terminal location, from Web application to make new powerful integrated services.

4.1 Web Service Access to Mobile Telco Services

The standardization of how third parties may access telco services in UMTS is done mainly within the Parlay group and 3GPP standardization body. See http://www.parlay.org/ and http://www.3gpp.org/. Currently APIs for the following UMTS services are specified by Parlay/OSA (see reference 3GPP TS 22.127 V5.2.0 (2001-12)):

- The Framework services managing the access to the available Telco services; i.e. Trust and Security Management (Authentication, Authorization), Service Registration, Service De-Registration, Service Discovery and Integrity Management;

- Network services; i.e. Call Control functions (Circuit Switched, Packet Switched), IM Session Control, Information Transfer and Charging;

- User data related services; i.e. User Status, User Location, User Profile Management, User Profile access Authentication/Authorization, Terminal Capabilities and Functions and Retrieval of Network Capabilities;

- Information Services;

- Presence related services.

All APIs so far have on a high level been specified in UML and mapped on Corba IDL for more detailed specification.

The Parlay Group has recently released version 3 of the Parlay specifications, which is completely consistent with the 3GPP OSA standard, and the work on Parlay 4 has started. In the work program a working group is identified, termed ParlayX, which aims to integrate XML and Web services in future versions of Parlay. Their current picture of how this may be done is shown in Figure 8.

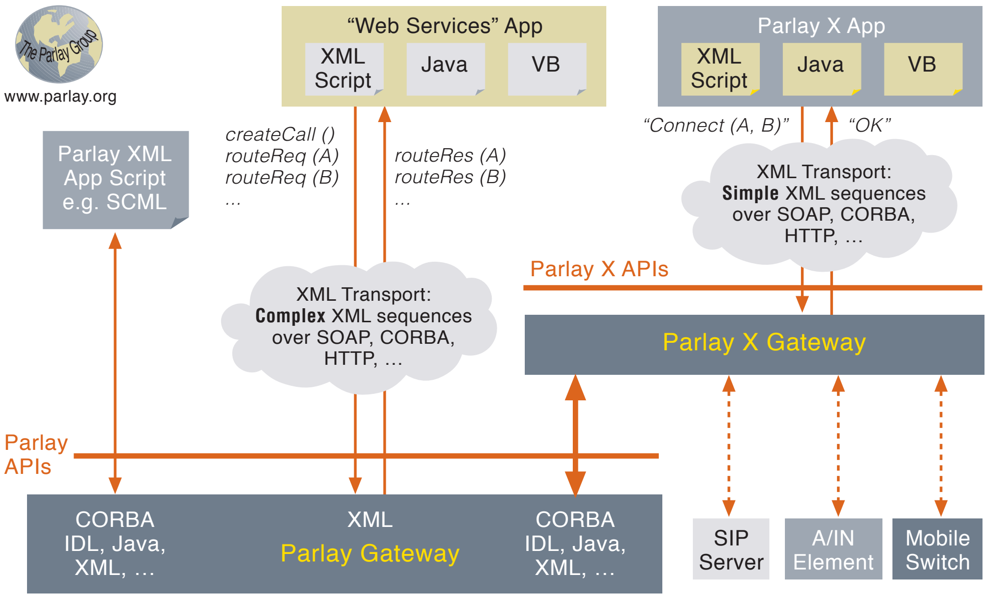


Figure 8 XML Realizations of Parlay APIs and Parlay X APIs

The Parlay X group intends to integrate XML in the interface towards the telecom service capabilities in two different ways:

- A UML to XML mapping for the Parlay 3 specification. This means that the third party Web Service application sees the same complex APIs as the Corba applications do, and requires some telecom skills by application developers. The XML transport may be quite complex.

- A new Parlay X Gateway is introduced at a higher abstraction level with a simpler set of APIs. Usage of those APIs does not require any telecom skills from the application developer at all. The XML transport is kept simple.

4.2 Mobile Terminals and Web Services

Both Microsoft and other Web Service vendors are very eager to develop applications on mobile devices.

4.2.1 Microsoft

Pocket PCs (P/PCs) and Handheld PCs (H/PCs) are Microsoft's own classification of mobile devices and powered with variants of Windows (CE), respectively Handheld PC 2000 OS version 3.0 and Pocket PC 2002 edition. The difference between these two is a bit diffuse, but we may say that P/PCs are quite comparable with PDAs while H/PCs are more like small laptops or notebooks. Typically, H/PCs are equipped with keyboards while P/PCs are not.

The developing frameworks and toolkits for H/PC and P/PC are based on Microsoft .NET, but compressed with respect to functionality and size so they fit the smaller end tinny mobile devices. Microsoft has powerful graphical toolkits that seem to reduce the developing time of mobile applications quite dramatically. When starting with a new or existing Web Services (.NET) based application as input to the mobile toolkits, the same application could be modified to run on H/PCs or P/PCs with minimal effort. Powerful terminal emulators show immediately what the applications would look like when deployed in the real environments.

The toolkits are available for free trial to everybody. The Mobile Solutions Developer Toolkit may be requisitioned from http://www.msdn.microsoft.com/vstudio/productinfo/trial.asp for 60 days free trial. Other tools are Microsoft .NET Framework SDK and Visual Studio.NET Smart Device Extensions.

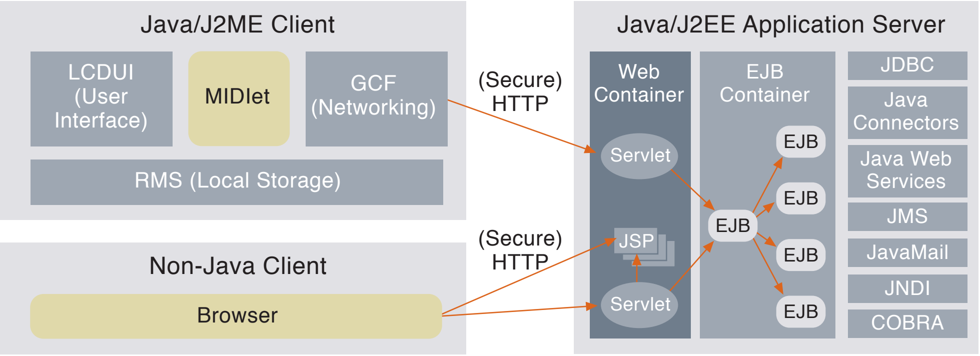
Microsoft also seems quite eager to cooperate in the context of mobile services with telecom operators all over the world.

4.2.2 The Other Vendors

Microsoft's competitors seem to base their mobile terminals on the Java 2, Micro Edition (J2ME) framework, which is the client-side counterpart of the J2EE server-side applications, or other dedicated platforms for mobile devices, e.g. WAP or i-Mode.

J2EE platform emphasizes reusable components through the use of enterprise beans. Applications may leverage these components to support multiple types of clients with little, if any, impact on the core business logic of the application. Figure

Figure 9 High-Level architecture of a J2EE application supporting a J2ME client and a browser client



9 shows the architecture of an application with a J2ME client and a browser client, e.g. a WAP client.

The MIDlet is an application that runs on a J2ME platform and conforms to the MIDP (Mobile Information Device Profile) standard.

5 Conclusion

Web Service products from vendors who are regarded as leading actors, are investigated in the previous chapters. However, there are several other products from not so prestigious vendors we have not looked into. So our conclusions are based on a rather limited selection of products.

In a quite early stage of the development of Web Services technology our conclusion is that there are several products mature enough for commercial utilization. The .NET, WebSphere and Workshop are all integrated platforms that provide nice graphical user interfaces and fully automate most operations involved in the Web service development. However, Apache Axis Web service engine is ready to be used, at least for testing and experimenting projects. We believe this platform will mature more in the coming months, and skilled J2EE developers may find it to be an excellent tool. Using Axis today is far from as simple as using one of the aforementioned integrated platforms.

Even if our investigation does not point out a winner, it shows that a developer has to choose between two types of Web Service products, i.e. those based on Java platforms (J2EE/J2ME) or those based on Microsoft platforms. The traditional competition between the Java world and the Microsoft world seems to continue in the field of Web Services as it has in many other fields of technology. Which type of Web Service product a developer chooses, will probably depend on his business policy. What is most favourable – being compliant with the Java platform or the Microsoft platform?

Security in Web Services

ERIK PARR, ERIK BERG AND SUNE JAKOBSSON

Erik Parr (26) received his Siv.Ing. degree (Ingénieur Diplômé) in Mathematical Modeling from the Technical University INSA Toulouse in 2000, including internships at Sintef ECY and Elf Center for Petroleum Research. Following his military service, he has worked in Telenor R&D in Trondheim since October 2001. His current research interests are in computer security, middleware and service platforms.

erik.parr@telenor.com

Erik Berg (27) received his Siv.Ing. degree (~MSc) in Telematics from the Norwegian University of Science and Technology (NTNU) in January 1999. He has worked in Telenor R&D as Research Scientist since February 2000 in the fields of middleware, e-commerce and dependable distributed systems.

erik.berg@telenor.com

Background

Web Services is a new set of technologies with the ambitious and important goal of tying together logic from distributed components and applications on the World Wide Web (WWW). Web Services, as the name indicates, are supposed to run on the WWW and its well-established protocol HTTP. The Internet has been around for some time, and its basic technologies are mature. Although the term Web Services has a new ring to it, the mechanisms for ensuring Web Service security are based upon well-established and tested technologies.

At the core of WWW security are the mechanisms to secure communications in the client-server model. Web Servers have the possibility of implementing authentication and encryption mechanisms like HTTPS to keep certain parts of client-server message exchange confidential. The client browser usually stores all the user's credentials in the form of digital certificates, which are used to access secure Web servers.

So what is actually the difference between evaluating WWW and Web Services security? First of all, WWW security is doubtlessly the most important part of Web Services security. Mechanisms and know-how with WWW security will be applicable and reused in Web Services security. Particularly important are the well-established authentication and encryption mechanisms used by Web servers. However, the way we see it a few additional points should be taken into account when evaluating Web Services security:

- Web Services security must address the effects of exposing systems that earlier were isolated;
- Web Service security must take into account that most hackers are likely to be familiar with weaknesses in Internet security technologies. At least, CORBA had the advantage of staying out of the typical hacker's electronic back yard;
- Web Services security must set requirements for SOAP security;
- Web Services security must define ways to secure *private* intra- and inter-company services in addition to *public* WWW services;
- Web Service Security must address standardisation issues to promote interoperability.

Introduction

The goal of this article is to give an overview of Web Services security, both the part that is based on established WWW/HTTP technologies and also the new components and protocols such as UDDI and SOAP. Web Services represent a new way of doing and organising business, and we will try to identify security requirements in some typical business scenarios. In the subsequent section, we identify and evaluate current Web Service technologies, and look at their prospects of fulfilling the identified security requirements.

In the Standards section, we will take a look at current Web Service standardisation efforts.

To see how the Web Service security requirements are met in practice, we will close our discussion by giving an overview of security mechanisms on the current state-of-the-art Web Services platforms.

We conclude with identifying important success criteria and pit-falls to avoid in order for Web Services to evolve harmoniously, fulfil security needs and build trust vis-à-vis end users.

Basic Web Services Glossary

- *XML* – eXtensible Markup Language; Universal language for defining data schemes. XML separates content from data format. Tags are used to separate data.
- *SOAP* – Simple Object Access Protocol; SOAP is essentially a one-way messaging protocol, which defines a uniform way of passing XML encoded data. The SOAP specification describes how SOAP over HTTP can be used for remote procedure calls. SOAP implements a client-server model over a transport protocol, most typically HTTP but also FTP, SMTP etc.
- *UDDI* – Universal Discovery Description and Integration; A set of specifications for creating XML-based directories of Web services offerings. The UDDI is separated in white, yellow and green pages. You can either publish() or inquire() in the UDDI directory. When nested in a global net, UDDI is the Web services equivalent to the CORBA naming service.
- *WSDL* – Web Service Description Language; A common framework for describing tasks

performed by a Web service. A WSDL file is used to generate the client proxy and thus make a Web service programmatically accessible to other applications.

- *HTTP* – HyperText Transfer Protocol; The protocol used to transport data on the WWW.

- *UUP* – Universal User Profile; although not yet standardized, UUP is a set of user-specific data and preferences stored on the WWW to perform tasks such as authentication and personalisation.

Web Services Security Requirements

Security Policy and Web Services Requirements

Simply put, one could say that securing Web Services is securing the interaction between its *components* according to certain *criteria*. These criteria are set down in a *security policy*, where the perceived threat of compromise or fraud and the eventual cost of bad publicity are weighed against the cost of implementing a technical safeguard.

A good security policy can help make the Web Service stable internally and predictable externally through the API. Therefore, a key element to secure Web Services is to implement a simple and consistent security policy. The policy should address basic issues such as identification and authentication, authorization, auditing, the need for data integrity and confidentiality, non-repudiation and privacy (see Figure 2).

The content of the security policy also depends on the business environment and the type of Web Service we consider. As a result, the requirements might be quite different for a fully public and a private Web Service. In addition, information about the end users such as the scale of the end user group and the character of the average end user should be taken into account when setting security requirements.

As an example of setting requirements for Web Services, let us consider the fully public Web Service myService.com. In this context, public means that everybody with a WWW connection can subscribe to the service. The components of a public Web Service are given in Figure 3.

Before myService.com provided by myCompany can go 'on the air', three basic steps have to be carried out. For each of them a set of security requirements has to be met:

Step 1 – Publish():

We suppose that the Web Service provider myCompany already has bundled the business logic into an application and made a WSDL file to describe the service interfaces. In order to make its service attainable through the WWW, it then deploys the application on a Web application server like the one given in Figure 3.

Now, to use myService.com, the client application has to know of its whereabouts. *Publishing* a Web Service is about registering the service with a UDDI service broker, which is the equivalent of a yellow pages registry. A client (application) can browse through this registry looking for a service that exactly suits it needs and obtain a reference or an address to that service[1].

Now that we have described what we mean by publishing a Web Service, we will try to identify the security requirements for the publishing process:

- There is a need for *Identification*, *Authentication* and *Authorization*. We need to make sure that nobody but the legitimate owner – myCompany – creates or modifies myCompany's UDDI record. Otherwise, an opportunistic business opponent – myWorstOpponent – might wilfully change the information in the registry.

- Depending on the security level desired, one could add mechanisms to ensure *integrity*, i.e. a guarantee that the correct information about the service is not altered between the application server and the UDDI registry. Appropriate integrity mechanisms would have a preventive effect of sophisticated man-in-the-middle attacks, but would also have the effect of preventing simple denial of service attacks: If, for example, the information about where the WSDL file is kept is altered en route, there is no way for a client application to access the service.

Step 2 – Find():

Find() is the process in which a client (application) browses through the UDDI yellow pages looking for a service that suits its needs. When the application finds a suitable service – for example myService.com – it obtains a reference to the service's WSDL file.

Sune Jakobsson (42) is Research Scientist at Telenor R&D Generic Service Platform Group. The group is active in research of tomorrow's computing platforms for telecom operators. Currently he is an active participant in the EURESCOM P1209 and P1242 projects, with focus in the area of Web Service usage for telecom operators. He received his Siv.Ing. degree from the Norwegian University of Science and Technology (NTNU), Faculty of Electrical Engineering and Computer Science in 1994. He joined Telenor in 1998 from Stentofon ASA, where he worked as HW designer at designing intercom switches.

sune.jakobsson@telenor.com

[1] *This process is in many ways similar to looking for a suitable repair facility for your car, except for the fact that there are no geographical limitations to Web Services whereas the repairman has to be situated somewhere in your geographical neighbourhood.*

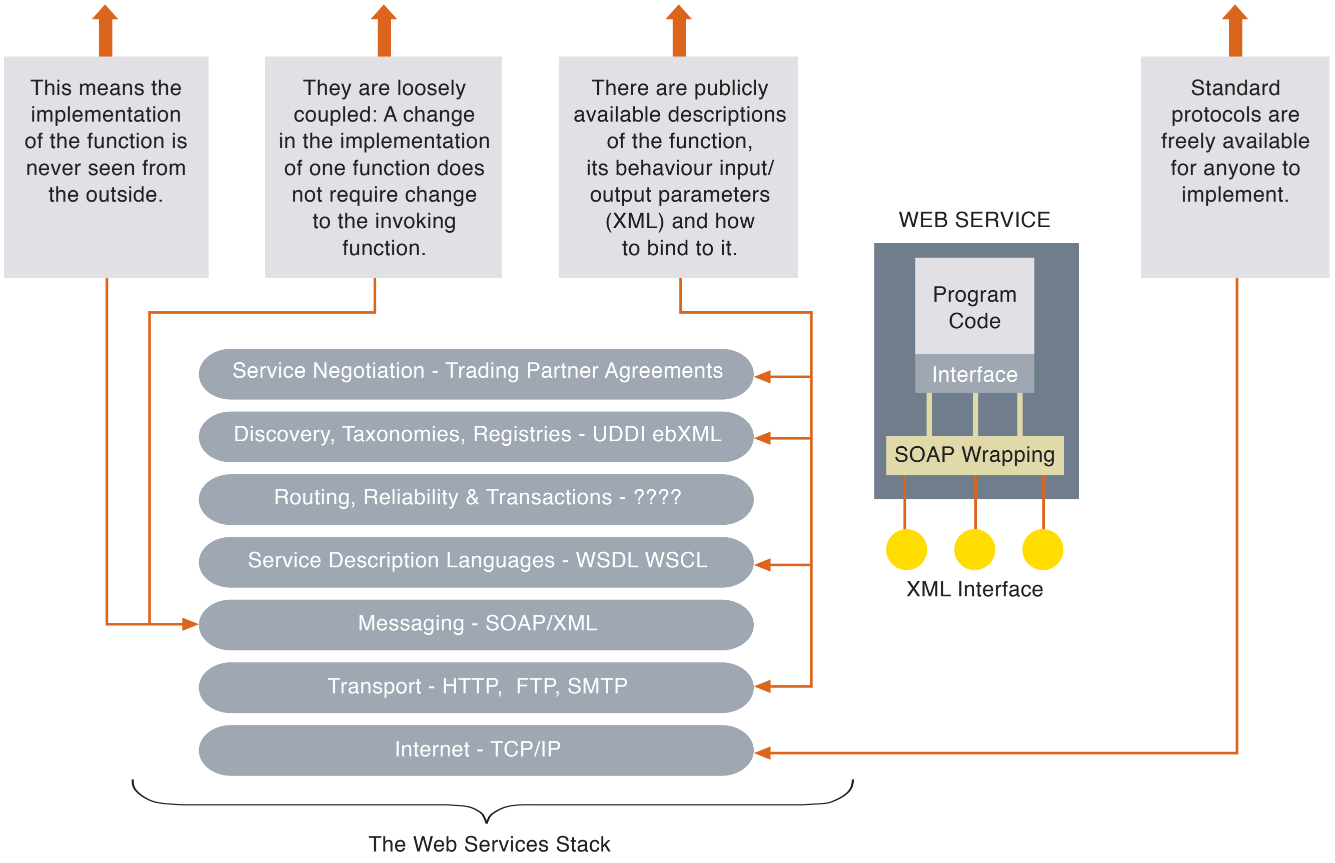

This means the implementation of the function is never seen from the outside.
They are loosely coupled: A change in the implementation of one function does not require change to the invoking function.
There are publicly available descriptions of the function, its behaviour input/ output parameters (XML) and how to bind to it.
Standard protocols are freely available for anyone to implement.
WEB SERVICE
Program Code
Interface
SOAP Wrapping
XML Interface
Service Negotiation - Trading Partner Agreements
Discovery, Taxonomies, Registries - UDDI ebXML
Routing, Reliability & Transactions - ????
Service Description Languages - WSDL WSCL
Messaging - SOAP/XML
Transport - HTTP, FTP, SMTP
Internet - TCP/IP
The Web Services Stack

Figure 1 XML Web Services

Since the integrity of the information in the registry is part of the publishing procedure, no particular requirements are tied to the find procedure, except:

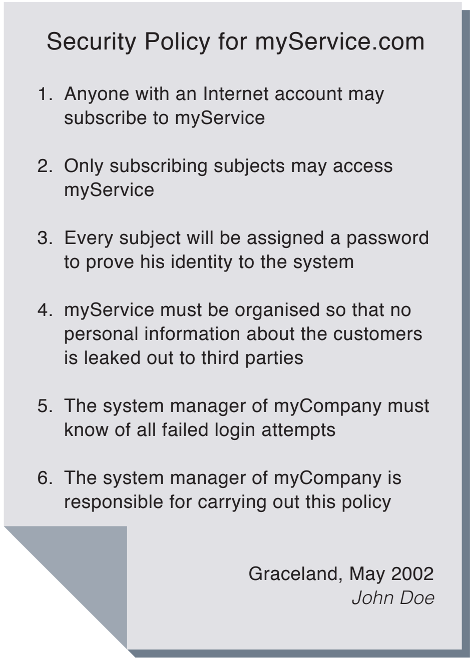
- To obtain a high security level there might be a need for the Web Server hosting the registry to authenticate to the client application. This avoids a possible cloning of a UDDI registry by a potential perpetrator. Server authentication is a standard part of the HTTP protocol, and is often conducted transparently.

Step 3 – Bind():

Bind() is the process in which a client application looks up a service's WSDL file in order to create a local *interface* to the service. The process of making the remote procedure call then is handled by 'invisible' Web Services middleware. The interface – or *stub classes*[2] – acts as a proxy for remote object on the local machine, encapsulating and hiding the communication protocol(s) from the programmer. This way, the programmer only sees the remote interface, not the actual SOAP and HTTP messages. The conversion of data and values is often referred to as *marshalling* and *demarshalling*.

Creating stub classes from a service's WSDL file can be done automatically using an appropriate tool, or manually.

For the public Web Service we consider in this example, we have not identified any compulsory security requirements for the Bind() process. For



Security Policy for myService.com

1. Anyone with an Internet account may subscribe to myService
2. Only subscribing subjects may access myService
3. Every subject will be assigned a password to prove his identity to the system
4. myService must be organised so that no personal information about the customers is leaked out to third parties
5. The system manager of myCompany must know of all failed login attempts
6. The system manager of myCompany is responsible for carrying out this policy

Graceland, May 2002
John Doe

Figure 2 Security policy for the public Web Service 'myService.com'

[2] *In object oriented languages such an interface or proxy is most commonly implemented by so-called* stub classes. *A* stub class *is the local representative of a remote object. When the client application wants to invoke a remote object, it does this through its stub classes.*

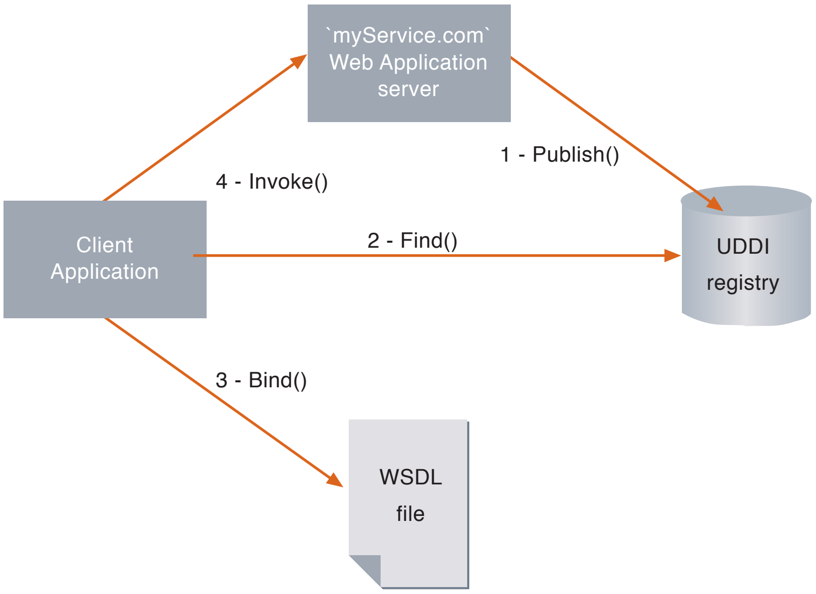


Figure 3 Components in a possible deployment scenario for myService.com

high-level security, one could consider some kind of integrity mechanism to avoid that the content of the WSDL file is changed when it is downloaded to the Client machine.

Step 4 – Invoke():

With the three previous steps the myService.com client-server battery is finally fully operational. The final step naturally involves the actual invocation of a Web service. The service can be invoked on the local proxy, which handles all the necessary communications with the myService.com Web server.

Security requirements for the invoke() operation correspond to what we could call *runtime* security requirements for myService.com. In the security policy given in Figure 2, the following requirements can be identified:

- Identification and authentication: #3 states that all users will receive a password to access myService.com. So a password-based authentication scheme must be implemented.

- Authorization: According to their security policy (#2), myCompany needs to make sure that only subscribed users can access myService.com.

- Auditing: As a minimum, to meet #5, rejected login attempts have to be logged.

- Privacy: #4 – No personal subscription information must leak out. This could introduce a need to keep certain parts of the message exchange confidential.

- Confidentiality (#4 – see above).

Remark: Although *integrity* and *non-repudiation* are not explicitly addressed in the myService.com security policy, they should probably – as a general rule – be considered in connection with 'real' public Web Services.

Web Services Security Mechanisms

Now that we have identified Web Services security requirements in a public environment, we need to identify which technical protection mechanisms we possess to deal with these requirements. For a given security policy and a list of requirements, an appropriate subset of 'all possible' security mechanisms should be chosen and deployed.

Securing Databases and Registries

Protecting databases and registries by means of encryption and access control mechanisms is fairly straightforward. Technologies for securing databases can be considered well established and mature.

However, in the introduction to this article, we suggested that Web Services security should address the effect of exposing systems that were earlier isolated. Many databases and registries either have been or at least have the basic characteristics of such isolated or standalone systems. So clearly some care should be taken, in particular with the interfaces exposing the data.

Securing Communications

However, the protection of data locally only solves a minor part of the problem. Like we pointed out in the previous section, the major challenge that is introduced by the Web Service security requirements is to secure data transport between the different components. Combining mechanisms at different levels of the Web Services protocol stack can help secure data transport (see Figure 4).

In the following discussion, we will take a closer look at the two main Web Service transport protocols HTTP(S) and SOAP and see what requirements can be met by a) either one, or b) by their combination.

HTTP Security (HTTPS)

There have been several attempts at securing the HTTP protocol to ensure secure access to the World Wide Web. The Secure Socket Layer (SSL) protocol is situated between HTTP and TCP/IP in the protocol stack and was a popular solution for transport layer security in the mid 90's. In 1996, the Transport Layer Security (TLS) Working Group was established by the Internet Engineering Task Force (IETF) in an attempt to standardize a 'transport layer' security protocol. This Working Group is responsible for developing the TLS protocol, which is intended to replace SSL.

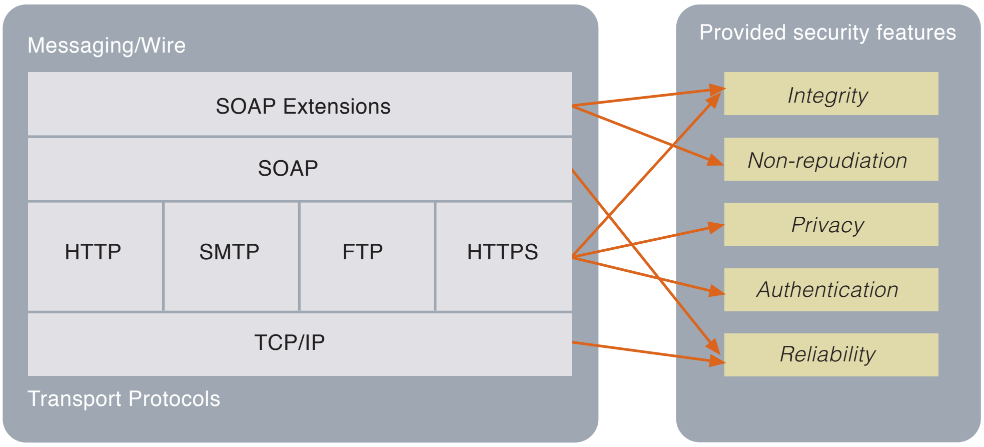


Figure 4 Web Services protocol stack and the basic security issues addressed by each layer. In the pre-WS era, HTTPS over TCP/IP was the main mechanism to enable World Wide Web security

The combined protocol HTTP/TLS or SSL is often referred to as HTTPS (see Figure 4). In the following discussion we will mainly focus on TLS, knowing that the evaluation of SSL would give quite similar results.

TLS consists of two main parts: A handshake protocol and a record protocol. The goal of the 'handshake' is to authenticate the server and to negotiate the cryptographic protocols to use for data transfer. User authentication is optional. The 'record' part assures the transfer of encrypted data between the server and the browser using the negotiated cryptographic techniques combined with a shared session key to ensure the integrity and the privacy of the message. Use of Public Key Infrastructure (PKI) for session key exchange during the handshake phase has been quite successful in enabling Web commerce in recent years.

TLS also has some known vulnerabilities. For instance, the Handshake protocol is open to so-called man-in-the-middle attacks because parameter negotiations tend to be in the open. Another attack exploits the fact that TCP is a reliable protocol: By discarding packages it is possible to jam the TCP reliability mechanisms and conduct a simple DOS attack.

Running HTTPS-based applications or services requires that both the Web Server and Web Browser support TLS (see Table 1).

To summarize, the main goal of HTTPS is to provide privacy, integrity and authentication mechanisms to secure web navigation and messaging. It is a well-established and thoroughly tested technology. On the downside, despite the possibility of TCP session resumption, HTTPS is significantly slower than HTTP run alone.

A non-exhaustive list of alternatives to HTTPS includes IPsec, S-HTTP, SET and SASL.

SOAP security

SOAP is an XML-based protocol for sending messages and making remote procedure calls in a distributed environment. Besides XML, SOAP is the cornerstone of the Web Services infrastructure. In principle, SOAP can be used with any kind of transport protocol including FTP and SMTP, but at the moment the only commercially available mapping is on HTTP.

The SOAP specification does not address security issues directly, but allows for them to be implemented as extensions. For example, the extension SOAP-DSIG, which has been submitted to the World Wide Web Consortium (W3C), defines the syntax and processing rules for digitally signing SOAP messages and validating signatures. Digital signatures in SOAP messages provides integrity and non-repudiation mechanisms. This and other extensions may turn out to be a useful supplement to the security provided by HTTPS – see standardisation section below.

SOAP is designed to pass through firewalls as HTTP. This is disquieting from a security point of view. Today, the only way we can recognize a SOAP message is by inspecting HTTP content and parsing XML at the firewall. But since SOAP calls are so diverse with no uniform addressing model or reliable internal structure even that does not solve the problem of which calls to trust. According to [1] SOAP and WSDL

Table 1 A (non exhaustive) list of Web Servers and Web Browsers that support HTTPS

Web Servers supporting HTTPS	Web Browsers supporting HTTPS
Apache-SSL (open SSL libraries)	Internet explorer
Apache mod_ssl (open SSL libraries)	Netscape
Stronghold	Opera
Roxen	Cryptozilla
INetStore	
Tomcat	

make no distinction between reads and writes on a method level, making it impossible to filter away potentially dangerous writes. This means that a method either needs to be fully trusted or not trusted at all. SOAP messages are designed to be free form, which makes log files hard to understand and analyse in a consistent manner.

SOAP is fairly new and untested. This means that the likelihood of problems showing up along the way are much greater than with for instance HTTP(S), which is well established and tested for shortcomings.

To summarize, SOAP is a simple and completely platform- and programming language-independent protocol. Unfortunately, simplicity comes with a cost. SOAP deliberately dodges firewall filtering by tunnelling over HTTP, and since SOAP calls are so diverse, they are unpractical to filter and audit. So uncritical use of SOAP will represent a security risk, at least until appropriate security tools have been deployed and the information community has had a chance to properly test the protocol.

Public Key Infrastructure (PKI) – a Key Enabling Technology

PKI key management provides a sophisticated framework for securely exchanging and managing keys. The two main technological features that a PKI can provide to Web Services are:

- *Encryption of messages*: by using the public key of the recipient;

- *Digital signatures*: by encrypting a message with your own private key so that anyone with your public key can decrypt it and furthermore know for certain that you wrote it. Non-repudiation mechanisms provided by PKI and defined in SOAP standards may provide Web Service applications with legal protection mechanisms.

Note that the features provided by PKI address the same basic needs as those that are recognized by the standardisation organisations as being important in a Web Services context.

In Web Services, PKI mainly intervenes at two levels:

- At the SOAP level (non-repudiation, integrity);

- At the HTTPS level (TLS session negotiation, eventually assuring authentication, integrity and privacy).

PKI is a proven concept and already widely adopted in browser-server interaction on the WWW. PKI can thus facilitate and enable transport level security in Web Services. Furthermore, because of its propitious scaling properties, PKI seems to be an indispensable component if Web Services multiply and gain momentum.

Web Service Security Standards

A key element to obtaining widespread Web Service interoperability is that implementers have good and universally adopted standards for Web Service security. Standardisation work is being conducted by a number of organisations and industry actors. In addition, there are organisations like WS-I whose dedicated focus is on interoperability issues. This seems to indicate that there is a strong momentum in the industry to develop Web Service standards.

Attempts to standardise WS security come in many forms. Since SOAP security measures are likely to play an increasingly prominant role in ongoing development toward meeting the full range of requirements for Web services, many standardisation efforts revolve around defining SOAP security headers and formats.

World Wide Web Consortium (W3C) XML security

Work on the following key XML specifications within the World Wide Web Consortium (W3C) is setting the stage for SOAP security considerations. These standards are seen as effectively laying the groundwork for SOAP or other XML based messaging security over the Web:

- XML Digital Signature (XML-SIG) specifies the syntax and processing rules for applying digital signatures to any XML data.

- XML Encryption group is developing a process for encrypting/decrypting digital content (including all or parts of XML documents) and is creating an XML syntax to represent encrypted content and the information for decrypting it.

- XML Key Management Services (XKMS) specifies protocols for distributing and registering public keys so these can be used in conjunction with XML digital signatures and encryption.

- SOAP Security Submissions to the W3C. The SOAP-DSIG submission to the W3C is an industry effort to get XML digital-signature extensions incorporated directly into the SOAP specification. SOAP-DSIG specifies the syntax and processing rules for a SOAP header entry, called the SOAP-SEC element, to carry digital-signature information as part of the SOAP message envelope.

e-business XML (ebXML) Messaging Service

- ebXML messaging is based on the SOAP 1.1 specification but defines some additional SOAP headers to handle ebXML specific functions such as message routing.

- The initial ebXML messaging specification recommended XML-SIG and supported S/MIME for attachments.

- As the SOAP specification versions move through the standardization process, the ebXML messaging group is likely to continue addressing both forward and backward compatibility with SOAP

Organization for the Advancement of Structured Information Standards (OASIS)

OASIS is promoting other standards work relevant to XML-based messaging security through two current efforts:

- Extensible Access Control Markup Language (XACML), which is standardizing security access control using XML by defining an XML specification (core schema and name space) for expressing authorization rules over the Internet.

- Security Assertion Markup Language (SAML), which is defining a standard XML syntax for specifying authentication and authorization credentials that can be sent along with SOAP messages. SAML will specify additional SOAP headers to carry assertions, among many other requirements.

WS-Security

WS-Security is a joint effort by IBM, Microsoft and VeriSign to standardise Web Service Security. According to [2],WS-Security describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. These mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

WS-Security also provides a general-purpose mechanism for associating security tokens with messages. No specific type of security token is required by WS-Security. It is designed to be extensible (e.g. support multiple security token formats). For example, a client might provide proof of identity and proof that they have a particular business certification.

Additionally, WS-Security describes how to encode binary security tokens. Specifically, the specification describes how to encode X.509 certificates and Kerberos tickets as well as how to include opaque encrypted keys. It also includes extensibility mechanisms that can be used to further describe the characteristics of the credentials that are included with a message.

Web Services Interoperability Organization (WS-I)

The Web Services Interoperability Organization is an open industry effort chartered to promote Web Services interoperability across platforms, applications, and programming languages. The main purpose of WS-I is to respond to customer needs by providing guidance, recommended practices, and supporting resources for developing interoperable Web services.

The organisation's deliverables are targeted at proving resources for any Web Services developer to create interoperable Web services, and verify that their results are compliant with both industry standards and WS-I recommended guidelines.

WS-I addresses these issues through the concept of 'profiles'. Profiles are named groups of Web Service specifications at specific version levels, along with conventions on how they work together. WS-I has so far defined only one profile, 'WS-I Basic Web Services'. This profile does not include any specific security aspects. WS-I will hopefully provide profiles addressing different security issues in the near future.

Existing Platforms

By studying state-of-the-art Web Services platforms, we have evaluated how the Web Service security requirements currently are met in practice.

Microsoft .NET

.NET is Microsoft's newest platform for developing and running applications, competing with the J2EE standard. It contains Web services functionality as an integrated part. Visual Studio is a development environment for developing applications. .NET integrates technologies that have been evolving during the last years, such as COM+, ASP, XML and Web services, based on protocols like SOAP, WSDL, UDDI and HTTP(S).

The .NET architecture handles security through Microsoft Internet Information Services (IIS) and is leveraged by ASP.NET. ASP.NET can take the identity information provided by IIS and use that to know who accesses the service or to make use of code access security for specific operations on the Web Service. HTTP-level security is the most common approach in making message transmissions secure. IIS provides

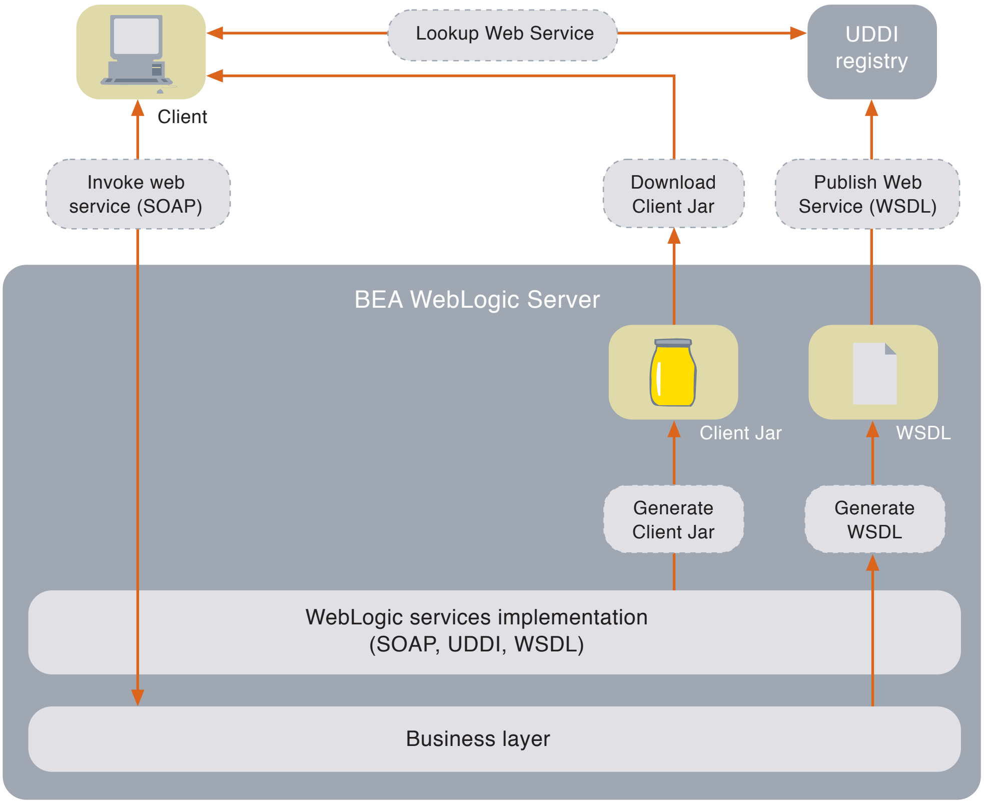


Figure 5 The WebLogic Server

SSL-support, which also includes the use of digital certificates.

To restrict access to the Web Services, client authentication is needed. IIS provides some such mechanisms, such as authentication by username and password, certificates and windows logons.

The .NET Framework provides several mechanisms for protecting resources and code from unauthorized code and users:

- ASP.NET Web Application Security provides a way to control access to a site by comparing authenticated credentials to Microsoft Windows NT file system permissions or to an XML file that lists authorized users, authorized roles, or authorized HTTP verbs.
- Code access security uses permissions to control the access code to protected resources and operations.

Role-based security provides information needed to make decisions about what a user is allowed to do.

Microsoft is currently involved in WS-Security, which is a joint effort by IBM, Microsoft and VeriSign to standardise Web Service Security (see Web Service Security Standards). A preview of the *WS-Security development toolkit* is available at [3].

BEA WebLogic Server 7.0 with Workshop

WebLogic Server (WLS) 7.0 is a fully J2EE-compliant application server, regarded as one of the leaders in the application server space because it has many Java-oriented features and is a reliable server. According to the documentation, it supports SOAP 1.2, WSDL 1.1 and UDDI 2.0 and integrates the developing environment with web service support.

WebLogic uses its own declarative and programmatic security models.

The *declarative security model* is an implementation of the J2EE servlet security model, which enables access restriction to any resource deployed on the server (i.e. a Web application, a Web service or a Web service method). Users are assigned to groups, and different groups are granted access to different Web resources.

The servlet security model also allows offering the Web Services within a given application on HTTP- or HTTPS-enabled ports as needed. HTTPS should be used whenever web service communication involves the transmission of sensitive data.

The *programmatic security model* allows making security decisions in the Web Service code itself to change how Web services behave based on the permissions of the user. Using this model

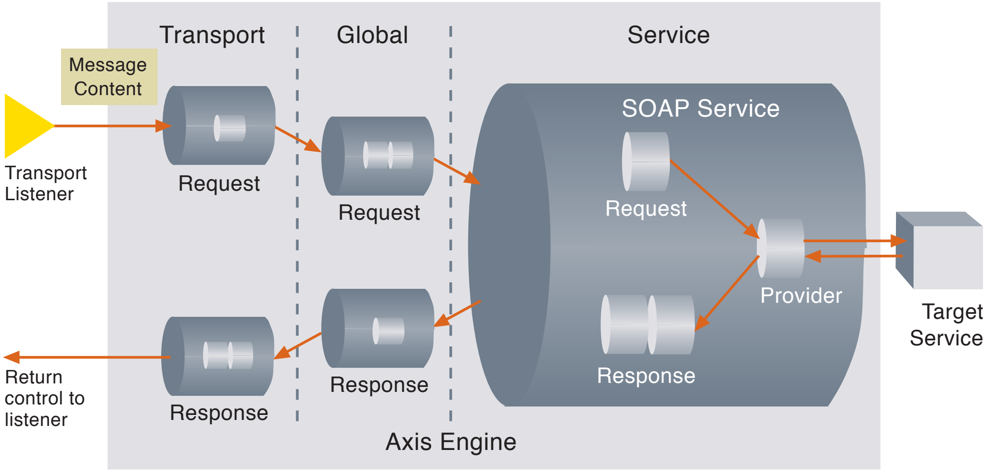


Figure 6 The Apache Axis Web Service Engine

it is possible to authorise clients within the Web service code and also send credentials with outgoing messages.

Apache Axis Web Service Engine

Axis is the open-source, freeware Web services container offered by the Apache project. The acronym stands for Apache Extensible Interaction System. It is a complete reimplementation of the architecture of the Apache SOAP project.

Axis is mainly a SOAP engine that acts as a client and server of SOAP messages with primary focus on HTTP. It can be viewed as a thin layer sitting between the business logic and the network transport. It includes a stand-alone server, but is most commonly used as a Web application on top of a Tomcat Web Server (also an open source, freeware implementation of the Apache project), or it can run as a Web application on top of an application server (like BEA).

Apache Axis has support for WDSL 1.1, limited support for SOAP 1.1, but no documented support for UDDI. Axis is only the Web Services engine, and as such has no need of explicitly supporting HTTPS, which is the responsibility of the Web server it is deployed on. When combined with a Web server that supports TLS (most typically Tomcat), one could say that Axis has ‘implicit’ support for HTTPS.

The XML Security package is made available under the Apache Software License and can be downloaded from Apache. It supports the XML-Signature Syntax and Processing recommendation. But except for this enabling of signatures and HTTP basic authentication, there are no specific security features in Axis. SOAP messages can be sent over HTTPS, but other than that there are no explicit transport security mechanisms. Developers have to implement the larger part of the security on the application level if such is desired.

Axis has defined some preliminary security extensions, which can integrate with the Servlet 2.2 security and role specification.

Conclusion

It is reassuring that Web Service security technologies are based upon well-established and thoroughly tested WWW technologies. Web servers can be enhanced with authentication and encryption mechanisms like HTTPS to keep certain parts of client-server message exchange confidential. HTTPS has been important for enabling e-business on the WWW, and can be counted on to provide a certain level of runtime security. Web Services span across firewalls and network boundaries and unlike CORBA has the advantage of avoiding practical problems with firewall configuration and deployment.

A Web Service *security policy* should address basic issues such as identification and authentication, authorization, auditing, the need for data integrity and confidentiality, non-repudiation and privacy.

To meet the security requirements set out in the security policy, a range of technologies and products are available. It is fairly simple to protect databases and registries. Protecting communications is somewhat more challenging, and a major industry effort is currently under way to provide the messaging protocol SOAP with security mechanisms to complement HTTPS. For the time being however, SOAP is a fairly new and untested technology that should be handled with care until it features universal security standards.

PKI is a proven concept and already widely adopted in browser-server interaction on the WWW. Integration of a PKI key management system in Web Services may provide the Web Service Providers and users with a backbone upon which technical security mechanisms can be built.

Many standardisation efforts are currently under way, but at this time no major and universal standard is available. The main question is: Will standardisation organisations and industry be able to agree on one single standard for Web Services security? In our opinion the success of Web Services depends on such a universal standard. It is worth stressing that Web Services, secured or not, should be able to interoperate. Without interoperability, the success of this emerging technology is doubtful. Thus it is imperative that organisations such as WS-I define guidelines and standards for interoperability that also entail security.

Current Web Services platforms support some basic security standards (HTTPS), but unfortunately the lack of universal standards has prevented platforms vendors from implementing standardised security solutions. This tendency toward using proprietary solutions may jeopardize the very important criterion of Web Service interoperability.

For the time being, the lack of standardisation makes it hard to unconditionally endorse Web Services security. Interoperability is the key issue and an important success criterion for this still emerging technology. Interoperable solutions may lead the Web Services concept toward wide acceptance in the business community. On the other hand, the lack of interoperability may lead Web Services toward its eventual demise.

References

1 Prescod, P. *Some thoughts about SOAP versus REST on Security*. November 15, 2002 [online] – URL: http://www.prescod.net/rest/security.html

2 Houston, L. *Best practices : Web security. SOAP Security Issues*. November 15, 2002 [online] – URL: http://dcb.sun.com/practices/websecurity/overviews/soap_security.jsp

3 *Using WS-Security with the Web Services Development Kit Technology Preview*. November 15, 2002 [online] – URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwssecur/html/wssecwithwsdk.asp

Further Reading

Van Do, T et al. *XML Web Services : The services of the future?* Fornebu, Telenor workshop, June 2002.

Van Do, T et al. *Telenor Mobile XML Web Services*. Fornebu, Telenor R&D, 2002. R&D Scientific Document N 29/2002.

Jakobsson, S et al. *PIR 2.1 – State-of-the-art XML Web Service Technologies and Standards*. Project Internal Result, EURESCOM Project 1209, 2002.

Peterson, L L, Davie, B S. *Computer networks – a systems approach*. San Francisco, 2000. (ISBN 1-55860-577-0)

Hartman, B et al. *Enterprise Security with EJB and CORBA*. OMG Press, USA, 2001. (ISBN 0-471-4031-5)

McKinsey Quarterly report. *Risk and resilience*. Special Edition, 2002.

Kaler, C (ed.). *IBM Web Services Security (WS-Security), Version 1.0 05 April 2002*. November 15, 2002 [online] – URL: http://www-106.ibm.com/developerworks/library/ws-secure/

Security in a Web Services World : A Proposed Architecture and Roadmap. (IBM white paper.) November 15, 2002 [online] – URL: http://www-106.ibm.com/ developerworks/webservices/library/ ws-secmap/?l=af

Web Services Interoperability Organization (WS-I). November 15, 2002 [online] – URL: http://www.ws-i.org/

Microsoft Web Services. November 15, 2002 [online] – URL: http://msdn.microsoft.com/webservices/default.asp

Apache Axis. November 15, 2002 [online] – URL: http://xml.apache.org/axis/index.html

WebLogic Workshop Documentation. November 15, 2002 [online] – URL: http://edocs.bea.com/workshop/docs70/index.html

Acronyms

API	Application Programming Interface
ASP	Active Server Pages
BEA	Basic programming Environment for interactive-graphical Applications, from Siemens-Nixdorf
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
ebXML	electronic business XML
FTP	File Transport Protocol
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Security
IETF	Internet Engineering Task Force
IIS	(Microsoft) Internet Information Services
IP	Internet Protocol
IPsec	Internet Protocol security
J2EE	Java 2 Enterprise Edition
OASIS	Organization for the Advancement of Structured Information Standards
PKI	Public Key Infrastructure
SAML	Security Assertion Markup Language
SASL	Simple Authentication and Security Layer
SET	Secure Electronic Transaction
S-HTTP	Secure HyperText Transfer Protocol
SMTP	Simple Mail Transport Protocol
SOAP	Simple Object Access Protocol
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDDI	Universal Discovery Description and Integration
UUP	Universal User Profile
W3C	World Wide Web Consortium
WLS	Web Logic Server
WSDL	Web Service Description Language
WS-I	Web Services Interoperability Organization
WWW	World Wide Web
XACML	eXtensible Access
XKMS	XML Key Management Service
XML	eXtensible Markup Language Control Markup Language
XML-SIG	XML Digital Signature

Evolving Service Creation; New Developments in Network Intelligence[*)]

JOHN-LUC BAKKER, DAVID TWEEDIE AND MUSA R. UNMEHOPA

The application and deployment concept of programmable network capabilities have been well understood and embraced in the industry. To date, Parlay and JAIN have focused on unlocking existing (e.g. IN-like) network capabilities to a Java and web-enabled developer community. More recently, additional efforts to further appeal to both application developers and service providers have emerged. The advanced abstraction and simplification in the programming interfaces (Parlay X), the interest in alternative transport and middleware technologies (Parlay Web Services), and the emerging field of telecom-oriented, XML-based scripting languages are clear indications of this development. Furthermore, we see an attempt to unify the Parlay concepts with another service mediation technology, SIP, to combine programmatic objects and data types with session control capabilities, as witnessed in the IETF SPIRITS initiative. In this paper, the authors will give a compendious overview of these new developments in the area of programmability, and the applicability thereof within today's and the next generation of networks.

John-Luc Bakker (30) holds an M.S. degree in programming aspects of distributed and parallel computing from the Delft University of Technology, The Netherlands, since 1996. Prior to his current position he worked with Lucent Technologies on component based service creation environments, advanced communication architectures, multimedia, and recent change code generation. He has published several papers in these areas. Bakker joined Telcordia in 2000 as research scientist and project manager in the Middleware and Mobile Applications Research Group. He is currently active in several research projects including participation in standardization fora such as JAIN, 3GPP and Parlay.

jbakker@telcordia.com

David Tweedie (30) has been working as software designer for Nortel Networks in Ottawa, Canada since 1998. He joined Nortel after receiving a Bachelor of Computer Science with a major in Information Systems from the University of New Brunswick in Fredericton, Canada. While at Nortel, David has primarily been working on service enabling technologies. He started out as a member of an Advanced Intelligent Networks development team. After that, he progressed to next generation third-party programmability solutions which eventually led to investigations into the Parlay/OSA APIs.

davidtw@nortelnetworks.com

1 Introduction

In present day communications networks, connectivity and high quality voice have become commodities. The supplementary services and A/IN based applications are increasingly becoming commonplace. We have come to realize that it is the value added services and applications that are the moneymakers, providing the greatest revenue-generating potential. In the struggle to secure a piece of application market share, all the players have embarked on a quest for the killer application. If history has taught us anything, however, it is that one simply cannot predict what the killer application will be. Time to market is yet another key factor in the multi-faceted equation. Killer applications are not tangible, sometimes subject to a disquieting amount of hype or fashion. Furthermore, the days of technology push have gone. Large service providers and telecommunication equipment vendors can no longer expect to introduce and rollout expensive dedicated network equipment and service platforms, supporting applications that are then forced upon the consumer community. Users of communications networks and services are educated and mature; there is no room for complacency. Hence, focus has shifted to finding and defining the killer environment, or the killer enabler, rather than the killer application. The killer environment will allow the network operator or service provider to quickly, easily, and without much disruption to ongoing network operations, introduce the killer application, or killer applications, once it is found.

It has generally been noticed in the industry that the killer enabler is provided by open access to, and programmability of, network service capabilities. There are two aspects to this killer environment. First, there is service mediation, i.e. providing application developers with a unified approach and consolidated architecture for open but secure, easy but regulated, flexible but scalable, access to core network service capabilities. Second, to increase the likelihood that a killer application will emerge, one needs to provide a fertile greenhouse for as large a developer community as possible. This greenhouse should not be limited to the traditional R&D labs; the larger developer community, including the enterprise application developers and web developers, need to be engaged and inspired to partake in the process of devising the successful value added services and applications of tomorrow. This is envisaged to be achieved by abstracting from the sheer complexity of the application development process, and the technologies and protocols involved. The killer environment needs to appeal to the large crowd of application developers, by defining the open interfaces using technologies and methodologies that are close and familiar to them.

Figure 1 shows a layered architecture designed for open access to service capabilities [3]. It consists of a resources layer, a services layer and application servers. The SCFs (Service Capability Features) are implemented in the services layer and provide their capabilities to the application servers. In general, application servers and the programmable gateway are physically separated entities. Therefore, a distribution technology between the SCFs in the gateway and the application server is needed. Figure 1 also indicates an interface between the resources layer and the services layer.

Several papers have addressed the issue of open access, service programmability using third generation programming languages and its service

[*)] *© The contents of this paper is under the copyright rules of Telcordia.*

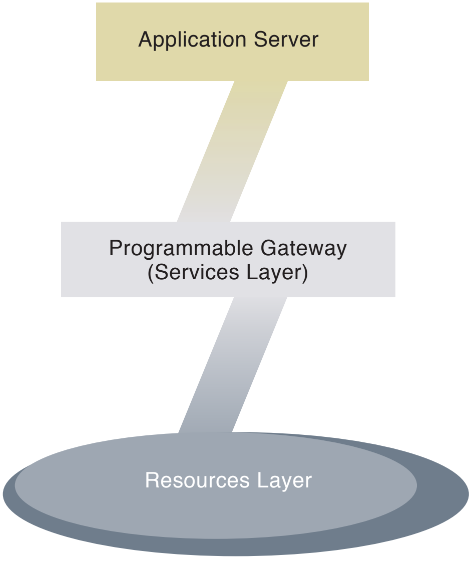


Figure 1 General Programmable Gateway Architecture

mediation aspect (see e.g. [16] and [23]). The focal point of this paper however, is the second aspect of a successful killer environment, i.e. the appeal to the application developer crowd. This is achieved by simple access, using the technologies they like, such as scripting technologies, at a higher level of abstraction. The most promising technology to fulfill these requirements, currently under development in the industry, is that of Web Services. A Web Service is a capability that can be invoked via standard Internet protocols. As such, the prospect of an SCF (Service Capability Features), accessible via SOAP/XML ([25] and [24]), would perfectly fit the Web Services paradigm.

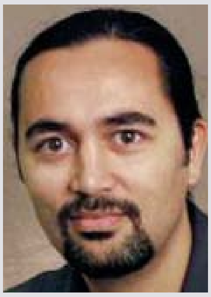
This paper will specifically address the aspect of engaging and involving the larger developer community. It is important to note that the work on service creation in next generation networks is in full progress. The approach that the authors adhere to, for the purpose of this paper, is to classify application programmability and next generation service creation into three main categories. These categories are generic Application Programming Interfaces (APIs), using many distribution technology realizations, comprehensive access to network capabilities using scripting, and simple and network specific interfaces deployed as Web Services (Parlay X). It is not the intention to provide an exhaustive survey; rather the authors will provide a concise synopsis by presenting one or more compelling and promising representative technologies out of each category. In addition, the paper will address the relation of Parlay with another service mediation technology proving to be of immense interest to the industry, i.e. the Session Initiation Protocol (SIP) [15]. The paper will conclude with a correlation of all these technologies and initiatives in a comprehensive overview, identifying how they link up into the global service architecture.

2 Programmability Through Programming Languages APIs

One of the first methodologies or design paradigms adopted by the telecommunications world from the information technology domain is the use of APIs. An API is a programmatic interface providing access to or programmability of software resources, such as database applications or telecommunication protocol stacks. An API provides application developers with programmability of software resources, by defining these resources in terms of objects and methods, data types and parameters that operate on those objects. Examples of resource layer APIs include the JAIN protocol APIs, such as JAIN SIP [17] or JAIN INAP [18]. At a higher abstraction layer we find the service layer or network capabilities APIs, for example the Parlay APIs [12] or JAIN JCC (Java Call Control) [20] and JCAT (JAIN Coordination and Transactions) [21]. The service layer APIs do not focus on individual resources, but rather provide application developers with access to service capabilities residing in a core telecommunications network. In the remaining we will focus on the Parlay APIs, their UML representation and the Parlay realizations.

Parlay APIs are independent of the actual resources; however, protocol mapping recommendations exist. Actual resources may reside either in the A/IN network, in Mobile networks, Managed IP networks, or in Next Generation Networks. Parlay Application is thus abstracted from resource implementation. As a consequence, the Parlay APIs expose only common aspects and generic functionality of communication networks.

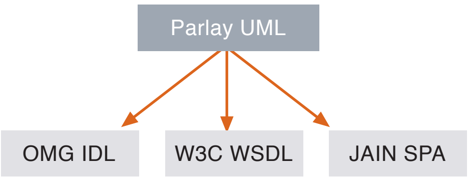
The Parlay APIs are defined using UML (Unified Modeling Language) [12]. The UML modeling technology is realization technology independent. However, as the Parlay UML was reengineered from earlier Parlay OMG (Object Management Group) IDL (Interface Definition Language) [7] definitions, the Parlay UML is not fully technology independent. Distribution technologies other than OMG's CORBA may realize particular distribution aspects differently. This leads to mismatches when mapping the UML to other realization technologies. Also, the Parlay APIs specify supports for distribution aspects, such as authentication, on the application level even while they are essentially orthogonal to the problem domain of telecommunications. A full discussion of the Parlay UML and its deficiencies is not the scope of this paper.



Musa R. Unmehopa (31) received his M.Sc. in computer science in 1996 from the University of Twente, The Netherlands, after which he joined Lucent Technologies, Bell Laboratories. He is currently a senior standards consultant engineer in the Wireless Advanced Technologies Lab within Lucent Technologies, The Netherlands. He is actively involved in the standardization of open network Application Programming Interfaces within 3GPP, 3GPP2, ETSI, the Parlay consortium, and more recently, the Open Mobile Alliance (OMA). Currently, Mr. Unmehopa holds the position of vice-chairman of 3GPP Technical Specification Group (TSG) Core Network Group 5 (Open Service Access).

unmehopa@lucent.com

Figure 2 Parlay Technology Realizations

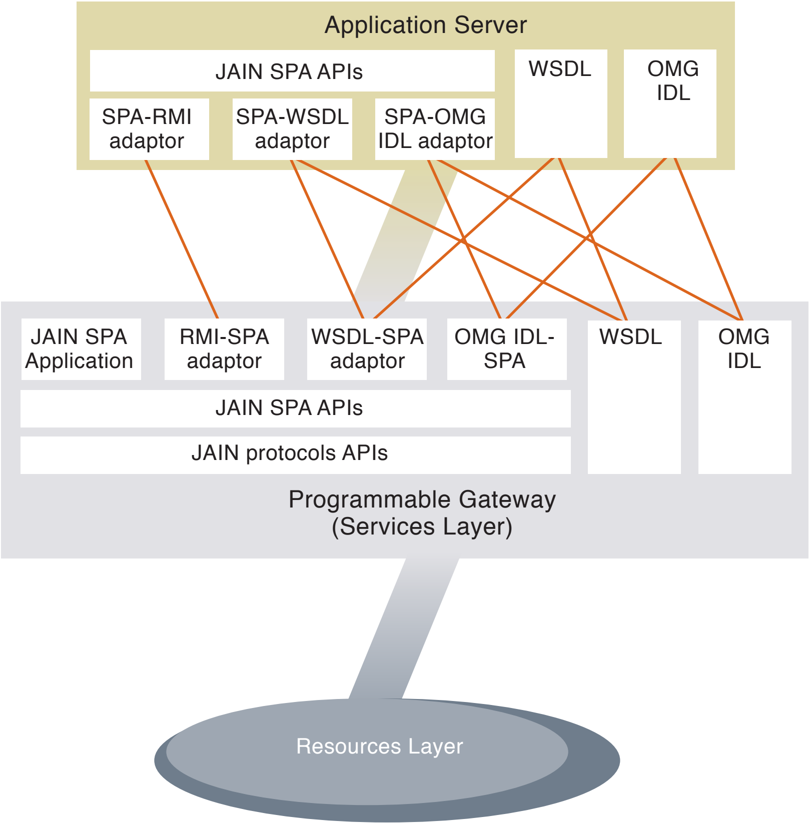


Still, with some effort, a number of technology specific realizations can be generated from the Parlay UML model. Initially OMG IDL was the only generated and published realization. Recently, a W3C WSDL [27] realization has been generated and published along with the rules that were established to enable automatic generation of future versions of the Parlay specification. In the immediate future a rulebook that specifies the mapping between the Parlay UML and the JAIN SPA (Service Provider APIs) will be made available. Figure 2 shows the relation between the Parlay UML, OMG's IDL, W3C WSDL, and JAIN SPA. The realization technologies will be further discussed in the remainder.

2.1 OMG IDL

The technology realization which was initially used with Parlay APIs was the OMG IDL. OMG IDL is used to specify the interfaces of remote objects. OMG IDL interface implementations (so-called remote objects) typically execute within a different process on a different host with respect to the implementation's client. As such, OMG IDL is well suited for defining the interface of the Parlay APIs. Additionally, OMG IDL is the definition language used to define a set of interfaces for use with the Common Object Request Broker Architecture (CORBA) [13]. CORBA is a standard, defined by the OMG, that supports distributed objects. CORBA provides a complete set of services to remote objects, including concurrency control, licensing, life-cycle management, security management, and persistence. Finally, another benefit of using OMG IDL as a specification language is that it is programming language independent; the OMG has provided standard mappings to many programming languages, including Java and C++.

Figure 3 Realization architecture



2.2 Java

It is recognized that Java is a firmly established programming language in the enterprise application market, and emerging in other fields. For that reason, many middleware systems support language mappings to Java. The Java software development packages come with a vast number of libraries that implement telecom-related (i.e. JAIN) and other standards. The JAIN SPA working group felt that the Java communities experience in implementing as well as certifying conformance to standards could be beneficial to the Parlay community. Note, however, that the Java community does not focus on the distribution technique, but more on ease of use of the APIs and whether they follow the patterns and paradigms expected of good citizens of the population of Java API. It was felt that the OMG IDL mapping to Java inadequately conformed to these goals. Additionally, some CORBA ORB vendors inadequately implemented the (latest) OMG standards that map Java to IDL, thereby inhibiting the portability of Java applications.

To circumvent these hurdles and increase acceptance of the Parlay APIs within the large community of Java developers the Parlay group is producing a rulebook which specifies the mappings between the Parlay UML model and Java. The Java realization activities underway within the Parlay group are part of SUN's JAIN SPA initiative.

As indicated in Figure 3, the JAIN SPA APIs are slightly different from the other technology realizations in that they specify a local API as opposed to a distribution technology API (such as IDL or WSDL). These JAIN SPA APIs can reside on either the Parlay application server or on the Parlay gateway. The JAIN SPA APIs are independent of the distribution technology used. It is up to the actual implementation of the JAIN SPA APIs to determine which type of distribution mechanism it wishes to use (i.e. IIOP, SOAP, RMI, etc). Figure 3 illustrates how the JAIN SPA APIs can be used to provide a layered approach to Parlay application and gateway servers.

2.3 WSDL

The XML-based Web Services paradigm transforms the Web into an anthology of independent application components, each with a well-defined and published interface, which allow other Web applications to find them and use them. Web Services are built on top of existing, inexpensive and easy-to-encrypt Web protocols such as HTTP and based on open XML standards for data encoding. Web Services merely draw upon the omnipresent Internet infrastructure to discover and compile services into compelling value added applications.

The Parlay APIs are realized in WSDL (Web Services Description Language) [27]; WSDL is an XML format definition language which is used to describe the programmable interface for a service. This usage is similar to the OMG IDL use discussed earlier. WSDL is specified by the W3C organization and is defined with XML and XML Schemas [28]. The first transport supported by WSDL is SOAP [25] over HTTP. Note that other transport bindings may be supported while retaining the WSDL interface definitions. A WSDL document is defined by a series of XML Schema elements.

Within Parlay, a set of mapping rules from the Parlay UML model to WSDL has been written. These mapping rules provide a mapping from UML constructs into WSDL elements. The detailed mapping from UML to WSDL can be found in the Parlay Overview specification [14]. The strength of this approach lies in the strong association with web protocols and associated service creation and deployment technologies. Hence, the WSDL mapping is done to further appeal to the developer community.

Figure 3 shows the possible interplay of all Parlay realization technologies. The figure illustrates how the various Parlay realization technologies could be utilized within a Parlay solution. As illustrated, CORBA, SOAP, or even RMI plays a similar role as a distribution mechanism within a Parlay solution. The JAIN SPA APIs can provide a further abstraction from the distribution layer both on the Parlay Application Server and on the Parlay Gateway. This section has introduced the objective of the Parlay APIs to expose only common aspects and generic functionality of communication networks, in order to allow for application portability across network technologies. To be able to deploy these applications using this common and generic API in the various network technologies, Parlay defines several distribution technology realizations.

3 Simplified Programmability Through Web Services

The Parlay APIs expose capabilities of the telecommunications network in a network technology and programming language neutral way. In fact, the contributors to the APIs go to great lengths to ensure that the common aspects of (converged) mobile, fixed, and managed packet networks are made programmable for a large audience of developers of carrier grade systems. Yet, some more specific but highly valuable capabilities remain unsupported. As an example, the support for SMS (Short Messaging Service) is inadequate. Additionally, the developer who wishes to program simple applications, such as setting up a third party call (a.k.a. click to dial), using the Parlay APIs, needs to go through elaborate interactions with different components and interfaces.

These observations motivated the need for APIs that are predominantly simple and, consequently, restricted in their capabilities; developers that need access to advanced means of control would not be users of these simple APIs, but rather use the existing Parlay APIs. Additionally, the rigid dogma that favors exposure of common network capabilities over specific capabilities needed to be relaxed. Finally, the resulting APIs would predominantly be used in a multi portal or a web environment. The Parlay community proved to be open to these views and approved the establishment of a group, the Parlay X group, in late 2001, which was chartered to create APIs that incorporated the above views. It was felt that new markets are made available through Web Services with Parlay X application definitions. Given a set of high level interfaces that are oriented towards the skill levels and telecom knowledge levels of web developers, the Parlay X APIs open the accessibility of the network capabilities to a much wider audience.

Figure 4 shows where the Parlay X applications are situated with respect to the Parlay X gateway. Note that many of the Parlay X capabilities will be mediated by the Parlay Gateway. Some, however, are not supported because of Parlay's focus on common capabilities as opposed to network specific capabilities. Such capabilities cannot be mediated through the Parlay Gateway, rather they need to be made available through the resources layer.

Consider a web server that allows end users to charge for services they consume to their prepaid accounts, or a customer support page that creates, by the press of a button, a voice call between an end user and a customer service representative. Developers in this environment often use web services technology to communicate with different capability servers, i.e. they

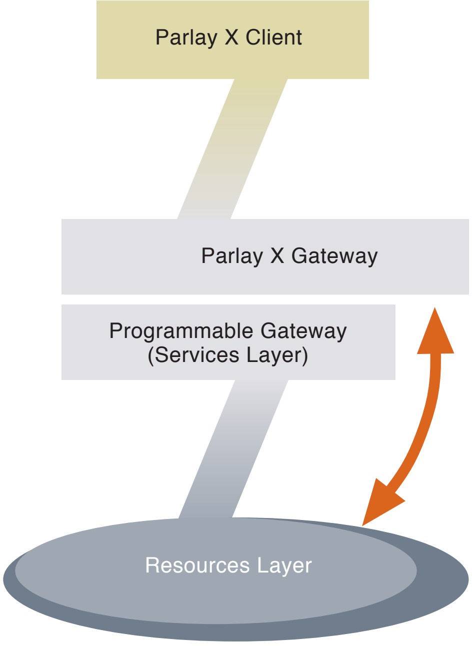


Figure 4 Parlay X Gateway accessing specific network capabilities directly

use SOAP and WSDL. Web services provide a set of capabilities and technologies that result in a very compelling foundation for supporting converged telecom/IT applications.

As with the Parlay APIs discussed in the previous section, authorization, discovery, extensibility and activation are very important features that need to be adequately addressed. The Parlay APIs introduce the set of Framework APIs to address these issues on an application level. The web services environment is different since several mature technologies exist today or are the industry norm and can be leveraged to achieve the same. For example, UDDI (Universal Description, Discovery and Integration) or WSIL (Web Services Inspection Language, a.k.a. WS-Inspection) can be used to discover or retrieve references to specific services. WSIL files describe web services, possibly in a hierarchical manner, while UDDI serves a centralized registration and service publication solution. UDDI or WSIL allow for Parlay X applications to discover published services. The UDDI or WSIL-driven registry, finally, contains information using which the Parlay X application can bind and activate the Parlay X service. Note that authorized personnel can extend the registry with more Parlay X services.

Authentication required prior to accessing the private registry can be achieved through acquiring access using a VPN or through applying HTTPS. Many VPN solutions exist, supporting a variety of authentication methods. Based on the authentication information (e.g. user handle and password in the case of HTTPS) access to only the subscribed services can be enforced through dynamically generated WSIL files; effectively authenticating access to subscribed services only. Finally, various accounting schemes can be employed based on the invoked service, time of authentication, service agreement established a priori, etc.

Today, Parlay X participating companies have submitted contributions that target Parlay X compatible definitions of web services for UI (User Interaction), 3PCC (Third Party Call Control), Payment, TopUp, UserStatus/Presence/Location, and Messaging (i.e. SMS). 3PCC and UI web services are motivated by the observation that applications that interface with telecommunications resources often initiate and receive voice calls. 3PCC supports initiation of voice calls (e.g. click to dial) and recognizes user input (i.e. voice or DTMF). The TopUp API allows consumers to increase the value of their prepaid accounts and the payment API allows content providers to charge for certain types of content such that the billing is handled by the operator. Examples of content that can be charged for are downloadable ring tones or downloadable voice mail announcements. Next, the UserStatus/Presence/Location contribution focuses on presenting User Status or Presence (such as online, offline, or engaged) and Location (fine grained as longitude and latitude or as coarse as within or not within an area). Finally, the Messaging API is intended for sending messages (most notably, SMSes) from web pages to devices that can accept such messages.

It is hard to measure simplicity. Parlay X focuses on exposing capabilities through accepted and largely applied technologies by the IT industry. Furthermore, the IT industry is currently furthering the Web Services architecture; emerging standards for transactions and integrated security will make the use of Web Services as a middleware solution even more attractive and will further reduce the complexity of creating telecommunications applications. Already, Parlay X is exploring the applicability of Web Services middleware in constructing a programmable gateway. As outlined above, a number of contributions have been suggested and are currently being consolidated and processed for inclusion in the first release of the Parlay X APIs.

4 Programmability Through Scripting

Scripting languages are lightweight, highly customizable, and typically interpreted languages, appropriate in the area of rapid application development, acting as glue to provide connections among existing components. These characteristics allow them to be used to code or modify applications at runtime, and interact with running programs. Such qualities and features make scripting languages suitable application enabler abstractions and highly applicable to the field of application programmability next to APIs.

XML is commonly seen as the preferred vehicle to create service creation languages. Aside from its standardization and readability by both machines and humans, XML offers several additional benefits. XML supports restrictions to its expressiveness. Such a restriction enables easy validation and determinability. In general, XML schemas allow the design of languages that can be non-expressively complete, thereby guaranteeing that the XML interpreter, while using a limited amount of time and resources, can easily execute the script. Finally, many tools and libraries are available to create, interpret and validate XML documents.

The next generation of scripting languages for creating value-added services in converged networks will be based upon XML. Industry fora like Parlay and JAIN have developed open standard APIs to enable service creation in today's and tomorrow's networks. While services can be developed in traditional (third generation) programming languages (e.g. Java or C++) using these APIs, the XML-based scripting languages offer some attractive advantages. While not as flexible or powerful as a programming language, scripting languages are typically easier to learn, and are platform independent.

Several XML-based call control markup languages have been previously proposed, including CPML, TML, XTML, CallXML [11], CPL ([9] and [10]), the Call Control XML (CCXML) [20], and Service Creation Markup Languages (SCML) [17]. A comprehensive survey of these call control markup languages proposals is not the purpose or within the scope of this section. Instead, we observe the SCML developed by the Service Creation Environment Expert Group of the JAIN industry forum, and developed according to open processes by a number of actively participating companies. The rigorous process and the openness, along with SCML's close relation to Parlay and JAIN will contribute to an increased industry acceptance and therefore warrant a closer look.

Languages such as SCML, CCXML and CPL are used to create applications that make use of the functions provided by the services layer (as opposed to the resources layer), see Figure 5. The figure shows a scripting interpreter either at the application side of the programmability architecture or at the gateway side. This means that both the third party and the operator can support programmability through scripting. Scripts can be stored such that they can be retrieved from storage indicated by a URL or in subscriber databases like HLRs.

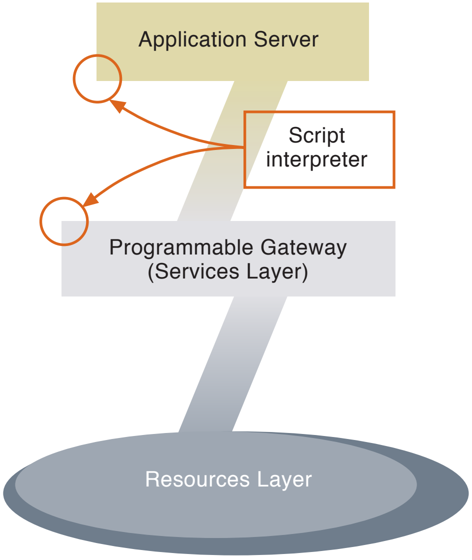


Figure 5 Script interpreters either integrated with the SCF or with the application server

The remainder of this section will further discuss SCML and its features. Note that a comparison between CPL, CCXML and SCML was presented in [1]. Note also that CPL, CCXML and SCML are all in the *work in progress* state and could change in the course of further standardization. However, we expect that the overall concepts will continue to apply. Finally, the examples given in this section are for illustrative purposes only.

4.1 SCML

The Service Creation Markup Language (SCML) is a scripting language that connects existing components such as JAIN SPA or Parlay APIs, enabling rapid prototyping, rapid application development, or easy end-user customization. We briefly discuss SCML here and compare it to CCXML.

SCML scripts are created, edited, and validated using regular editors or as a result of applying transformation techniques. Next, the scripts are deployed in a retrievable location (e.g. identified by a URL). SCML scripts follow a pattern of registering the static events and criteria that can be matched by events (e.g. those emitted by SCFs), followed by declaration of business logic to be executed in response to such an event. Therefore, activation initially encompasses provisioning the events and criteria and, subsequently, executing the business logic upon occurrence of the provisioned event. A script is deactivated through removing the provisioned criteria. The SCML specification only specifies the scripting language; no APIs are specified for activation, deactivation, and other service life-cycle events.

Figure 6 Example SCML Script: Call Forwarding on Busy

```
</scml>
  <register>
    <disconnected causeCode ="CAUSE_BUSY" destination ="sip:jdoe@home.com"
      connection="terminating" block="true">
      <routeCall targetAddress ="tel:2125552121"
        redirectingAddress ="sip:jdoe@home.com"/>
    </disconnected>
  </register>
</scml>
```

SCML is intentionally extendible. It consists of a common core, defined in the scml-core package that should be extended per JAIN SPA or Parlay component. Currently, the core is extended with the jccml package (JAIN Java Call Control Markup (JCC) Language); the jccml package assumes a JCC 1.1 API [20] implementation. The jccml package is defined using an XML Schema that is derived from JAIN's JCC API. JCC provides an API to pure call control related capabilities and can support traditional A/IN services as well as NGN services such as Click-to-Dial. Finally, JCC can be mapped on top of SIP [8], MGCP, and H.323 [8]. No public documents showing an informative mapping from JCC to ISUP, INAP and CAP may be available, but the JCC call model was designed to be protocol agnostic.

An example script in SCML is shown in Figure 6 for Call Forwarding on Busy. In this script the activation criteria (indicated by the <disconnected>-element) are registered with the gateway. The criteria specified by the registration element are the condition that call setup fails due to a busy callee, that callee's address, the fact that it concerns the terminating portion of the call, and an indication that the call processing must be suspended while the script executes. If these criteria apply, the scripts will be executed and the call will be redirected through specifying an alternative target address in the <routeCall> element. In this case, after forwarding the call, processing, which was suspended, will automatically be resumed.

Note that SCML is not limited to support for the JCC 1.1 API only. Further extensions (e.g. adding Short Message Service (SMS) components) are intended. In fact, Figure 7 shows a script that exemplifies support for and usage of the <sendSMS> element. The <sendSMS> element causes an SMS to be sent to the RFC2806 compliant URI tel:2125556767 only if destination tel:2125551212 is busy. The differences with Figure 6 are subtle; the example in Figure 7 validates the document against a Schema that extends the Schema used in Figure 6 such that it accepts the element <sendSMS>. Additionally, processing of the call processing machinery is not blocked (as the attribute block of the <disconnected> is set to false); since handling of busy connection and sending the SMS are parallel activities.

Concluding this section we state that there are many XML-based scripting languages. We have further discussed SCML and found that SCML is promising as an application enabled abstraction as it makes use of the capabilities provided by the various SCFs. SCML is from the ground up designed to be protocol agnostic and extendible to other sources of events.

5 Service Mediation Through SIP

JAIN and Parlay have introduced the concept of third party access to service capabilities residing in the core network. Apart from Parlay, as described in detail above, there is another popular service mediation technology that is receiving a lot of interest in the industry, i.e. SIP. This section discusses two recent standardization activities that incorporate the synergy of Parlay and SIP. The two activities are SPIRITS/PINT, taking place in the IETF, and OSA-to-ISC[1)],

Figure 7 Example SCML Scripts: SMS on Busy Subscriber

```
<scml>
  <register>
    <disconnected causeCode="CAUSE_BUSY" destination="tel:2125551212"
        connection="terminating" block="false">
      <getOriginatingAddress address="orig"/>
      <sendSMS to="tel:2125556767"
        content="'Call attempt by '+%orig;+' on 2125551212'"/>
    </disconnected>
  </register>
</scml>
```

taking place in 3GPP. In a nutshell, SPIRITS/PINT take conventional A/IN/CAMEL triggers/events from PSTN/ISDN networks and make those available in the Internet domain. OSA-to-ISC allows application developers to access the emerging 3G IP Multimedia Subsystem. Hence both activities target different underlying core network technologies, i.e. PSTN/ISDN, and 3G IP Multimedia. In the remainder we will elaborate on these two examples.

5.1 SPIRITS and PINT

Within the Internet Engineering Task Force (IETF) efforts have been ongoing for a while to define interworking between the traditional telephony networks and the Internet. PINT (PSTN/Internet Interworking Protocol) [5] addresses the requirement to invoke telephony services from the Internet, whereas SPIRITS [6] focuses on carrying A/IN triggers and events from the telephony network to the Internet. Combined, PINT and SPIRITS allow for A/IN-type service logic to be executed on IP hosts and to be delivered to traditional telephony subscribers. The remainder of this section will focus on SPIRITS.

The approach in SPIRITS has been to select those A/IN parameters of specific interest to application developers. The selection of relevant A/IN parameters has been based on the Parlay specifications. Various considerations on automatically generating these XML parameters and the use of XML Schema can be found on [6]. Summarized, a process collocated on the A/IN SCP (called the SPIRITS Client) monitors for A/IN triggers and events of interest from the telephony network. Once received, the SPIRITS Client extracts the relevant A/IN parameters, according to the Parlay definitions, and parses them into XML. The XML-encoded parameters and data types are then placed in the payload of a SIP message. This message is transported to the SPIRITS Gateway in the Internet domain. The SPIRITS Gateway either passes this message on to the SPIRITS Server, located on an IP host, which terminates the telephony request and is responsible for executing the A/IN-type service logic. Or, as depicted in Figure 8, the SPIRITS Gateway parses the XML-encoded parameters and hands them to the appropriate SCF. The SCF would then use one of the Parlay realization technologies to contact the application server. For a more detailed description of the SPIRITS architecture the reader is also referred to [6]. This architecture and protocol definition allows the SPIRITS Server to execute a Parlay application, based on the XML encoded Parlay representation of the relevant A/IN parameters.

The SPIRITS protocol, between the SPIRITS Client and the SPIRITS Gateway, uses the SIP SUBSCRIBE and NOTIFY messages, as these are specifically suited for trigger and event reporting type functionality. The work on the SPIRITS protocol is currently in progress. Figure 9 depicts an example where the A/IN "Answer" event is represented in terms of Parlay parameters, and carried as an XML encoded body inside a SIP NOTIFY message. The SIP header portion of this message is simplified for the sake of brevity; again, the reader is referred to [6] for more detail.

5.2 OSA to ISC mapping

SIP has been adopted by 3GPP [1] as the control protocol for the wireless next generation core network. This core network is referred to as the IP Multimedia Core Network Subsystem (IM CN). In a nutshell, session setup, control, and teardown functionality are performed by SIP servers, referred to as CSCF (Call Session Control Functions). The CSCFs do not perform any service logic execution. Instead, these tasks are performed by application servers. One such application server, acting as a SIP application server, is the Parlay Gateway. The protocol defined between the CSCF and the application servers is the IM CN Service Control protocol (ISC). With the CSCF and application servers being SIP servers, the ISC protocol is in fact the SIP protocol. The IM CN and its functional enti-

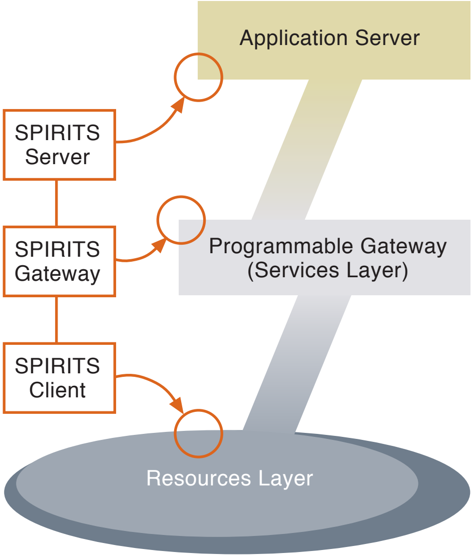


Figure 8 SPIRITS architecture in conjunction with programmability architecture

[1] *In this paper, OSA (Open Services Access) and Parlay are used interchangeably. Technically this is not correct, as the set of Parlay APIs is slightly larger than the set of OSA APIs. However, for the purpose of this paper this distinction is not essential.*

```
NOTIFY sip:jones@iphost.home.com SIP/2.0
From: <sip:directory_nr@ptt.com>;tag=SPIRITS-ER_RES-direc-
      tory_nr
To: <sip:jones@home.com>
Via: SIP/2.0/UDP gateway.ptt.com
...
<spirits-event>
  <DP Parlay=ER_RES/>
  <EVENT_REPORT_RESULT ver=1.0>
    <CALL_EVENT_TYPE>
      <P_CALL_EVENT_ANSWER />
    </CALL_EVENT_TYPE>
    <CALL_MONITOR_MODE>
      <P_CALL_MONITOR_MODE_NOTIFY />
    </CALL_MONITOR_MODE>
    <CALL_EVENT_TIME>
      1998-12-04 10:30
    </CALL_EVENT_TIME>
</spirits-event>
```

Figure 9 Example SPIRITS NOTIFY message for the Answer event

```
TpCallLegIdentifier createAndRouteCallLegReq (
   in TpSessionID callSessionID,
   in TpCallEventRequestSet eventsRequested,
   in TpAddress targetAddress,
   in TpAddress originatingAddress,
   in TpCallAppInfoSet appInfo,
   in IpAppCallLeg appLegInterface
   )
```

Figure 10 OSA API method

```
INVITE sip:targetAddress SIP/2.0
To: Bob <sip:targetAddress>
From: Alice <sip:originatingAddress>
Call-ID: callSessionID
CSeq: . . .
Alert-Info: <http://www.example.com/appInfo.CallAppAlerting-
Mechanism.wav>
```

Figure 11 Corresponding ISC interface operation

ties, as well as the ISC protocol, are being specified by 3GPP [4].

As part of their specification set, 3GPP publishes API to protocol mapping recommendations for various underlying network protocols. For OSA to be deployable in IM CN networks, the 3GPP specification in [1] recommends mappings of OSA to the ISC protocol. The OSA-to-ISC mappings cover the entire scope of the SIP protocol (i.e. beyond just SUBSCRIBE/NOTIFY) and its supported headers in order to provide support for the full breadth and depth of the Parlay Call Control APIs.

Figure 10 and Figure 11 show an example of how an OSA API method maps to an ISC interface operation. The first part of the figure shows the IDL definition of the createAndRouteCallLeg method, which is used by an application to request the creation and routing of a new call leg. The second part of the figure shows the SIP INVITE message onto which the createAndRouteCallLeg method is mapped. In bold font it is indicated how the individual OSA method parameters and data types map onto the SIP headers and their contents. The appInfo OSA parameter is defined as a union data type (`TpCallAppInfoSet`) of which the alerting mechanism (`CallAppAlertingMechanism`) is one of the possible fields. In this example, the alerting mechanism is mapped onto the Alert-Info header. For the benefit of simplicity, in this example the application does not request for the arming of triggers with the routing of this call leg.

The bold font indicates how the OSA parameters are mapped onto their counterparts in the ISC operation.

When comparing SPIRITS with OSA to ISC, the different approaches are quite apparent. In the SPIRITS approach the Parlay data is carried as an XML body inside a SIP message, whereas in the OSA-to-ISC approach the Parlay data is mapped onto the SIP equivalent data (headers).

Concluding, SIP is an Internet protocol that operates well with Internet related technologies like HTTP and MIME. So the combination of Parlay with SIP allows third party telecom application development at Internet speed, tapping into Internet creativity.

6 Conclusion

In this paper the authors emphasize the thesis that the key success factor of a thriving killer environment consists of the potency to appeal to a large application developer community through the flexible programmability of service capabilities. The authors present an extensive survey of established and emerging service programmability technologies. Three main categories are put forward, i.e. programmability through the use of Application Programming Interfaces, programmability through Web Services, and programmability through scripting. Subsequently, the authors impart their view on the relationship of Parlay with another service mediation technology, SIP, by showing they are complementary, and synergy can be achieved. A recurring element throughout all these discussions is the use of XML. The authors then make an effort to compile and amass all the concepts and discussions and provide an all-encompass-

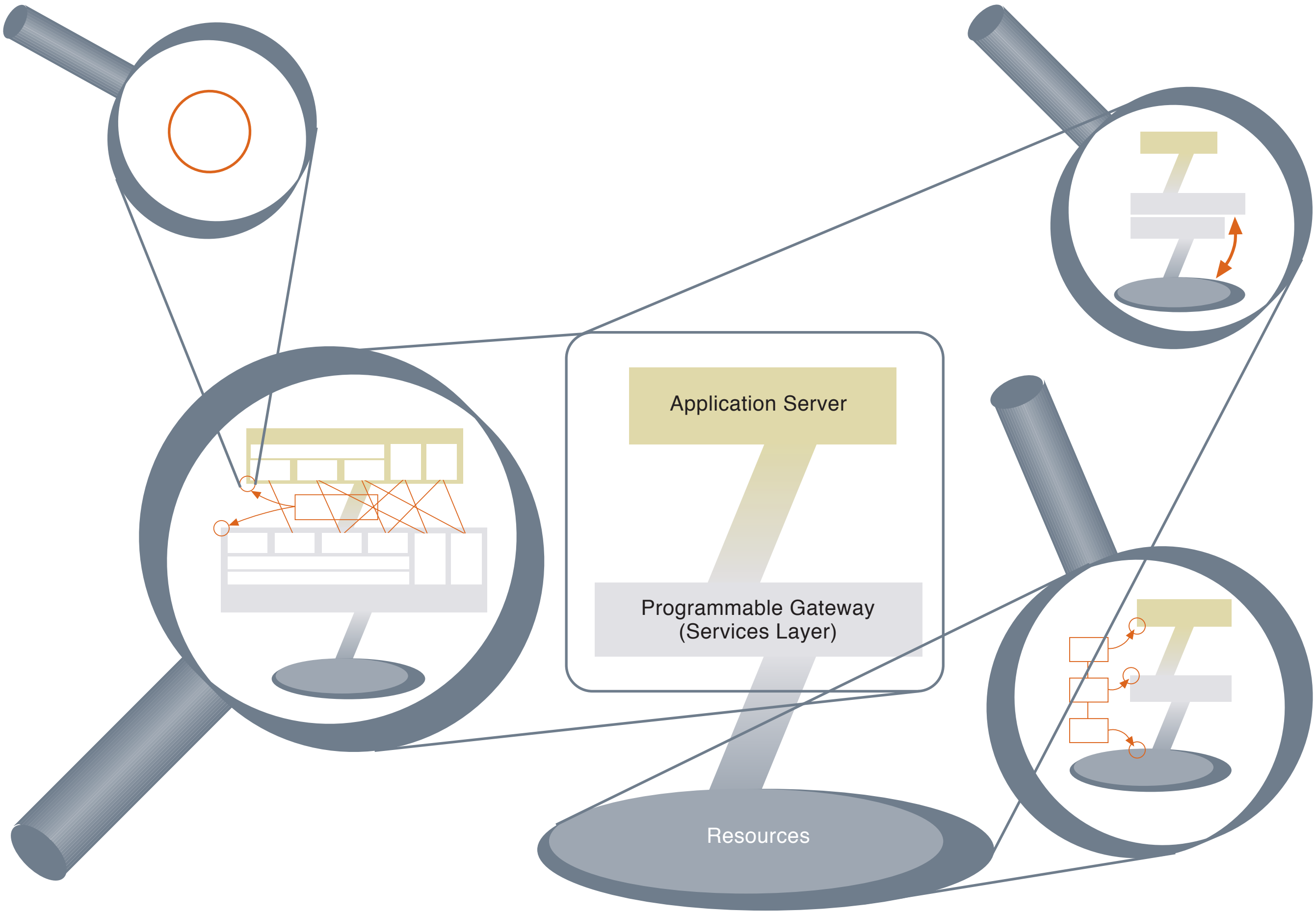


Figure 12 Overview of programmability efforts

ing helicopter view of all concepts and technologies and hence show the correlations and associations.

Figure 12 shows all initiatives aimed at the further evolution of service creation in next generation networks together. In the figure references to the original figures are made. The overall programmable architecture discussed in Figure 1 is at the center of Figure 12. The referenced Figure 3 discusses the Parlay APIs and their technology specific realization: WSDL, OMG IDL, and JAIN SPA. These realizations concern the services layer and application server. Figure 4 shows the Parlay X architecture in which functionality is mainly delegated to the Parlay APIs discussed in Figure 3. However, as Parlay X exposes specific network capabilities as opposed to the common capabilities exposed by the Parlay APIs, some Parlay X APIs can only be implemented through interaction with the resources layer. Therefore, the magnifying glass concerns the services layer, application server, and the resources layer. Figure 5 focuses on an alternative programming means: scripting languages. Scripting interpreters can be found in the services layer as well as in the application server. Finally, Figure 8 shows that programmability of network intelligence can also be achieved through designing protocols that interact with the resources layer.

References

1 *3rd Generation Partnership Project; Technical Specification Group Core Network; Open Service Access (OSA); Application Programming Interface (API) Mapping for Open Service Access; Part 4: Call Control Service Mapping; Subpart 4: Multiparty Call Control ISC (Release 5).* 3G TR 29.998-04-4 v5.0.0. December 3, 2002 [online] – URL: http://www.3gpp.org/ftp/Specs/latest/Rel-5/29_series/29998-04-4-500.zip

2 Bakker, J-L, Jain, R. *Next Generation Service Creation Using XML Scripting Languages.* ICC 2002, New York, NY (USA), April 28 – May 2, 2002.

3 Bakker, J-L et al. Rapid Development and Delivery of Converged Services Using APIs. *Bell Labs Technical Journal*, 5 (3), 12–29, 2000.

4 Grech, M, Torabi, M, Unmehopa, M R. *Service Control Architecture in the UMTS IP Multimedia Core Network Subsystem.* IEE 3G2002 Mobile Communications Technologies, London, UK, 8–10 May 2002.

5 IETF. *PSTN and Internet Internetworking (pint) Charter.* October 18, 2000 [online] –

URL: http://www.ietf.org/html.charters/pint-charter.html

6 IETF. *Service in the PSTN/IN Requesting InTernet Service (spirits) Charter*. October 18, 2000 [online] – URL: http://www.ietf.org/html.charters/spirits-charter.html

7 ISO. *Interface Definition Language (IDL)*. March 1999. (ISO 14750)

8 Jain, R, Bakker, J-L, Anjum, F. Java Call Control (JCC) and Session Initiation Protocol (SIP). Invited Paper. *IEICE Trans. Communications*, E84-B (12), 3096–3103, 2001.

9 Lennox, J, Schulzrinne, H. *Call Processing Language Framework and Requirements*. May 2000. (URL: http://www.ietf.org/rfc/rfc2824.txt)

10 Lennox, J, Schulzrinne, H. *CPL: A Language for User Control of Internet Telephony Services*. (Work in progress.) January 2002. (URL: http://www.ietf.org/internet-drafts/draft-ietf-iptel-cpl-06.txt)

11 OASIS (Organization for the Advancement of Structured Information Standards). *The XML cover pages*. October 18, 2002 [online] – URL: http://www.oasis-open.org/cover

12 OMG. *Unified Modeling Language (UML)*. Version 1.4, September 2001. October 18, 2002 [online] – URL: http://www.omg.org/technology/documents/formal/uml.htm

13 OMG. *Common Object Request Broker Architecture (CORBA)*. Version 2.6, Dec 1, 2001. October 18, 2002 [online] – URL: http://www.omg.org/technology/documents/formal/corba_iiop.htm

14 Parlay Group. *Parlay APIs Overview*. October 18, 2002 [online] – URL: http://www.parlay.org/docs/Spec3_es_20191501v010101m.pdf

15 Rosenberg, J et al. *SIP: Session Initiation Protocol. June 2002*. October 18, 2002 [online] – URL: http://www.ietf.org/rfc/rfc3261.txt

16 Stretch, R M. The OSA API and other related issues. *BT Technical Journal*, 19 (1), 80–87, 2001.

17 Sun Microsystems. *JAIN SIP Specification 1.0, Java Specification Request (JSR) 32*. October 18, 2002 [online] – URL: http://jcp.org/aboutJava/communityprocess/final/jsr032/

18 Sun Microsystems. *JAIN INAP API Specification 1.0, Java Specification Request (JSR) 35*. October 18, 2002 [online] – URL: http://jcp.org/aboutJava/communityprocess/final/jsr035/

19 Sun Microsystems. *JAIN Service Creation Environment (SCE) API Java Specification Request (JSR) 100*. October 18, 2002 [online] – URL: http://jcp.org/jsr/detail/100.jsp

20 Sun Microsystems. *JAIN Java Call Control (JCC) API Java Specification Request (JSR) 21*. October 18, 2002 [online] – URL: http://www.jcp.org/jsr/detail/21.jsp

21 Sun Microsystems. *JAIN Coordination and Transactions (JCAT) API Java Specification Request (JSR) 122*. October 18, 2002 [online] – URL: http://jcp.org/jsr/detail/122.jsp

22 Sun Microsystems. *The JAIN APIs*. October 18, 2002 [online] – URL: http://java.sun.com/products/jain/

23 Unmehopa, M R et al. The Support of Mobile Internet Applications in UMTS Networks Through the Open Service Access. *Bell Labs Tech. Journal*, 6 (2), 47–64, 2002.

24 W3C. *Extensible Markup Language (XML)*. October 18, 2002 [online] – URL: http://www.w3.org/XML/

25 W3C. *Simple Object Access Protocol SOAP 1.1. W3C NOTE, 8 May 2002*. October 18, 2002 [online] – URL: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

26 W3C. *Voice Browser Call Control: CCXML Version 1.0. W3C Working Draft, 21 February 2002*. October 18, 2002 [online] – URL: http://www.w3.org/TR/ccxml/

27 W3C. *Web Services Description Language (WSDL) 1.1. W3C Note 15 March 2001*. October 18, 2002 [online] – URL: http://www.w3.org/TR/2001/NOTE-wsdl-20010315

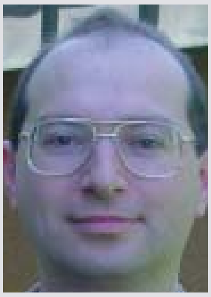
28 W3C. *XML Schema*. October 18, 2002 [online] – URL: http://www.w3.org/XML/Schema

Real World XML Web Service Scenarios for Telcos

ERIK VANEM, GEORGE BILCHEV, EDWARD BUCKLEY AND THOMAS HOPPE

Erik Vanem (30) received his MSc in Physics from the University of Oslo in 1996. Before joining Telenor R&D in 2000 he worked as a geophysicist at PGS Reservoir, as a research assistant at the Norwegian Defense Research Establishment and as a physics teacher at the Oslo University College. Since 2000 he has worked with user centric services, mobile applications and services, Voice over IP and mobility management in next generation wireless networks. Recently he has been working with distributed computing and XML Web Services in the PANDA group (Personal Area Networks and Data Applications) at Telenor R&D.

erik.vanem@telenor.com



Dr. George Bilchev (32) joined BT in 1996. He holds a Ph.D. in engineering design from the University of Plymouth and an MSc in Artificial Intelligence from New Bulgarian University. While in BT, George has been working on m-commerce applications, personalisation, service-oriented architectures and complex systems research.

george.bilchev@bt.com

1 Introduction

As XML Web Services are rapidly becoming widespread and straightforwardly used by more and more actors in the global marketplace, it is important from a Telco's point of view to identify the possible areas where it can benefit from these emerging technologies. In what kind of scenarios can a Telco play an important role and how can they best exploit the new opportunities that arise with this new approach to service provision?

This article tries to identify various real world scenarios that are relevant to Telcos with the emergence of XML Web Services. First, in section 2, a general overview with some different visions about what XML Web Services can be is given. The rest of the article considers different scenarios where Telcos can benefit from XML Web Services. Most of the scenarios are realisable as of today, but also some visions about future Web Services scenarios are mentioned. Finally, a conclusion sums up the article.

2 Overview

"Web Services" is the buzzword of today in the software industry. A lot of hype is made about them and this is not very surprising. From a certain perspective Web Services can be considered "as being for applications, what the WWW is for humans". The same way HTML links insert documents into a huge hypertext, the SOAP calls of Web Services can be perceived as interconnecting applications and eventually forming huge applications out of discrete distributed functionality. From this interpretation it is easy to develop the vision that Web Services can repeat the success the WWW has had for humans once more, this time for applications.

On the other hand, there are critical voices that say that there is nothing new in Web Services, since the underlying concepts already exist (in e.g. CORBA, DCOM, RPC) and since some of the underlying standards are relatively old and mature (e.g. HTTP, XML). This argument goes even further: Web Services do not introduce new functionality, thus they will not introduce new business opportunities.

EURESCOM Project P1209

The P1209 project is run by EURESCOM with partners from Telenor, British Telecom and Deutsche Telekom. It started up in May 2002 and will run until March 2003. Regarding Telenor's contribution, this is a joint effort from different research programmes at Fornebu and Trondheim – Future Wireless World, Internet Network Architecture and Service Platforms are all participating in this project.

The focus of P1209 is on XML Web Services and its opportunities for Telcos, which are in a great position to benefit from the development of this progressive technology. The development of XML Web Services will alow them to become an application service provider and will allow third party content and application providers to rapidly deploy applications that will utilise Telcos' networks. Telcos are thus in a pole position to either offer an infrastructure for Web services or to supply Web services on their own. However, Web services require different standards that are still evolving. These standards are implemented by competing companies and are based on different implementation technologies. It can therefore be expected that the development of an interoperable solution is not as straightforward as one would like to believe.

The main objectives of the P1209 project are the following:

- Evaluate key XML Web Services technologies
- Produce a technology overview
- Investigate relevant standards bodies' activities
- Understand key XML Web Services standards and future trends
- Analyse XML Web Services pertinent to the Telco environment
- Investigate new XML Web Services business models
- Experiment in a multi-party, multi-technology environment on issues such as interoperability, scalability, robustness, etc.
- Gain hands-on experience on available XML Web Services technologies
- Build a body of knowledge related to the design of XML Web Services

However, the vision is a reasonable one due to the combination of these concepts and standards and because of an implicit commitment of nearly the entire software industry to support the standardization efforts around Web Services. Furthermore Web Services are designed having application on the Internet in mind. The vision that there will be introduced a secure, stable, interoperable, standardized and platform-independent interface to function calls over the Internet in some future is thus realistic. This, in conjunction with the joint effort of the industry (which surely sees some business opportunities in their effort), gives reason to develop different visions about Web Services. Web Services can be perceived as:

- *Interoperable Internet middleware*. Even though currently available middleware like CORBA, DCOM, RMI etc. has found its market in intranets, utilisation of this technology over the Internet between partners from different domains that operate different infrastructures is limited and thus poses a great challenge for the integration of different partners' systems. Web Services can be considered as middleware, which establishes interoperability between different infrastructure platforms over the Internet.

- *Extensions of applications onto the Internet*. This vision refers to Web Services as the means for applications to obtain remote functionality from services running somewhere on the Internet. For Example, some spreadsheet program could contain some generic functions, which receive information about up-to-date currency conversion rates directly from providers on the Internet.

- *Means for the commercial exploitation of data and services on a "pay-per-use" basis*. This vision refers to Web Services as a means that could be used to sell data (like the above up-to-date currency conversion rates) or services (such as the transformation of different XML-based business formats) over the Internet.

- *Enabler for an open service market on the Internet*. Under this vision Web Services are regarded as the means for advertising, brokering and accessing of data and services on a dynamic basis. Service providers can announce their services in a UDDI registry. Service requesters can then find services appropriate for fulfilling their needs in these registries and Web Service technology provides the means for accessing these functions.

- *Means for automatic, dynamic e-business*. As an extension to the previous vision, this can be identified as the ultimate goal of web services. In this vision, e-business via Web services is realised as some kind of highly dynamic, peer-to-peer like business transactions, which are based on on-the-fly negotiated contacts to and contracts with Web Service providers. Obviously, this vision is still far from being fulfilled.

Whatever vision one has in mind, the purpose of the entire development is of course the realization of business. The goal would be either to make future developments less expensive or more effective, or to develop new business opportunities. This quest for the business cases of Web Services is currently ongoing.

Even if there is little new with Web Services, the things that are new will have some impact on current businesses, and it will also allow the establishment of some new businesses. First of all the software industry will profit (either by developing and selling software platforms, tools and servers, or by realising integration projects). Secondly, providers of information and services might profit (the first examples to be mentioned here are Google [1] and Amazon [2]). Suppliers of infrastructure services might profit (examples of such companies are Primordial [3], Grand Central [4], Flamenco [5] or TalkingBlocks [6]), which provide the infrastructure for Web Service Networks) and finally the operators of registries might profit from the development of Web Services.

For Telcos, important questions arise: Will they find some business opportunities in Web Services or not? What will those business opportunities look like and which business opportunities should they pursue? As a starting point for such an investigation, this article describes some generic scenarios that on the one hand can have some business impact for Telcos and on the other hand could be of direct interest to Telcos' business units.

3 Enterprise Application Integration

EAI (enterprise application integration) is a business computing term for the plans, methods, and tools aimed at modernising, consolidating, and coordinating the computer applications within an enterprise. Typically, an enterprise has existing legacy applications and databases and wants to continue to use them while adding or migrating to a new set of applications that exploit the Internet, e-commerce, extranet, and other new technologies. EAI may involve developing a new total view of an enterprise's business and its applications, seeing how existing applications fit into the new view, and then devising ways to efficiently reuse what already exists while adding new applications and data.

After receiving a BSc in Computer Science from the University of Bradford, Edward Buckley (23) has worked for BTexact Technologies since 2001. He has worked on designing mobile commerce services and website design and implementation. Currently he is working in the area of XML Web Services, and technology to support the use of the Semantic Web in eBusiness and Knowledge Management.

edward.buckley@bt.com

Thomas Hoppe (42) completed his study of computer science at the Technical University (TU) Berlin in 1986. After working with a German automative company and a stay abroad he went back to TU Berlin and worked on two knowledge representation projects. He received his doctor's degree in logic programming in 1995 from the Uni. of Dortmund and joined Deutsche Telecom Berkom GmbH in 1996. Since then he has worked in the field of search engines, where he holds two patents, and more recently business models in the area of B2B e-commerce. He is currently Project Manager in the T-Systems Nova GmbH, a business unit of Deutsche Telecom AG.

thomas.hoppe@t-systems.com

EAI encompasses methodologies such as object-oriented programming, distributed, cross-platform program communication using message brokers with Common Object Request Broker Architecture and COM+, the modification of enterprise resource planning (ERP) to fit new objectives, enterprise-wide content and data distribution using common databases and data standards implemented with the Extensible Markup Language (XML), middleware, message queuing, and other approaches.

EAI exploiting Web Services technologies can be beneficial to Telcos in the same way it would be to any other types of enterprises in integrating their own different internal legacy systems. Apart from that, business units in Telcos with expertise in EAI can find a business opportunity in acting as integrator for other enterprises that need help from external experts in order to run internal projects on EAI within their systems.

4 B2B Integration

B2B integration is another area where Web Services technologies can be exploited. The following describes a scenario of B2B integration relevant to Telcos.

In the outlined scenario a wholesaler wants to establish business relationship with a retailer. The main steps in such scenarios are:

- Preoperational B2B
  - Establishment of B2B Capabilities
  - Establishment of B2B Contract

- Operational B2B
  - Running of B2B collaborations

We assume that the wholesaler does not have B2B implemented, but that the retailer has complete B2B:

The wholesaler needs to establish Collaboration Protocol Profile (CPP) and search for CPP for a possible Retailer in a Repository. The wholesaler needs to work with two different third parties, a System Integrator to establish CPP capabilities and a Registrar to be registered in the Repository.

The UML use case diagram in Figure 1 provides an overview of such a generic scenario.

The relationships between actors and their respective roles are shown in Table 1. It should be noted that a Telco could take all of these roles in different B2B scenarios. It can be the Wholesaler that wants to establish contact with Retailers to sell its services to the end customers. It could also be a Retailer that resells services on behalf of others and integrate other's services in their products. Finally it could act as the Service Integrator that help its customers to implements B2B capabilities.

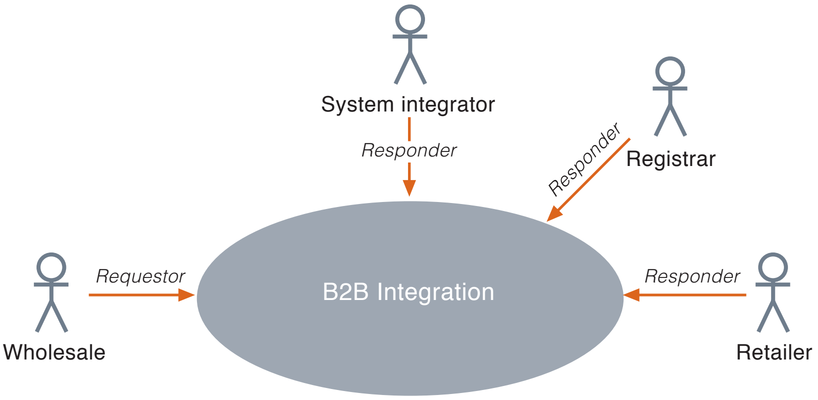


Figure 1 Actors in a generic B2B integration scenario

Actor Name	Role Description	Role Type
Wholesaler	Wants to establish contacts with retailers to sell its products to the market (Requestor)	Organisation
Retailer	Sells products on behalf of wholesaler	Organisation
Service Integrator	Implements technical B2B capabilities	Organisation
Registrar	Defines and register CPP/CPA in Repository	Functional

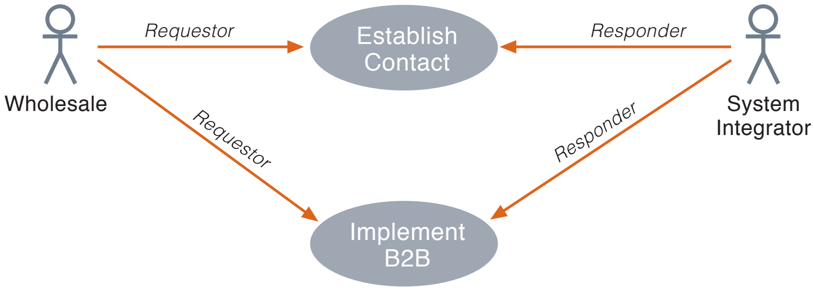
Table 1 Actors and roles in a B2B scenario

The first step for the wholesaler is to establish B2B capabilities, and this is illustrated in Figure 2.

The Wholesaler would first browse a repository (ebXML/UDDI) manually to find a suitable System Integrator. The Wholesaler would then run manual negotiations with the System Integrator before signing a TPA (Trading Partner Agreement). These transactions are not based on B2B standards other than knowledge about B2B issues by at least one of the business partners. The B2B issues should be stated in the contract.

The next step is for the System Integrator to implement the TPA requirements for the Wholesaler. This step does not include any specific B2B transactions and the implementation is controlled by the contract.

Figure 2 Implementation of B2B for Wholesaler



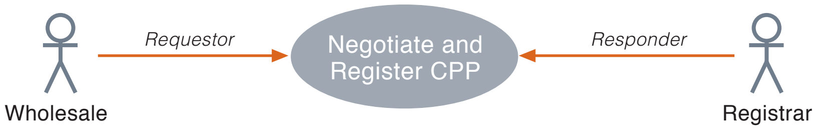


Figure 3 Collaboration negotiation and CPP registration

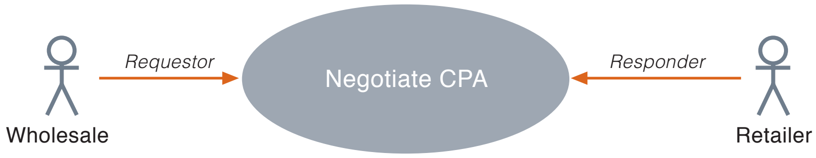


Figure 4 CPA negotiation

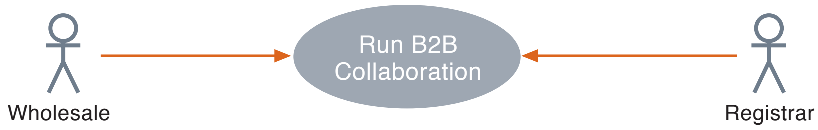


Figure 5 Running B2B collaboration

Then, the Wholesaler should use a Registrar to establish a CPP (Collaboration Protocol Profile) and put it into a Repository as illustrated in Figure 3. Regarding the transactions, the Wholesaler will have B2B capabilities installed, but the communication with the Registrar can also be by other means. The Registrar can be a software package installed at the Wholesaler site, but it must interface the B2B Repository and satisfy certain requirements.

Having completed these steps, the Wholesaler is now ready to negotiate a CPA (Collaboration Protocol Agreement) with the Retailer as illustrated in Figure 4.

The Wholesaler does a B2B search in a Repository for a Retailer Partner, starts negotiating CPA with a desired Retailer before they both sign the CPA agreed upon. The transactions involved in this job will depend on the tool installed at the Wholesaler site, which must comply with the specifications.

With a CPA signed, one is now ready to run B2B operational collaboration in accordance with the CPA (Figure 5).

The transactions to be run will depend on the Business, but they will all have to be stated in the CPA. For certain businesses it will be possible to use already developed transactions, which will be visible in the B2B Repository. It will also be possible to use existing payloads like EDI (Electronic Data Interchange) if required. For new transactions, they must comply with relevant standards.

5 Web Service Provision

There are different instances of scenarios that can be summarised under the scenario class “Web Service Provision”. In general, Web Service Provision means to offer Web Services to customers, either in order to distribute content or to make content services available.

5.1 Telco as a Web Service Provider

In this scenario class a Telco takes on the role of a Web Service Provider. It provides information or services commercially via Web Service interfaces for applications belonging to its customers.

The information and services which could be provided by a Telco range from information it already owns (like e.g. phone numbers, address information, yellow pages, telephone bills, personalisation and localisation information), and their corresponding services (e.g. phonebook lookup, configuration of telephones and Internet accounts, product search and comparison, bonus programs), to information which they redistribute or resell (e.g. stock quotes, news, audio, videos, multimedia presentations), and services which they offer for third parties (e.g. access to legacy systems, shopping carts).

The customers could be private consumers (e.g. if the Web Service allows customised access to streaming content), or they could be business users (e.g. if functionality of remote office packets is made available via Web Services). Additionally the customers could be application development companies, which either hard-code calls to Telco Web Services (like the afore mentioned phone book lookup) or pre-configure them into their applications.

Usually the access to these Telco Web Services will be via some application, either by a web page or integrated into some desktop or server application.

Web Services provided by a Telco in this way could either be free (for example during the market entry, as a marketing instrument or for advertisement purposes) or they could be available for a fee, either as a subscription fee, in pre- or post-paid mode. Different accounting schemes could be used here either pay-per-use (-per-volume, -per-time), monthly with limit, etc.

5.2 Players and their Roles

Telco

As mentioned already the Telco plays the role of a Web Service provider, which makes information and services via Web Services available in this scenario. Provision in this context means that the Telco owns and operates the systems which deliver functionality and which are made accessible via Web Services.

The information and services might be free (in that case they function only as a marketing or service instrument) or they are commercially available at some fee.

Especially in the case of commercially available Web Services, it is reasonable, that the Web Service provider makes use of some infrastructure services, e.g. for secure messaging, billing, authentication, authorisation etc. These services can either be provided by the Web Service provider itself or they can be obtained from some external provider of Web Service Infrastructure Services.

The Web Service Provider can own the information and services provided or this information can be obtained from some external provider. Thus different constellations can be identified:

- In the case of a Telco acting as Web Service provider, which provides its own information and services, it is clear that this information and services will originate from the telecommunication domain, i.e. that they are either related to phone, mobile, Internet or networks (Figure 6).
- In the case of a Telco acting as a Web Service provider, which provides external information and services, the Telco may either redistribute or resell the information, or it augments it with own functionality. In both cases the Telco may get some margin (Figure 7).
- In the case where a Telco combines or aggregates different information or services, it actually combines them to some value added information that could be charged on its own (Figure 8).

External Provider
External Providers in this scenario are all parties from which a Telco obtains external information, external services or infrastructure services.

Customer
Customers in this scenario are those parties that make use of the Web Services provided by the Telco. More precisely customers are those instances whose applications make use of the provided Web Services. They include private end users, business users and application developers, but also other Web Service providers.

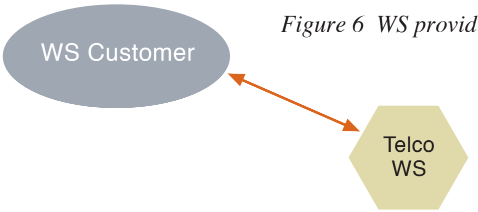


Figure 6 WS providing its own information or services

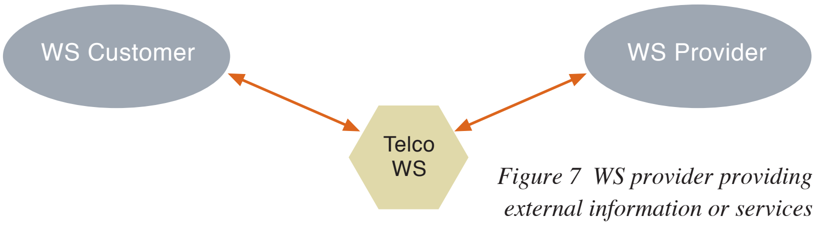


Figure 7 WS provider providing external information or services

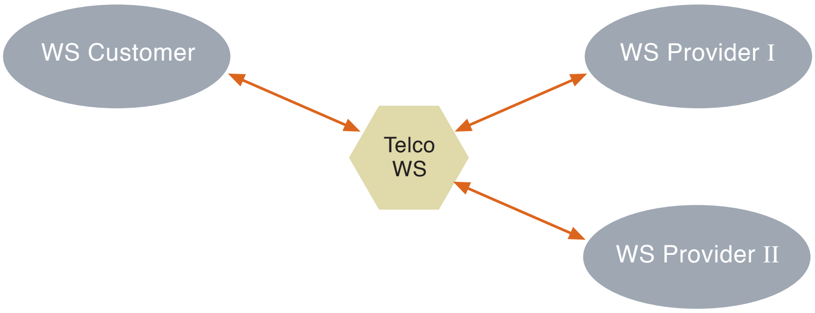


Figure 8 WS provider combining external information or services

5.3 Relationships

The primary relationship between these different roles is of course the usual customer-supplier relationship. The Telco takes on either the role of a customer or the role of a supplier. However, the Web Service itself introduces a major dependency, whether it is a commercial or a free service. While trading partner agreements suffice in the latter case, the former case needs to go further by establishing a solid contractual basis for the relationships.

Telco – External Provider
Usually the relationship between the Telco and some external provider will be of some commercial nature. In this relationship the Telco takes on the role of a customer, while the external provider takes on the role of a supplier.

Even so it is conceivable that in some near future, relationships to suppliers will be established dynamically and automatically, requiring of course that all issues of trust and automatic contracting are solved. In most cases it will be in the interest of the customer to establish a longer lasting relationship to its suppliers. Otherwise he could hardly warrant an operational Web Service to his own customers.

The information and services a Telco uses from some external provider are the prime relationship between them. However a solid business relationship needs to be established in order to agree on technical details of the usage and on the business issues. Trading partner agreements alone are not sufficient here, and contracts need to be established.

Caused by the establishment of contracts between both partners it becomes obvious that the Telco and the external provider need to speak the same language. This means that they must agree on the interpretation and on the understanding of the terms they use in their contracts and in their communication.

Telco – Customer

The relationship between a Telco and its customer resembles the previous relationship, but with opposite roles. Now the Telco plays the role of the supplier.

The relationships to the different customers need to be differentiated depending on the type of the customer, i.e. whether it is some private end user, some business user or an application developer:

- In the case of a private end user the relationship is not guaranteed to be long lasting.

- A business user as a customer is probably more interested in some longer lasting relationship, but there exists no guarantee for a longer lasting relationship in this case either.

- Application developers, however, who integrate the usage of the Telco's services into their own applications establish a long lasting relationship.

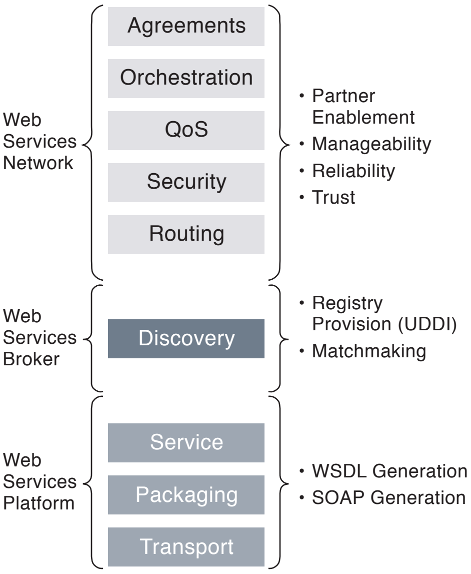


Figure 9 Web Services Architecture Stack

All other aspects – trading partner agreements, contracts and usage of the same vocabulary between the Telco and its customers – will be the same as described in the previous section.

6 Web Services Broker

Technically speaking, a Web services broker is an intermediary positioned anywhere within a Web Service message path that performs a value-added function. This very broad definition includes entities ranging from all sorts of software components and network appliances that function as part of the messaging infrastructure through third party Web Services networks and carriers. However, in this section we will focus on the role of a third party intermediary organisation that hosts Web Service interfaces and brokers communications between requesters and providers (as depicted in the "Discovery" layer of Figure 9).

Web Services brokering provides the opportunity of an organisation to become an indirect supplier channel. The main role of the Web Services broker is to connect (for a fee) users with service providers. It is believed that suppliers of commodity Web Services (like reservations, shipping and content) will deliver more than 70 percent of them through a Web Services broker within a few years.

This section presents a scenario where Telcos play the role of a broker who gets requests from Web Service developers or applications and who directs them to selected/found destination Web Service provider. To better understand how this scenario fits the Web Services architecture stack and ecosystem, the reader is referred to Figure 9.

Figure 9 describes a generic Web Services stack where layers are grouped into three groups: the Web Services platform, the Web Services broker and the Web Services network. The role of the Web Services broker is to provide as a minimum a registry for publication of Web Services and means of searching for Web Services. However, a broker might also provide categorisation, mediation and validation of Web Services. Brokers do not create services, nor do they manage or host services. Their main focus is to build a large supplier network and to exploit that network to generate incremental revenue. To achieve that, Web Services brokers will have to add value to the minimum requirements of registry provisioning and searching. Value-added broker services might include aggregating an appropriate set of services and cataloguing those services as a portfolio of offering, building vertical catalogues of Web Service components, exploiting their brand name to offer those services through the registry, etc.

Figure 10 describes the place of the Web Services broker in the Web Services ecosystem. The broker is linked to the Web Services network ,and it is quite often the case that a Web Services network will provide a broker as part of their solution. Also recalling the broader definition of a Web Services broker, the tighter coupling between the Web Services network and the broker would allow brokerage to occur at other levels of the Web Services stack. For example, solutions exist (mainly from Web Services network vendors/providers) where horizontal value-added brokerage occurs to provide encryption, authorisation, access control, monitoring, metering, logging, auditing, provisioning, billing, non-repudiation, etc. It is worth noting that the above-mentioned horizontal brokerage services will become commoditised (especially when the standards mature and the big vendors incorporate them in their platforms). Ultimately, such commoditisation will drive prices down and trim profit margins. Therefore, the selected scenario in this section will focus on providing vertically oriented brokerage services on behalf of an industry specific community (the Telecom sector).

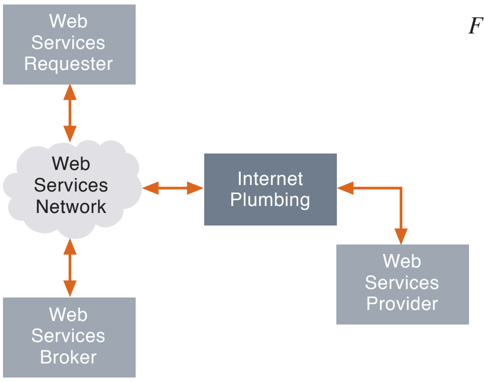


Figure 10 Web Services Ecosystem

6.1 Example Scenario

Consider the following scenario for a Telco as a Web Services Broker: A large Telco is looking for new opportunities to leverage its expertise in the Communications market and its network of suppliers, so a decision is made to become a vertical Web Services broker for the Communications sector. The broker will run a UDDI registry where suppliers will be able to register Web Services. A taxonomy/categorisation will be provided to facilitate search. As an added value to that, the broker will provide packaged offers where one supplier is used for more than one Web Service or discounts where the Telco has already agreed volume deals with given suppliers (thus aggregating volume and leveraging existing relationship with suppliers).

To leverage its knowledge of the supplier network, the broker will provide ratings of the participating suppliers and it can further provide ratings for individual web services based on either a testing programme or customer feedback or popularity. Other value-added services such as payment brokerage, availability and QoS could be offered if the broker is part of a Web services network solution.

The Telco plans to use its established brand to attract customers to search for solutions through its registry. To aid that, the broker is providing value-added solutions consultancy and integration services. The solution consultancy will address the problem of solution composition and workflow at least until the Web Service orchestration standards mature. The integration will mainly involve those missing components of the overall solution, which are not yet Web Services enabled.

Another value-added service the broker will provide is translation of Web Services calls. The translation (or mapping) will be required in real life because two different suppliers of say an SMS Web Service could provide two different SOAP interfaces for sending SMS. All SMS suppliers will be listed at the same level of the taxonomy, but without real time translation/mapping a customer's application will not be able to dynamically select a provider. The translation could happen at design time at the customer's end, but the value of a translation service run by the broker is that when new providers are added only the broker would implement the translation and not all the customers.

The Telco is charging customers either per use of the registry or a subscription to use the registry. Any consultancy and integration services will be charged extra at current consultancy or integration rates.

7 Web Services Infrastructure Services

As with all distributed applications, some main concerns related to Web Services are:

- *Trust*. This is intertwined with security, due to the validation of origin being an authentication issue.

- *Security,* such as Data confidentiality, authentication and validation of data origin, data integrity and non-repudiation, and authorisation of user access.

- *Reliability and failover*. As well as being reliable, distributed services need to be "aware" of their environment and handle failures (and timeouts) gracefully.

- *Scalability*. Can a service handle many concurrent requests without major degradation of performance and reliability?

Even though Web Services in themselves are simple to create, an infrastructure needs to be applied as with any other business application, and Web Services can be used as the actual integration infrastructure.

Infrastructure Services will allow application developers to design and develop without needing to worry a great deal about how the services will be managed. Essentially this is a combination of value-added network services.

The Web Services Infrastructure Services scenario takes the Telco as a provider of basic infrastructure services offered to Web Services Providers, such as:

- Secure messaging – the ability to communicate reliably and securely over an insecure medium (e.g. the Internet).

- Authentication – being able to identify who is being communicated with (trust between partners).

- Authorisation – only allowing certain users (who have been authenticated) access to certain parts of a system; this includes e.g. configurable usage policies.

- Billing – a standard method for payment to occur.

- Payment – including factors such as non-repudiation (being able to confirm that correct billing has occurred).

The rationale behind this is that Providers cannot implement the infrastructure needed for business collaborations in Web Services on their own, for example due to lack of resources and the expense involved. The Web Service Providers would be charged for this instead of the End Users and players will be able to communicate with each other in a more easy and trustworthy manner.

Additional infrastructure services can be summarized here, e.g.

- Auditing & logging – monitoring the access and usage of Web Services.

- Metering & accounting – metering Web Service resource consumption in terms of number of calls, required time or transported data volume and condensing these figures.

- Monitoring & filtering – ensuring the healthiness of Web Services by alarming their providers about problem situations or by ensuring system availability in the case of denial-of-service attacks.

These infrastructure services can therefore be described as a management platform. The actual Web Services implemented will make use of the "basic" infrastructure services in order to remove all the complexity of management. Within the Web services ecosystem, the Infrastructure Services provider fits into the Web Services Network paradigm, allowing requesters and brokers to communicate with the Providers.

7.1 Players and their Roles

Telco

In the described scenario, a Telco could take the role of an infrastructure provider, allowing Web Service Providers to make use of management services. These services will be generic – not specific to an actual Web Service – such as messaging. Implementation issues, however, mean that minor customisations may be required when communicating with specific Web Service Providers. The Telco will handle the basic infrastructure services as described above.

Web Services Providers

These are the providers of the actual Web Services that are accessible to other providers, users (which in this model are mainly other Web Service Providers), etc. In effect, they will "sit on top" of the Telco, since their services will make use of the infrastructure provided by the Telco. Once Services begin using the infrastructure, they will be dependent on the Telco. Therefore a comprehensive trading agreement is required between the Telco and the Web Service Providers.

Certificate Authority

For the authentication and authorisation services, two solutions can be applied:

- The Telco acts as its own Certificate Authority.

- A separate Certificate Authority, a third party, is used. This is the recommended solution, as the trust will be greater – the Telco will not control the certificates and instead pass the control to an independent organisation.

This also applies to the secure messaging provided by the Telco, where certificates will be applied to messages sent between Players.

7.2 Relationships

Sun proposes an overview of the relationships in [7] where smart = context aware. The majority of the relationships in this model will require TPAs (Trading Partner Agreements) between the Players involved in a relationship or interaction. It would be beneficial to be able to define these in a pragmatic and consistent way, and ebXML is ideally suited for this.

Telco – Web Services Provider

The communication between Telco and Web Service Providers will depend on the architecture model used, of which there are two main ones, centralised and de-centralised.

In the centralised approach, the Telco acts as a Hub for connecting Web Service Providers and Web Service Consumers (Figure 11). This means that all communication passes through the Telco. For ease of management and coordination this is a good solution, however the scalability of this model is questionable – the reliability and performance of the hub will depend on the amount of traffic passing through the hub, which would cause increasing investments into the operational infrastructure. In addition, complexity is added to RPC (Remote Procedure Call) or synchronous style Web Services, where the Services communicate directly with each other; overheads and potential failure points are added to the development and deployment.

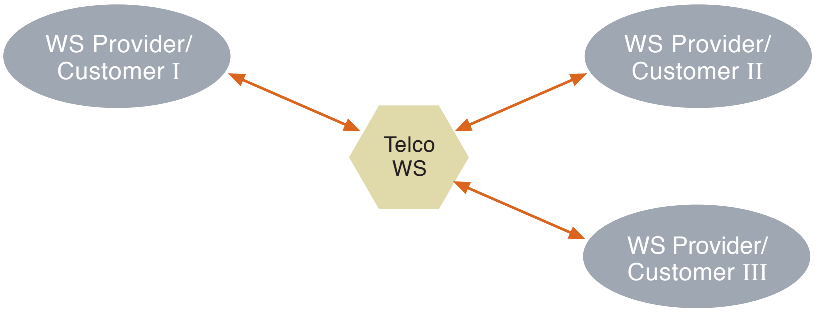


Figure 11 Centralised model of a WS Infrastructure Provider

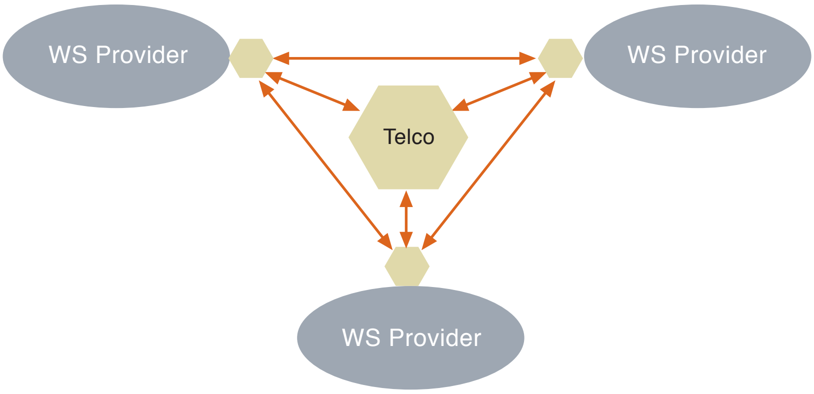


Figure 12 De-centralised model of a WS Infrastructure Provider

The other model is a de-centralised approach, where there is centralised management but with the communication being de-centralised as illustrated in Figure 12.

The idea is that the communication between Web Service Providers is peer-to-peer. The main advantages to this model are that the Providers are not as tied into the Telco as with the centralised model, and that the bottlenecks associated with the Hub approach will not exist.

For the Telco to be able to manage the services provided by the Web Service Providers, a proxy is required at all ends of a Web Service-based communication at each Web Service Provider. This is a remote piece of software associated with the Infrastructure Management, which all Web Services traffic from the relevant Provider passes through. Because the proxy exists at both ends, the communication can be securely sent and other infrastructure services can be hooked into the proxy. The communication between Telco and Web Service Provider in this model will not be constant, but sent only when needed (e.g. authentication), for reporting purposes or for upgrades of the proxy.

This model is more in line with the distributed loosely coupled concept of Web Services, but it does have its disadvantages:

- More complicated coordination. The Web Services are not in direct communication with the Infrastructure Provider, but communicate through the proxies. The complicated task of managing numerous, exposed proxies could cause deployment and usage problems.

- Due to the proxies being de-centralised and exposed, they will be much more susceptible to attack, which in turn puts the centralised management at risk.

There is also the possibility that the de-centralised approach can be taken, but without the central management. In this case the Infrastructure Provider provides the management services as a packaged solution to Web Service Providers. This would mean the Telco renounce control over the infrastructure. They would therefore only benefit from licensing, and the real task lies in finding licensing models that will turn every use of the packaged solution into profit. Other benefits such as secure messaging and reporting are also much harder to provide with this approach.

The communication between Telco and Web Service Provider will not just be for the running and usage of provided Services; other factors such as Service development and integration into the infrastructure services will be part of the agreement between the two Players.

The architectural model chosen depends on what particular Web Service Providers require, i.e. the types of services being offered and what type of management is required. Generally, if performance were important then the de-centralized approach would apply, and if security were a high priority then the centralized hub approach would be the preferable of the two.

Web Services Provider – Web Services Provider

As Telcos have two alternative architectures, the communication between Web Service Providers can be in two different ways. Either they communicate "directly" (peer-to-peer, but communicating through each other's proxy), or indirectly through an infrastructure hub.

Telco – Certificate Authority

This communication will be relatively simple, since each Player knows which Services at either end will be used. Security is an obvious requirement, which is an integral part of the infrastructure anyway, due to the level of trust that will be placed on the certificates by every Player. The agreement between Telco and CA is probably the most important trust link in the infrastructure.

8 Telcos as Provider for Mobile Web Services

Concerning the general characteristics this scenario class follows the scenario class above. However in the mobile case, it has to account for the characteristics of the mobile context, i.e. mobile users, mobile devices and applications in a mobile environment.

In the scenarios covered by this class it is more reasonable that the Web Services are provided and hosted on some server system, while the mobile devices just issue the calls to them. This gives reason to conclude that the prime customers covered by this scenario class are private end users or business users.

Mobile users might range from users using laptops connected via some wireless network (WLAN, Bluetooth) to an Intra- or Internet, to users of PDAs and cell phones (utilising GSM, GPRS, UMTS) and combinations thereof.

Depending on the used mobile devices the used Web Services need to account for small memory footprints. Additionally the exchanged amounts of information and SOAP messages cannot be too large, because of limited communication speeds.

As a further consequence of the limited memory footprints, it is conceivable that applications on the mobile devices will be pre-determined in advance and not via some UDDI registry dynamically, since the latter would require dynamical binding and hence additional application code on the mobile device.

Further dependencies originate from the mobile context, such as connections that are temporarily lost due to "radio holes", situations where the mobile net is too crowded (at "hot spots" such as airports, etc.) or where no access network is accessible.

9 Other Scenarios

In this article several general scenarios have been described, which can be of interest to Telcos in general. These scenarios are either already realisable today or will be in the near future. In addition to these, there are also a number of other possible scenarios, which will only be mentioned briefly in this section.

Web Services Hub

A Web Services Hub can be considered as a kind of "Web Services marketplace", where Web Service Providers offer their Web Services via a marketplace model. More precisely the Web Services Hub could be a combination of Web Services Broker and Web Services Infrastructure Services for closed customer/supplier groups, forming an important intermediate step on the road to "dynamic business webs". A Web Services Hub will most likely evolve from usual marketplaces, by extending them with Web Service functionality. An important factor for such an evolution will be the customer and supplier base of existing marketplaces.

Web Services Trust Center

On the road to establishing automatic e-Business via Web Services several major obstacles need to be overcome. While the technical obstacles can be overcome quite easily, the discussion of solutions to other obstacles (e.g. automatic contracting and establishing trust) has not yet started. An interesting scenario might arise if the Web Services Broker extends its registry by value added services, which are used to "monitor", "measure" and "judge" the QoS of Web Services. This information is an important step on the way to the establishment of trust in the form of service level certificates between two (as yet) unknown business partners, which can be sold and used for marketing purposes as well.

DBW (Dynamic Business Webs) Infrastructure Provider

Even though the ultimate vision for Web Services, i.e. Dynamic Business Webs, has not yet become a reality, it contains the scenario of a "DBW Infrastructure Provider". This will provide the required infrastructure for DBWs, where brokerage and infrastructure services are available for anybody and any application to perform dynamic business. Appropriate Web Services are searched for in a registry when needed and new business contacts are formed dynamically. Contracts are negotiated completely automatically, and Web Services are used to handle business transactions on a "pay-per-use" basis. Although this scenario is still quite unattainable, it seems to comprise the scenarios of Web Services Provider, Web Services Broker, Web Services Infrastructure Services, Web Services Hub and Web Services Trust Center, where a Telco would act as a full service provider or "one stop shop" for the DBW environment.

10 Conclusion

As has been shown in this article, a number of Web Services scenarios will be relevant to Telcos in the near future. They can use these technologies in order to integrate enterprise solutions and legacy systems internally or they can take on various roles in a B2B context. They can also get involved in Web Services provision either as a Web Service Wholesaler, Retailer or Broker or offer added value services to other Web Services actors, for example by offering Web Services Infrastructure Services. Either way, there are great opportunities related to the emergence of Web Services, and Telcos should consider carefully how to best take advantage of these opportunities.

References

1 *Google*. November 13, 2002 [online] – URL: http://www.google.com

2 Amazon.com. November 13, 2002 [online] – URL: http://www.amazon.com

3 *Primordial – WSBANG*. November 13, 2002 [online] – URL: http://www.primordial.com

4 *Grand Central Communications*. November 13, 2002 [online] – URL: http://www.grandcentral.com

5 *Flamenco Networks*. November 13, 2002 [online] – URL: http://flamenconetworks.com

6 *Talking Blocks*. November 13, 2002 [online] – URL: http://www.talkingblocks.com

7 *A Reference Architecture for smart Web Services*. November 13, 2002 [online] – URL: http://dcb.sun.com/practices/devnotebook/webserv_refarch.jsp

XML in Electronic Commerce and Electronic Business

SVEIN TORE JOHNSEN AND BERNARD QUARRE

Svein Tore Johnsen (60) graduated from NKI Technical College with Electronics and Cybernetics in 1969 and from Herriot Watt University, Edinburgh with Computer Science in 1975, and is currently Senior Research Engineer at Telenor R&D. He worked as consultant and project manager in different industry segments and joined Telenor in 1984 where he has been working with TMN for many years both in internal projects and in EU and EURESCOM projects. He has been Telenor project responsible for EURESCOM project P1106 and is at present working in EURESCOM project P1209 (XML Web Services).

svein-tore.johnsen@telenor.com

Bernard Quarre (61) graduated from Paris Technical College in 1965 with aeronautical engineering and obtained his M.Sc. from the University of Sherbrokke, Canada in 1966 in automation theory, and is currently Senior Research Engineer at Telenor R&D. Quarre has worked as chartered engineer in Peugeot, Paris and in Computas Simulation tools. He started working with Telenor in 1984, where he has been working on management aspects related to ATM-based broadband networks. He has been involved in several European projects and his research interests include both lower and upper level of the TMN (NEM-BM) model.
bernard.quarre@telenor.com

1 Introduction

1.1 Business to Business (B2B) and Business to Consumer (B2C)

Electronic Business has existed for some time where the best known standard is the UN/EDIFACT standard, which today is used by many big businesses. The UN/EDIFACT standard has not been a great success, mostly due to the high cost related to implementation and the limited reach for interoperability. The introduction of the Internet has opened for world-wide interoperability at a low cost, and has therefore again put the focus on electronic business and electronic commerce.

In B2B the market has been dominated by costly proprietary technology, and in B2C the view has been that electronic commerce is just buying from a catalogue over a web interface.

The trend today is that B2B is being standardised and that the content is coded into XML messages that can be interpreted by both humans and machines.

If this turns out to be a success all kinds of businesses (big or small) can do electronic commerce world-wide over the Internet at low cost. The Internet and XML are therefore key factors to make this happen.

For electronic commerce we usually talk about B2C and B2B where B2C means commerce between an end consumer and an enterprise while B2B means commerce between different enterprises. Both B2C and B2B are therefore part of what we call Electronic Commerce, and enterprises that want to do B2C or B2B must satisfy certain requirements for Electronic Business. The enterprises operate in different business areas, where the telecom business is one area.

The information to be exchanged between enterprises can be very special to one business area (vertical) or common to many business areas (horizontal).

For enterprises to be able to do B2C and B2B they must turn into electronic businesses. Enterprises today are run with a combination of manual and automated processes seen as processes inside an enterprise, between different enterprises, and between an enterprise and an end-consumer. A trend in electronic commerce is to make the business more cost effective and suited for electronic commerce by automating many of the required processes, both internally in an enterprise (EAI) and between different enterprises (B2B), ref. Figure 1. This procedure has today started in many big businesse areas, also among telecom operators like Telenor.

1.2 B2B

Third Parties are companies specialized in functions required by the seller or buyer to complete the trades, e.g. Security or Financial services, ref. Figure 2. In a B2B context both Buyer, Seller and Third Party are companies using internal processes to run the trade. In a telco context one such internal process can be the Billing System.

This again means that the B2B collaboration has to be integrated with the companies' internal processes. It is a very big issue in electronic business how to integrate B2B collaboration with back-end processes (internal systems).

Today the focus is as follows:

1 *B2B Standards*
Forums like RosettaNet, ebXML and OASIS are making common standards for B2B collaboration.

2 *Modeling Internal Processes*
The internal processes should be modeled such that they can be connected to B2B interfaces.

Forums like TeleManagementForum is doing a big job with their eTOM/ngOSS approach, and there exist different kinds of vendors delivering EAI technologies.

Web Services will be an excellent technology for EAI, but also in a B2B context.

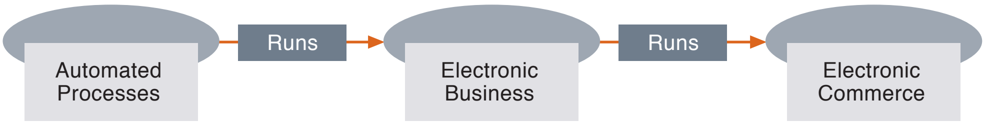


Figure 1 From automated processes to electronic commerce

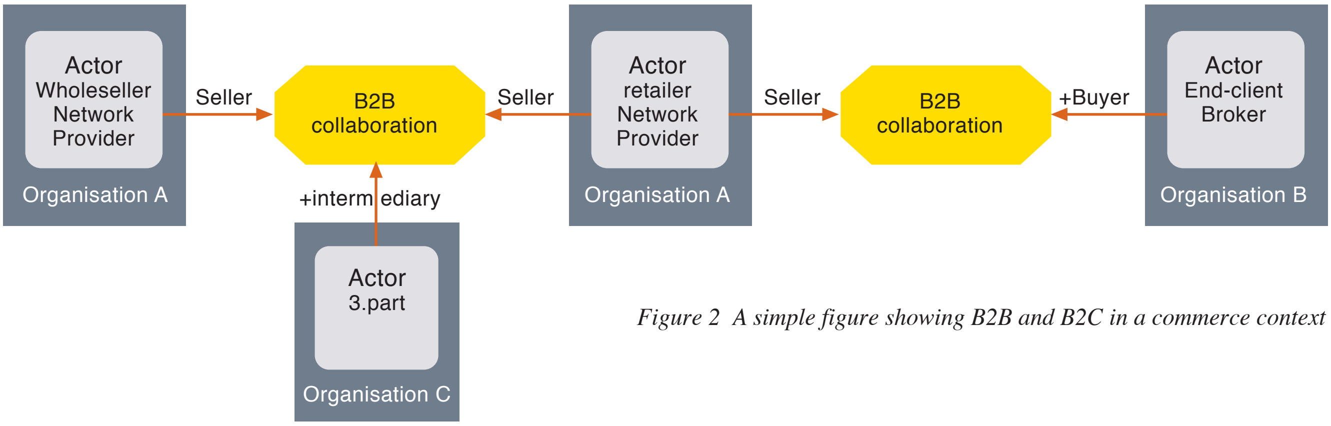


Figure 2 A simple figure showing B2B and B2C in a commerce context

1.3 Structure of the Document

After having listed out the different initiatives for enabling B2B Integration, the most complete of them, ebXML will be described.

2 eCommerce Standardization Initiatives

For organizations that conduct electronic businesses with partner organizations, it is important to see how Web Services would blend into providing support for XML vocabularies like ebXML, RosettaNet, cXML, etc. For Web Services to be used effectively for enabling B2B Integration, the Web Services platforms need to address common XML dialects, translate between dialects via XSL, security, compliance to contracts, etc.

2.1 ebXML

ebXML is an open e-business initiative started by UN/CEFACT and OASIS. Its aim is to make it easier for companies of all sizes and locations to conduct business on the Internet. The group is currently focusing on the specific needs of business-to-business (B2B) and Internet security as it relates to XML.

The specifications include a way to register and discover companies, business processes and related messages and content (registry and repository), a way to form trading partner agreements (collaborative partner agreement), and a messaging specification (ebXML Messaging Service).

ebXML may seem as an alternative to Web services, but it is more accurate to say that ebXML provides a necessary context or framework within which Web service technologies can be applied.

After the intellectual property rights to the SOAP specification were released in May 2001, ebXML Messaging Service was merged with SOAP to create a messaging protocol that uses the SOAP envelope but also adds ebXML specifications that cover areas left out in the SOAP specification. The CPP provides the same information as WSDL and more, such as the role of an organization in the context of a particular service, error-handling and failure scenarios. Likewise, information published on the ebXML Registry Service covers the same as that published on the UDDI registry, but more. Both systems can be used complementary, with the UDDI entry referring to Web services in the ebXML Registry.

The ebXML framework is designed in such a way that it is possible to exchange non-ebXML payloads within the framework. This because in the real world today there are different kinds of payloads in use. As part of the ebXML standards development process, a parallel stream known as the *Proof of Concept team* was established to implement ebXML specifications as they were emerging. It is therefore not only legitimate to exchange non-ebXML payloads within the ebXML framework, it is encouraged, ref. Figure 3.

The ebXML framework will incorporate the SOAP and SOAP Messaging with Attachment specifications into its upcoming releases. Building the messaging infrastructure of ebXML on top of SOAP will give SOAP another sign of industry-wide acceptance.

2.2 RosettaNet

RosettaNet is a vertical e-business framework for IT hardware and software vendors. The organization is set up by leading IT companies to define and implement a common set of standards for e-business. RosettaNet defines a common parts dictionary so that different companies can define the same product the same way. It also defines up to 100 e-business transaction processes and standardizes 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 has developed a structured four-part approach for creating what it calls Partner Interface Processes (PIPs).

1 Business Process Modeling examines common business procedures and defines the components of the processes.

2 Business Process Analysis analyzes the processes and defines a target list of desirable changes to the processes.

3 PIP Development establishes guidelines and documentation for the changes.

Dictionaries consist of two data dictionaries: a technical properties dictionary and a business properties dictionary. Along with the RosettaNet Implementation Framework (which defines an exchange protocol for PIP implementation), the dictionaries form the basis for PIP development.

RosettaNet's more than 40 members include Microsoft, Netscape, 3Com, Toshiba America, Compaq, CompUSA, Hewlett-Packard, IBM, and Intel. Its name refers to the Rosetta Stone, a stone on which Egyptian hieroglyphics were also written in other languages, making it possible to decipher the hieroglyphics. Rosetta stone has the more general meaning of "something that provides a key to understanding". The organization's slogan is "lingua franca for eBusiness". A lingua franca is a common second language, such as English for countries in the industrialized world whose first language is not English.

2.3 cXML

cXML is an open, versatile language designed for B2B e-commerce, to be used in e-business applications.

cXML transactions consist of documents which are simple text files containing values enclosed by predefined tags. Most types of cXML documents are analogous to hardcopy documents traditionally used in business. The most commonly used types of cXML documents are catalogs, punch outs and purchase orders.

From the start in February 1999, the cXML standard has been available for all to use. It leverages XML, and is supposedly extendable in itself. It is not yet compatible with the SOAP protocol.

2.4 xCBL and UBL

The XML Common Business Library is an XML component library for business-to-business e-commerce. It contains a collection of schemas defining common business processes such as: purchase orders, invoices, product descriptions, and shipping schedules.

xCBL, previously CBL, began as a research project at Veo Systems in 1997. CBL was developed to test the limits of XML for e-commerce and to identify requirements for XML design, development, and transaction tools and platforms. The first object-oriented XML schema language – SOX, the Schema for Object-Oriented XML – was a result of the lessons learned in the first version of CBL.

On October 17, 2001, OASIS announced the forming of the UBL technical committee. It was also stated that xCBL would be the starting point for the work. Specifically the UBL Charter states that the UBL Technical Committee will ... *"Begin with xCBL 3.0 as the starting point ... to develop the standard UBL library by mutually agreed-upon changes to xCBL 3.0 based on industry experience with other XML business libraries and with similar technologies such as Electronic Data Interchange"*. In many ways, the results of UBL are likely to be similar to xCBL but based on the input of many more individuals.

As UBL is starting with xCBL, it is likely to have many similarities with xCBL. This means that mappings from xCBL to UBL will probably be easier than from any other document standard.

3 The ebXML Framework

The ebXML initiative was an 18-month project, concluded in May 2001, to develop a set of specifications for electronic business interoperability. It is one of the most ambitious and important specification development efforts in its field in recent times.

Several hundreds of people have been actively involved in ebXML as contributors to the distributed multinational development teams that worked on the specifications, and several thousands of people were subscribed to one or more of the mailing lists through which the teams communicated and obtained feedback on their drafts.

Further work on ebXML is now taken over by OASIS and UN/CEFACT where OASIS is a not for profit member based organisation that identifies, builds and maintains industry-standard specifications for interoperability and UN/CEFACT is the United Nations Center for Trade Facilitations and Electronic Business.

The ebXML framework can be considered as evolutionary rather than revolutionary. It represents the state-of-art in e-business architecture, addresses the issue of integration at a high level, namely the level of public process interface, and supports the public process management application pattern.

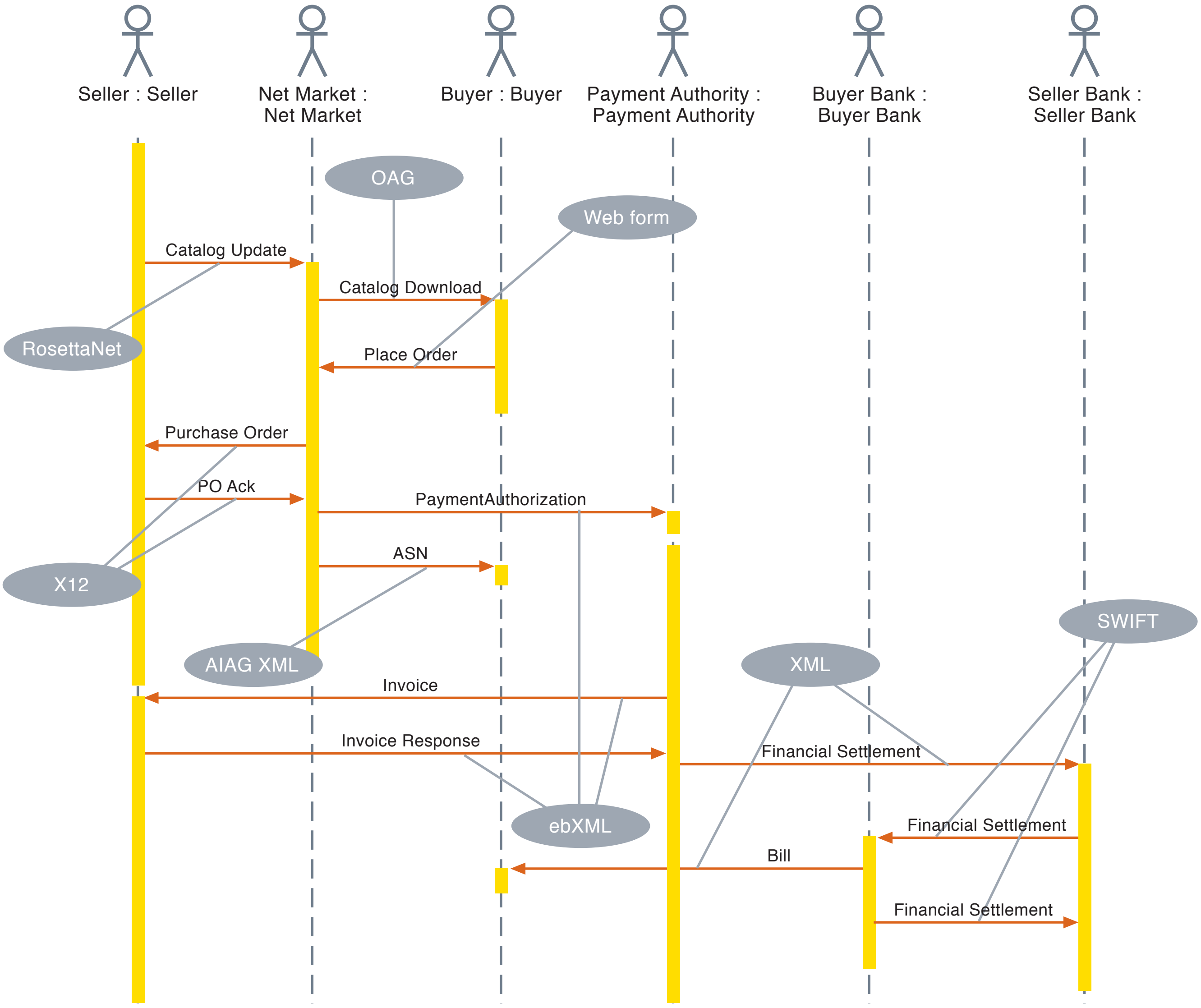


Figure 3 Partners involved in an ebXML compliant infrastructure exchanging non ebXML payload

The ebXML specifications are not a functional description of en e-business integration product. They are specifications that enable interoperability of software products (especially at the messaging level) and provide high-level systems configuration information (especially the protocol agreement and business process layer). This means that developers can use a variety of middleware products to implement an ebXML-compliant system.

3.1 ebXML Main Requirements.

The most important requirements ebXML framework tries to satisfy are:

- *Globalisation*; facilitating international trade, towards a single "global electronic market";
- *Interoperability*;
- *Security*; confidentiality, authenticity, integrity, non-repudiation.

For interoperability purposes, the introduction of the ebXML framework can be done gradually, because it makes possible the coexistence with other standards. It is of special importance to keep the payload interchanged between companies independent of the way the information is controlled. In other words, the ebXML framework does not mandate the use of ebXML messaging service for its payloads. The use of a flexible enveloping technique allows ebXML compliant messages to contain payload specified by legacy e-business systems employing traditional syntaxes like EDI FACT and SWIFT [1].

The sequence diagram in Figure 3 shows a scenario where several trading partners are engaged in a procurement process. All partners support an ebXML compliant infrastructure, but for bilateral collaboration, two partners use different types of message content.

3.2 ebXML Overall Architecture

The ebXML framework consists of seven main components:

1 Architecture
The ebXML archtecture can be used also for non ebXML payload.

2 Business Process specification Schema (BPSS)
BPSS is an XML based specification language that formally defines "public" business processes. The BPSS is strongly influenced by UMM, a modelling methodology developed by UN/CEFACT [2].

3 Core Components
Provides business information encoded in business documents that are exchanged between trading partners [4].

4 Registry/Repository
Specifies a general-purpose registry/repository for registration/storing company Business Profiles, and collaboration details to be used in electronic contracts and business transactions [5].

5 Collaboration Protocol Profiles and Agreement
XML documents that encode a party's e-business capabilities (CPP) or two partners' e-business agreements (CPA). They are closely related to BPSS [7].

6 Messaging Services (ebMS)
Provides an elegant general-purpose message mechanism using SOAP with attachment [8].

7 Security
Topic that is pervasive to all components and is critical for a production e-business system.

Figure 4 illustrates a scenario showing schematically how the ebXML architecture enables e-business between two parties: Company B is supposed to have already implemented the e-business according to ebXML architecture. Company A wants to do such kind of implementation to be able to conduct e-business with other companies like Company B.

3.3 Relationship Between the Different Components

The different components listed above are relevant in the different steps. Figure 5 shows the relationship between them.

- Step 1: Global information: Definition, specification, design, registration of the common model for business process and the common semantic for the data to be exchanged.

- Step 2: Each party: design, specification, registration of own business.

- Step 3: Agreement between two partners: discover each other, make agreement, prepare collaboration, configuration of e-business system.

- Step 4: Engage bilateral collaboration between partners.

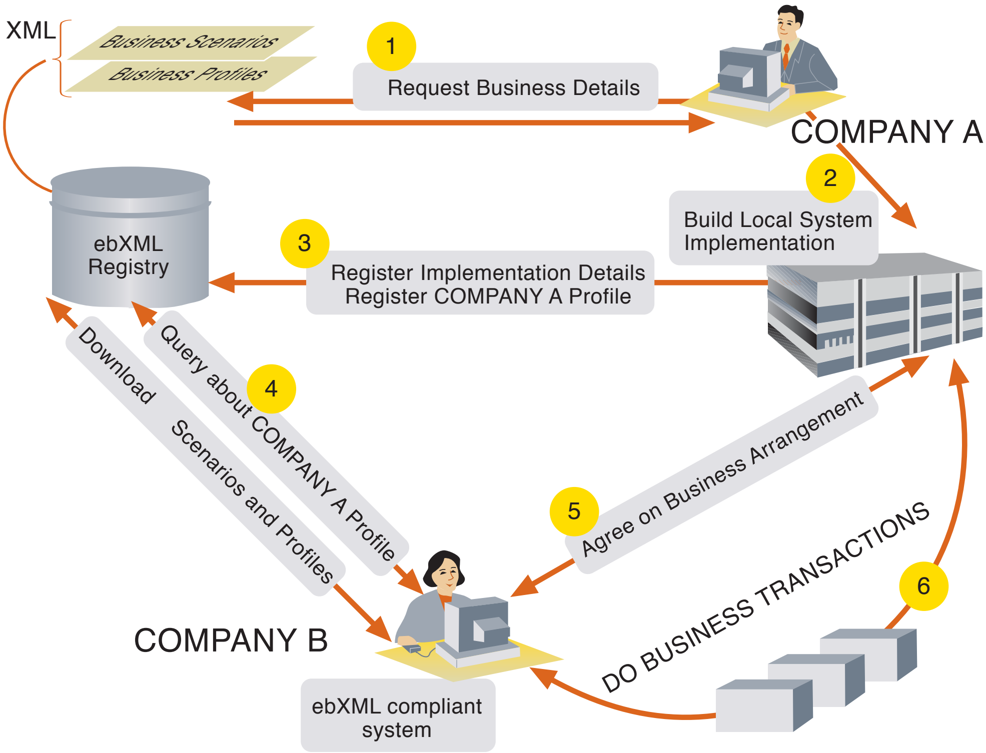


Figure 4 High level overview of the interaction of two companies conducting e-business using ebXML

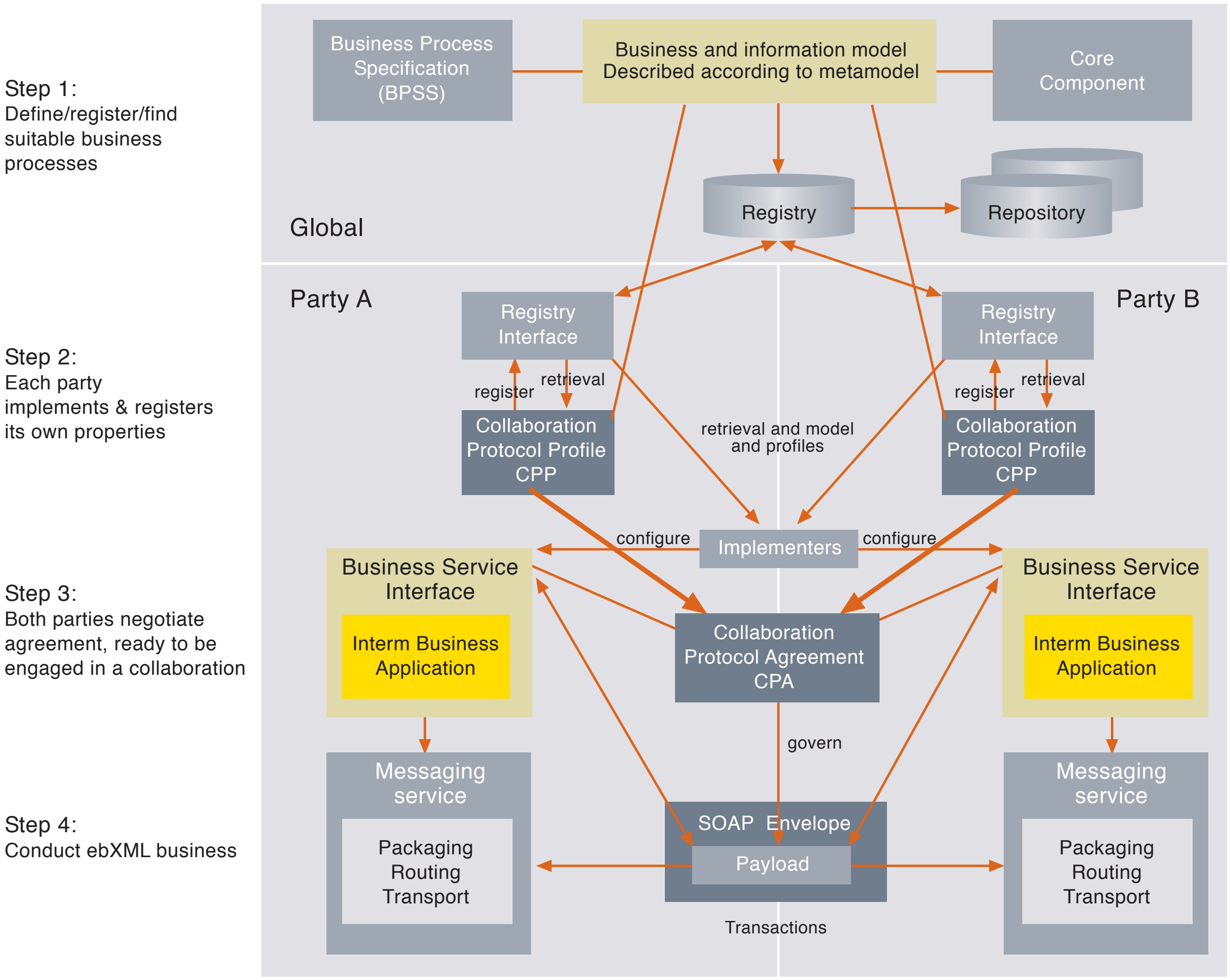

Step 1:
Define/register/find suitable business processes
Step 2:
Each party implements & registers its own properties
Step 3:
Both parties negotiate agreement, ready to be engaged in a collaboration
Step 4:
Conduct ebXML business
Global
Business Process Specification (BPSS)
Business and information model Described according to metamodel
Core Component
Registry
Repository
Party A
Party B
Registry Interface
register
retrieval
Collaboration Protocol Profile CPP
retrieval and model and profiles
Registry Interface
register
retrieval
Collaboration Protocol Profile CPP
configure
Implementers
configure
Business Service Interface
Interm Business Application
Collaboration Protocol Agreement CPA
Business Service Interface
Interm Business Application
govern
Messaging service
Packaging Routing Transport
SOAP Envelope
Payload
Transactions
Messaging service
Packaging Routing Transport

Figure 5 The different ebXML components involved in the different steps

3.4 The ebXML Business Process Specification Schema (BPSS)

The BPSS is one of the most innovative sections of ebXML Specifications [2]. At a high level, a BPSS instance specifies all the business messages that are exchanged between two business partner roles, their content, and their precise sequence and timing. As such it is the direct link between the business analysts or subject matter experts that define the business processes, and the implementers that use these specifications either to configure their ebXML infrastructure, or write the appropriate code to enforce all aspects of the business process.

The relationship between the UMM metamodel and the *ebXML BusinessProcess Specification Schema* is shown in Figure 6.

Using the UMM methodology, and drawing on content from the UMM Business Library a user may create complete Business Process and Information Model conforming to the UMM metamodel. Since the ebXML *Business Process Specification Schema* is a semantic subset of the UMM metamodel, the user may then in an automated fashion extract from the Business Process and Information Model the required set of elements and relationships, and transform them into an ebXML Business Process Specification conforming to the *ebXML Business Process Specification Schema*.

The architecture of the ebXML Business Process Specification Schema consists of the following functional components:

- UML version of the *Business Process Specification Schema*;
- XML version of the *Business Process Specification Schema*;
- Production Rules defining the mapping from the UML version of the *Business Process Specification Schema* to the XML version;
- Business Signal Definitions.

Together these components allow you to fully specify all the run time aspects of a business process model.

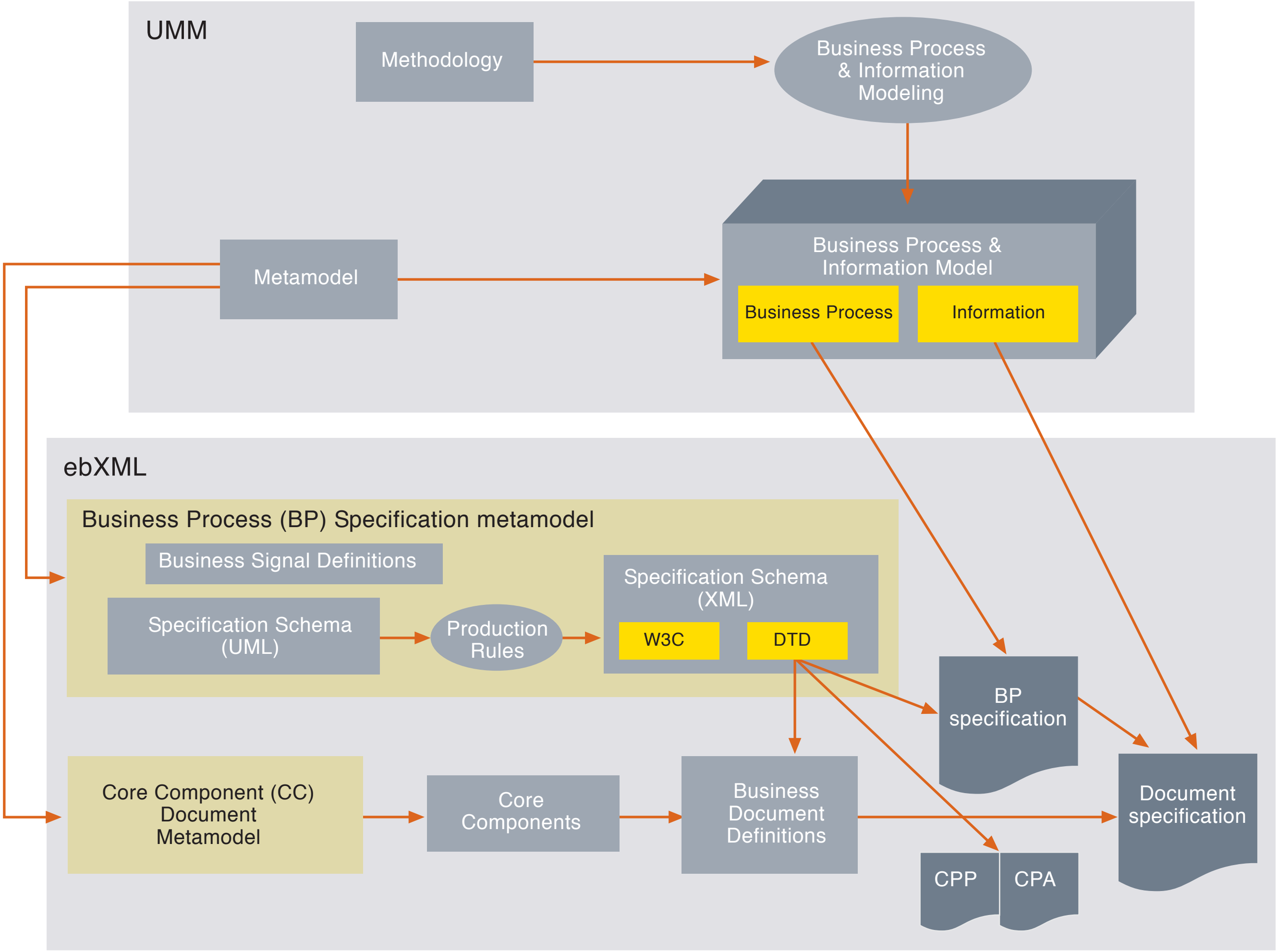

UMM
Methodology
Business Process & Information Modeling
Metamodel
Business Process & Information Model
Business Process
Information
ebXML
Business Process (BP) Specification metamodel
Business Signal Definitions
Specification Schema (UML)
Production Rules
Specification Schema (XML)
W3C
DTD
BP specification
Core Component (CC) Document Metamodel
Core Components
Business Document Definitions
Document specification
CPP
CPA

Figure 6 Relationship between UMM Metamodel and ebXML Business Processes/ Core Components

The BPSS metamodel allows the description of business processes as distributed processing between loosely coupled systems. Figure 7 illustrates the relationship between the different components of the metamodel as a solution to integrate business partners. The BPSS allows the description of the collaboration between two partners in a generic way: the *binary collaboration* consists of:

- *Activities* involved include:
  - *Transaction activity (T activity)*: conducting a Business Transaction (BT)
  - *Collaboration activity (C activity)* (re-using of activities: recursive property)
- *Business Transactions (BT)* between roles defining the exchange of document (doc)
- *Roles*: initiating role, responder role
- *Collaboration activity*: choreography of activities. It defines a "public" business process which has to be mapped with private processes of the corresponding partners.

It should be noted that the concept of multiparty collaboration is covered by ebXML, but it is defined as a synthesis of binary collaborations.

The ebXML *Business Process Specification Schema* provides a standard framework for business process specification. As such, it works with the ebXML Collaboration Protocol Profile (CPP) and Collaboration Protocol Agreement (CPA) specifications to bridge the gap between Business Process Modeling and the configuration of ebXML compliant e-commerce software, e.g. an ebXML Business Service Interface, as depicted in Figure 5.

3.5 Core Components

ebXML defines a core component as a "building block that contains pieces of business information, which go together because they are about a single concept" [4]. Core components are *reusable pieces of business information, horizontal to many business processes*. Examples of core components could be things like "Business Party Details" or "Data of Purchase Order".

The reusability is achieve by taking in account two main issues:

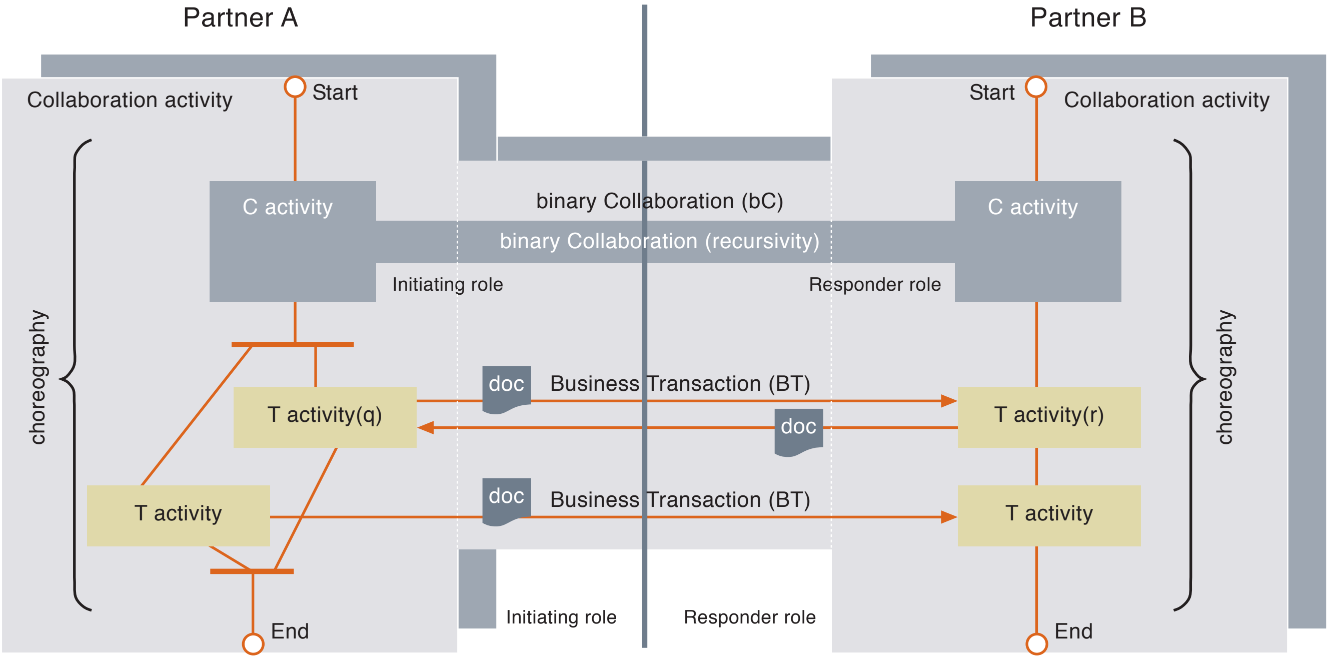


Figure 7 BPSS describing Business processes

- *The context:* CC is context free: A core component can be used across several business sectors.
- *The domain:* CC is domain related: a core component can be re-used by another industry area if it is found to be appropriate for their use.

The context gives the relationship between Core Components (CC) and Business Information Entities (BIE). The context is related to / derived from the Business Process.

Figure 8 illustrates schematically how core components can be built into business documents.

A *Core Component (CC)* is a building block for the creation of a semantically correct and meaningful business information exchange 'parcel', containing the information pieces needed to describe a specific concept. There are three categories of *Core Components*:

- Basic Core Component
- Core Component Type
- Aggregate Core Component

A *Business Information Entity (BIE)* is a piece of business data or a group of pieces of business data with a unique business semantic definition. A Business Information Entity can be either a *Basic Business Information Entity* (BBIE) or an *Aggregate Business Information Entity* (ABIE). A Basic Business Information Entity is based on a *Basic Core Component* (BCC). An Aggregate Business Information Entity is a re-use of an *Aggregate Core Component* (ACC) in a specified business context.

Figure 9 describes the *Business Information Entity* types and shows relationships to the *Core Component* counterparts.

A *Naming Convention* is necessary to gain consistency in the naming and defining of all *Core Components* and *Business Information Entities*. The resulting consistency facilitates comparison during the discovery and analysis process, and precludes ambiguity, such as the creation of multiple *Core Components* with different names that have the same semantic meaning.

Figure 8 Business document structure

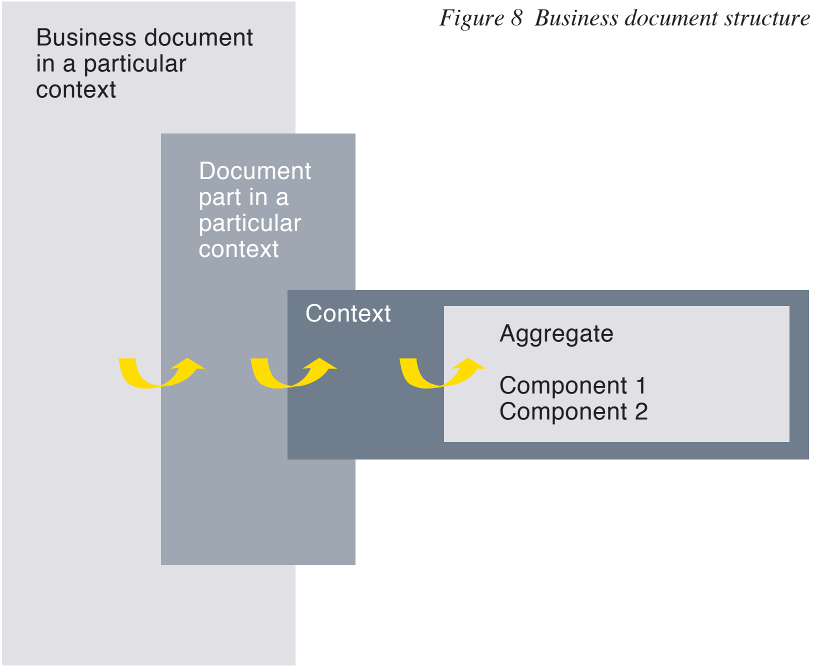
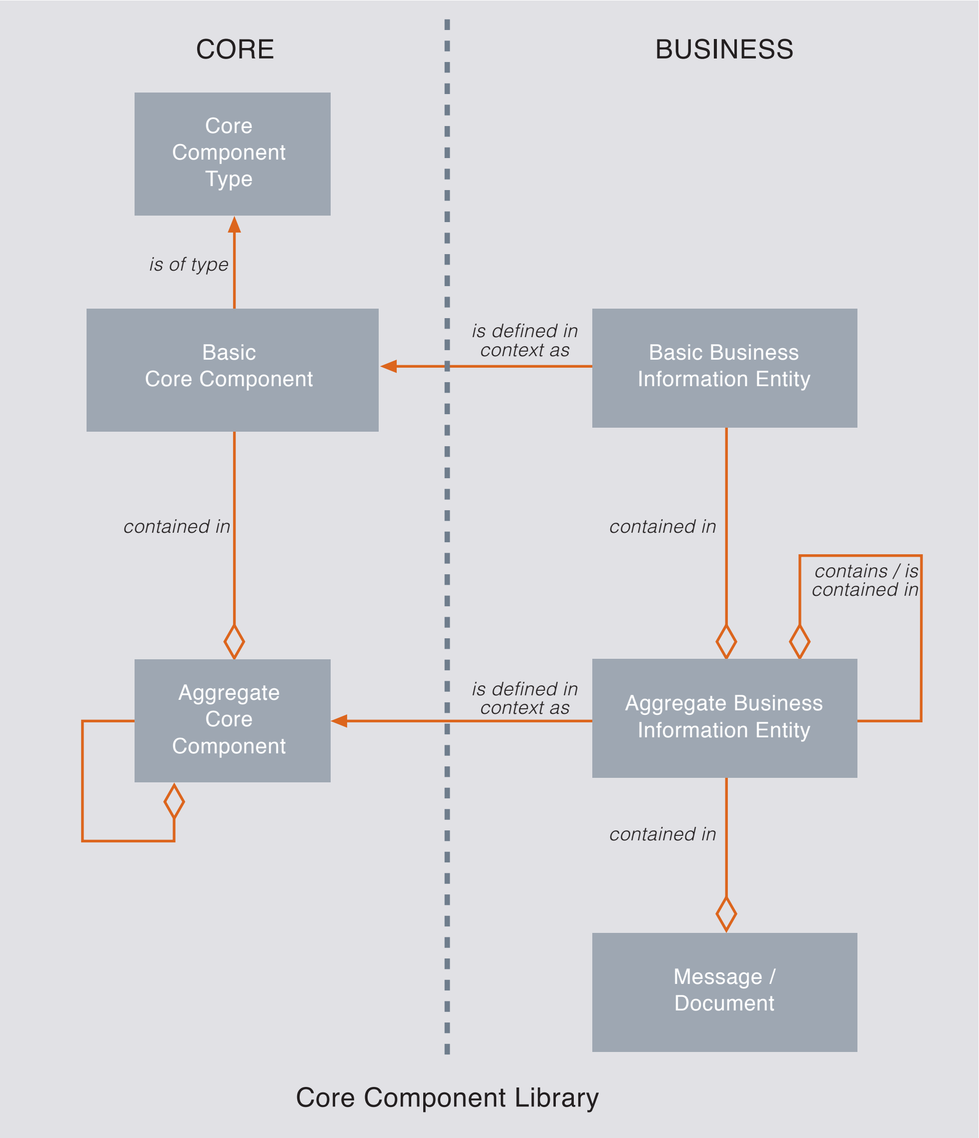

Business document in a particular context
Document part in a particular context
Context
Aggregate
Component 1
Component 2

Figure 9 Relationships between Core Components and Business Information Entities



Catalog of Core Components and Common Business Processes

The catalog of *Common Business Processes* (CBP) [3] is useful for discovery and analysis of core components that will be used as the building blocks for deriving business documents within a given context. This can be done by checking all sources of documents listed and cross-referenced on the Common Business Process Catalog to identify a document that may have the information needed.

Table 1 shows an example of how Common Business Process “Manage Purchase Order” is shown in the Catalogue and its relation to other standards.

The Common Business Process is structured as a cross reference table with the following components:

Table 1 Example of CBP cross reference table

Common business process	Normative category	Normative sub-category	EDIFACT	X.12	Rossetta-Net PIP	C11	OAG BOD
Manage Purchase Order (Create/Change/ Cancel and Accept PO)	Procurement Management	Procurement	ORDCHG, ORDERS, ORDRSP	860, 852, 850, 865, 875, 876	PIP3A4	0410, 0411	004_Acknowledge_PO_005 056_Add_PO_005 058_Cancel_PO_005 057_Change_PO_004 010_Get_PO_005 054_Getlist_PO_004 055_List_PO_004

Common Business Processes
A business process describes in detail how trading partners take on roles, relationships and responsibilities to facilitate exchange of information. Common Business processes are identified as commonly used across various organizations, industries or other business entities.

Normative Category
Built from components of a Porter Value Chain, see Figure 10.

Normative Sub-Category
Decomposition of the Porter Component into logical sub groups.

EDIFACT/X12/CII/OAG BOD/xCBL
Common industry standards used as a cross-reference, by identifying their specific equivalent business documents commonly used today.

RosettaNet PIP
Common business processes cross-referenced to business transactions as specified by RosettaNet Partner Interface Processes™ (PIPs™) which define business processes between trading partners.

3.6 Registry/Repository

The ebXML spefication's goals are that of creating a platform independent open registry/repository for housing the description and facilitating the exchange of business artifacts, and discovering business via collaboration profiles. The registry contains the descriptions of these business artifacts (like the index of a book), and the repository actually stores them (like the book's actual content).

The Registry Information Model (ebRIM) provides information on the type of metadata that is stored in the Registry as well as the relationships among metadata. The relationships are industry ontologies providing a structured coded vocabulary, making a significant enhancement to current glossary-based mechanisms.

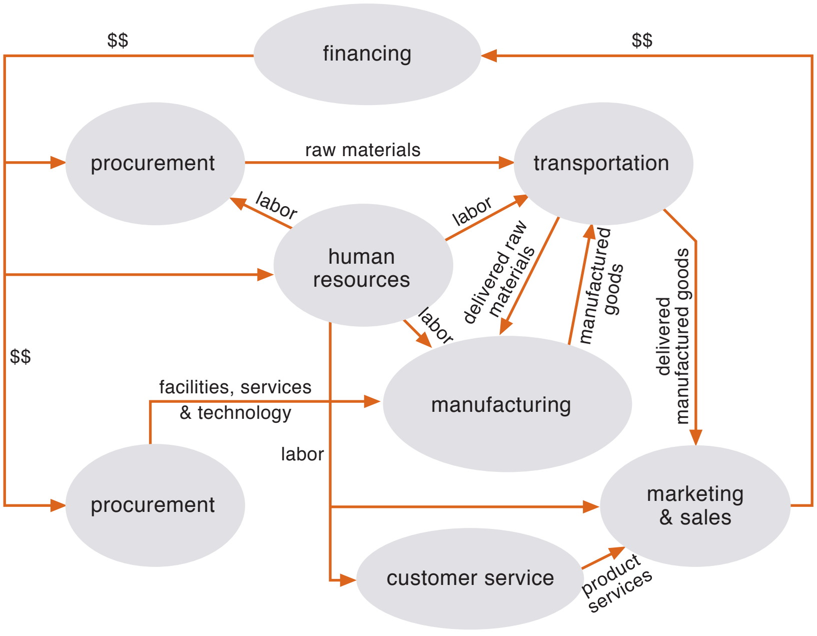


Figure 10 Graphical representation of the Porter Value Chain

3.6.1 Registry Information Model (RIM)

The ebXML Registry provides a key to new functionality using the RIM information tree, which can be navigated based on business domains and required business functions. Note that the information model is not modeling actual Repository items [6].

A short meaning of the main classes is given in Table 2.

Figure 11 summarizes the structure of the Registry and the relationship with the Repository.

3.6.2 Registry Architecture

The ebXML Registry architecture consists of an ebXML Registry and ebXML Registry Clients [5]. The Registry Client interfaces may be local to the registry or local to the user. Figure 12 shows the relationship between the different interfaces.

There are several possible topologies supported by the registry architecture with respect to the Registry and the Registry Client as shown in Figure 13.

Table 2 Main classes in the RIM

```
RegistryObject                Top Class
    User
    AuditableEvent            log all event (client initiated request) as instance
    RegistryEntry             Provide an access to all instance in the registry
        ExtrinsicObject       Describes content whose type is not known to Registry
        IntrinsicObject       Describes content whose type is known to Registry
            Package           Gouping in logical entities
            Organization      Info about submitting organisation
            Classification    Define tre structure
            ExternalLink      Define all what is not content in the registry
```

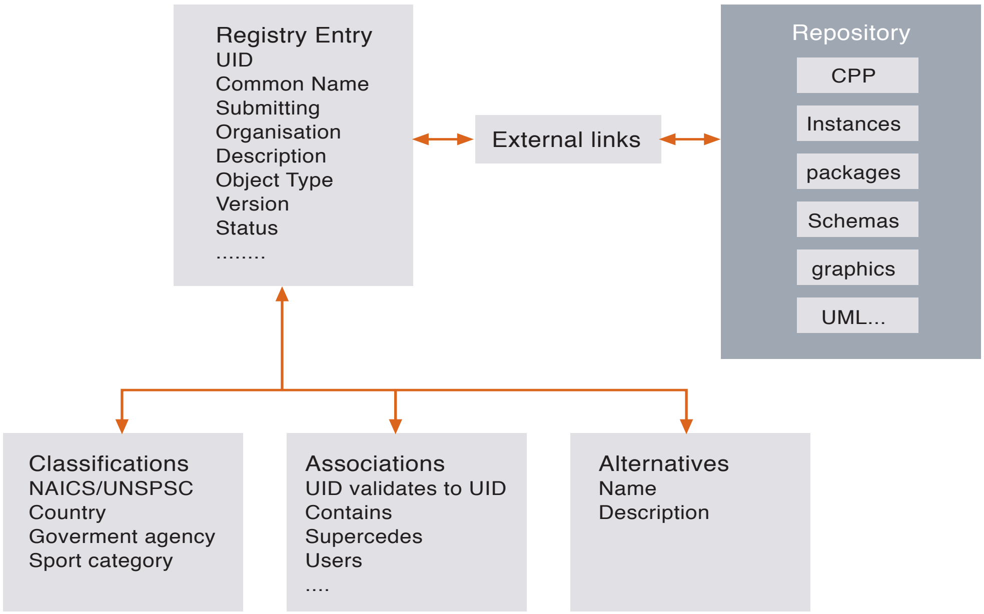


Figure 11 Overview over exXML Registry/Repository

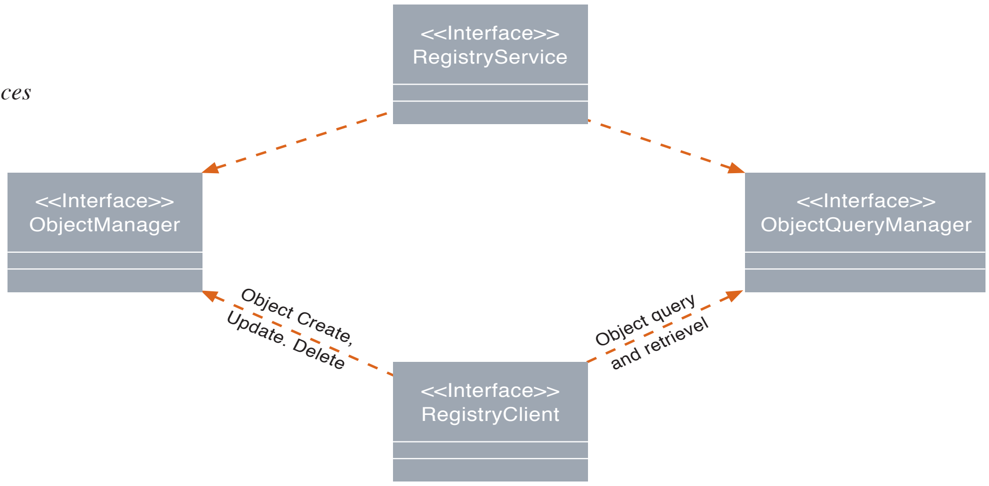
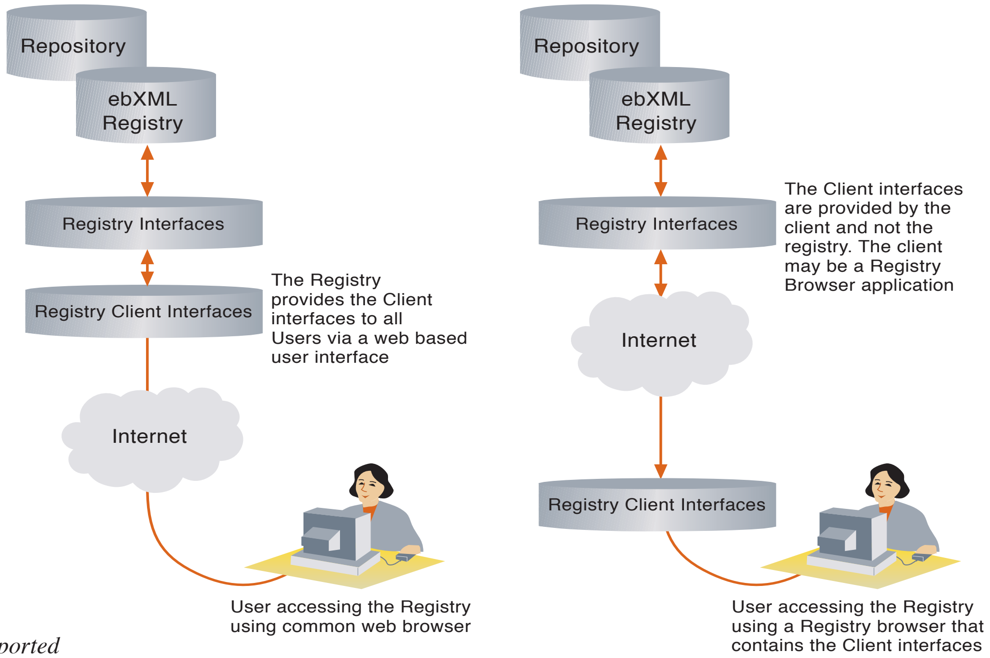


Figure 12 ebXML Registry Interfaces



The Registry provides the Client interfaces to all Users via a web based user interface

User accessing the Registry using common web browser

The Client interfaces are provided by the client and not the registry. The client may be a Registry Browser application

User accessing the Registry using a Registry browser that contains the Client interfaces

Figure 13 Different topologies supported

Conformance as an ebXML Registry
An implementation conforms to this specification as an ebXML registry if it meets the following conditions:

1 Conforms to the *ebXML Registry Information Model [ebRIM]*.

2 Supports the syntax and semantics of the Registry Interfaces and Security Model.

3 Supports the defined ebXML Registry DTD.

4 Optionally supports the syntax and semantics of Section 8.3, SQL Query Support.

Conformance as an ebXML Registry Client
An implementation conforms to this specification as an ebXML Registry Client if it meets the following conditions:

1 Supports the ebXML CPA and bootstrapping process.

2 Supports the syntax and the semantics of the Registry Client Interfaces.

3 Supports the defined ebXML Error Message DTD.

4 Supports the defined ebXML Registry DTD.

3.7 Collaboration Protocol Profiles and Agreement

The Collaboration Protocol Profiles/Collaboration Protocol Agreement (CPP/CPA) specification [7] is the implementation of trading partner agreements in the ebXML framework. The term trading partner agreement (TPA) is a general term that can cover both technical and business related agreements, or a combination of boths. The ebXML, collaboration protocol agreements (CPA), are the machine interpretable versions of such TPAs. One can think of such a CPA as the bridge between the transport (messaging) layer and the business layer.

Figure 14 illustrates schematically how the Collaboration-Protocol Profiles (CPP) are built.

The scenario in Figure 15 gives an overview of the Collaboration Protocol Agreements (CPA) and how they are built from CPPs.

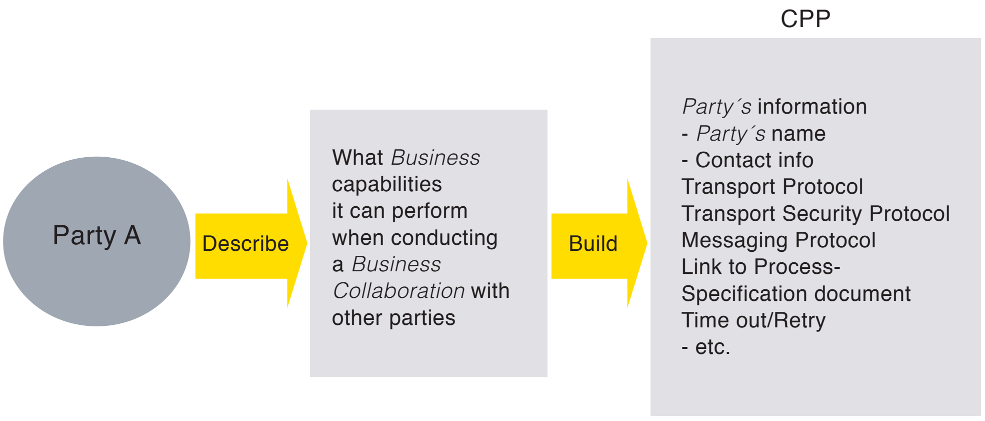

CPP
Party A
Describe
What *Business* capabilities it can perform when conducting a *Business Collaboration* with other parties
Build
Party´s information
- *Party´s* name
- Contact info
Transport Protocol
Transport Security Protocol
Messaging Protocol
Link to Process-Specification document
Time out/Retry
- etc.

Figure 14 Overview of the CPP

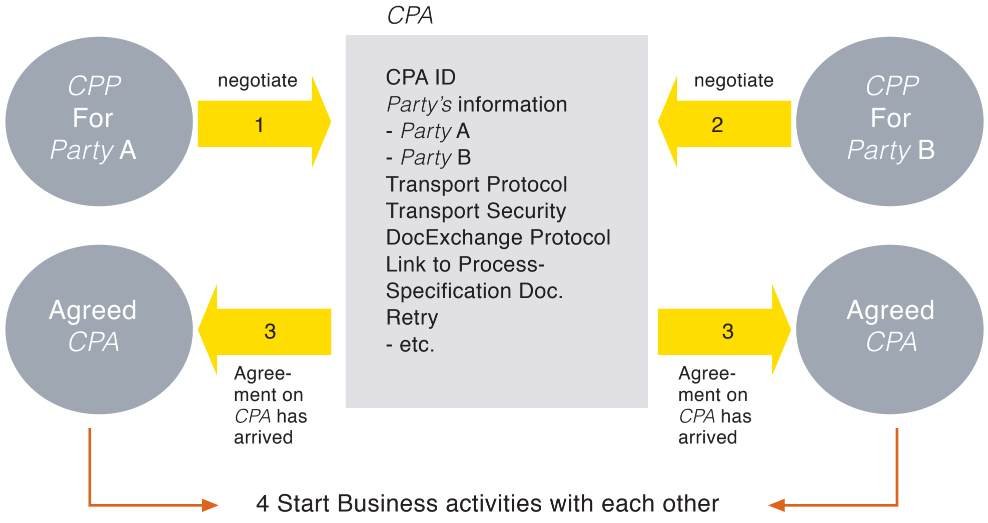


Figure 15 Overview of building of CPA

Figure 16 The ebMS functionality

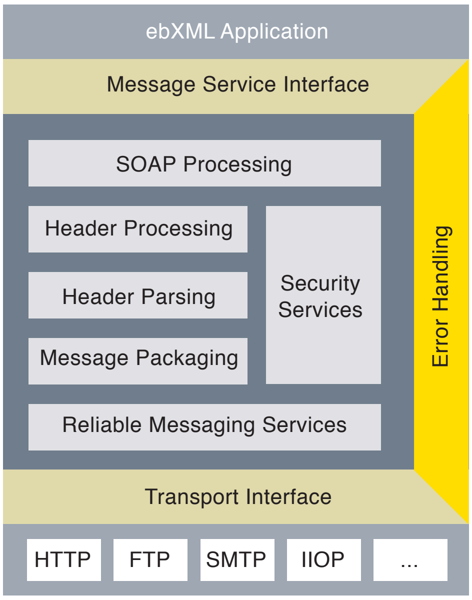
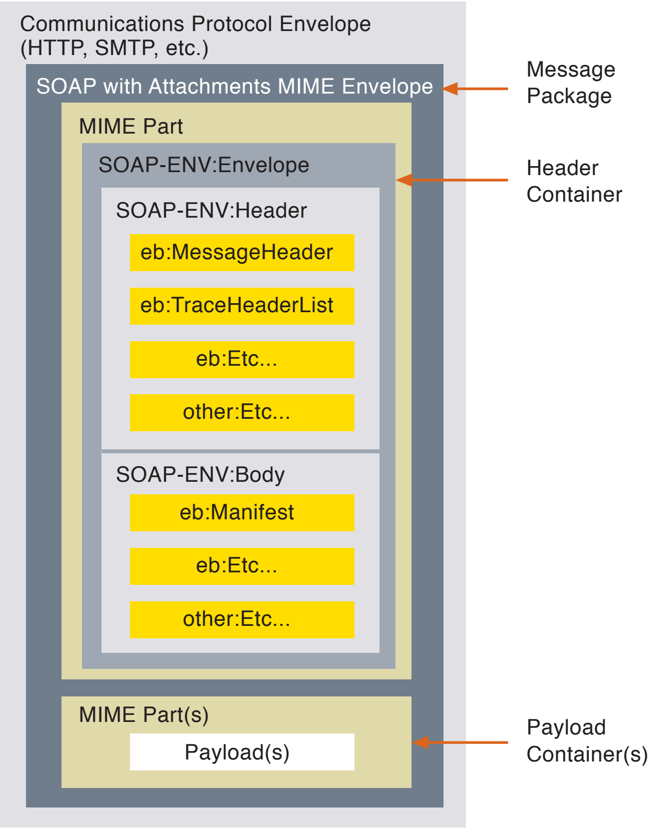


Figure 17 Layout of the ebXML message



The CPPs and CPA together with Process Specification are registred in the ebXML registry.

3.8 Messaging Services (ebMS)

ebMS, specified by ebXML OASIS, is a standardized Messaging Service which enables an interoperable, secure and reliable exchange of messages between two parties [8].

ebXML compliant software can be used to implement eBusiness scenarios where two or more partners are engaged in binary/multi-Party Business Process and run the associated electronic exchanges.

The document-exchange layer accepts a Business document from the Process Specification layer at one Party and passes it to the transport layer for transmission to the other Party. It performs the inverse steps for received Messages.

The TRP gives the specification to the *Messaging Service* which comprises:

1 *Message Service interface* defines the operations that local object needs to perform.

2 *Message Service Handler (MSH)* for interpretation of the ebXML messages.

3 *Transport service interface* for Adaptation/ mapping to the transport layer.

4 The *error handling* for the report of errors occurring during the processing of a message.

Figure 16 shows the ebMS functionality in more detail.

Against the “application layer”: This is done by using SOAP (Simple Object Access Protocol) which allows access to services, objects, and servers in a platform-independent manner and facilitates interoperability by bridging competing technologies in a standard way.

Against the “transport layer”: A transport interface allows the use of different protocols like HTTP, SMTP, IIOP.

The ebXML message exchanged between partners is considered as an attached “SOAP envelope”. The way of attachment depends on the communication protocol chosen. Figure 17 illustrates how the SOAP envelopes the ebXML message.

3.9 Security

When working with B2B frameworks such as ebXML [1], we could agree that there is a fundamental conflict between achieving the level of openness required to deal with a (potentially large) number of parties, and achieving the appropriate level of security. A characteristic of B2B initiative is that they try to use Internet – a global communication infrastructure – as a global trading infrastructure. While this is definitely a very challenging endevor in terms of size, scope and nature, history has shown that it is possible to successfully open up new trade routes and find acceptable ways to deal with the associated risks. ebXML will prove itself to be instrumental in opening up these new digital trade routes; but in doing so, it is necessary to implement a robust security strategy to protect the business involved.

In an ebXML environment it is not just an individual element that needs to be secured; the whole business process defines the security needs and the first step in solving security problems is to analyze the risks; from this analysis, it is possible to determine what to secure and what not to secure. This characterises the approach to security taken in ebXML. There is no attemp to secure the whole business process 100 % – that simply is not feasible; instead security is based on mutual understanding and agreements, coming up with an acceptable risk and adjusting the security measures accordingly.

As shown in Figure 18 the Collaboration Protocol Profile (CPP) contains the representation of agreed security policy. It is created as a result of mapping security policies and collaboration parameters onto the business process definition.

Thus the CPP contains the set of possible and required security measures from the own company's point of view, after mapping to the business process. It is the first step towards a full policy-based security system

A key role in implementing security within the framework is the use of Trusted Third Parties (TTP). Security is often necessarily fully integrated into internal processes to be most effective, however in the B2B realm, a clearer intersection domain of shared governance is needed to enable security agreements to be reached between trading partners. This issue is often ignored because of the way security agreements between trading partners have been developed historically, with confidentiality between parties. These agreements may often be reached on an *ad hoc* basis and sometimes dictated by dominant market players. TTPs need to be implemented to overcome the poor definition of security generally in the B2B eCommerce environment and will be essential for fully automating supply chains.

The following TTPs is seen as the primary security service providers for the near future:

- Identity Management
- PKI Trust Service Providers
- Trusted Time Stamp and Repository
- Trading Community Registry and Repository
- Outsourced OSS Component SLA manager
- Third Party Security Audit
- Fraud Management Services
- Aggregated Trust Service Providers (for efficiently combining security services)

4 Differences Between ebXML and WS

4.1 The Web Services stack

In order to describe how Web Service standards relate to the above features it is useful to begin by looking at a representative Web Services architecture.

Web Services architecture [9] is built from layers of technology and standards on which services can be implemented and deployed. Each layer on this Web Services stack depends on the layers below it. There are many variations of this architecture, but each variation generally

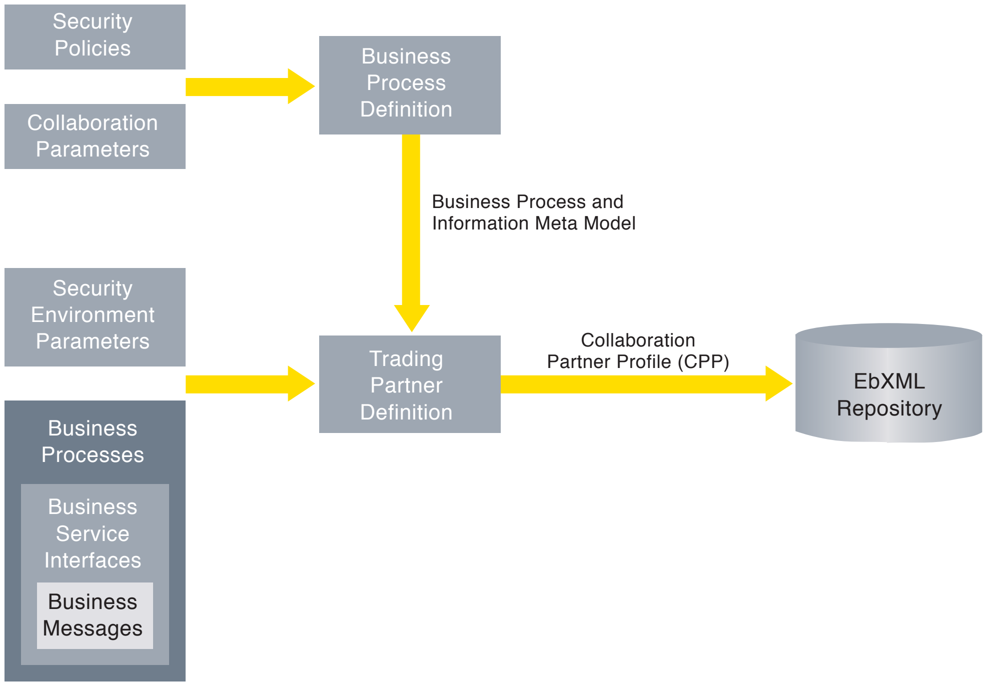

Security Policies
Collaboration Parameters
Business Process Definition
Business Process and Information Meta Model
Security Environment Parameters
Trading Partner Definition
Collaboration Partner Profile (CPP)
EbXML Repository
Business Processes
Business Service Interfaces
Business Messages

Figure 18 Security in CPP

includes the features described in the previous section above the basic messaging and service description foundation layers.

The diagram in Figure 19 illustrates a generic Web Services architecture and how it maps to specific architectures from prominent organizations or companies.

Figure 20 is an attempt at illustrating the similitude and the difference between WS reference and ebXML. EbXML is broadening the traditional Web Service view and can be regarded as the Web Services for Electronic Business.

5 Towards an e2-OSS Framework for the Telecom Industry

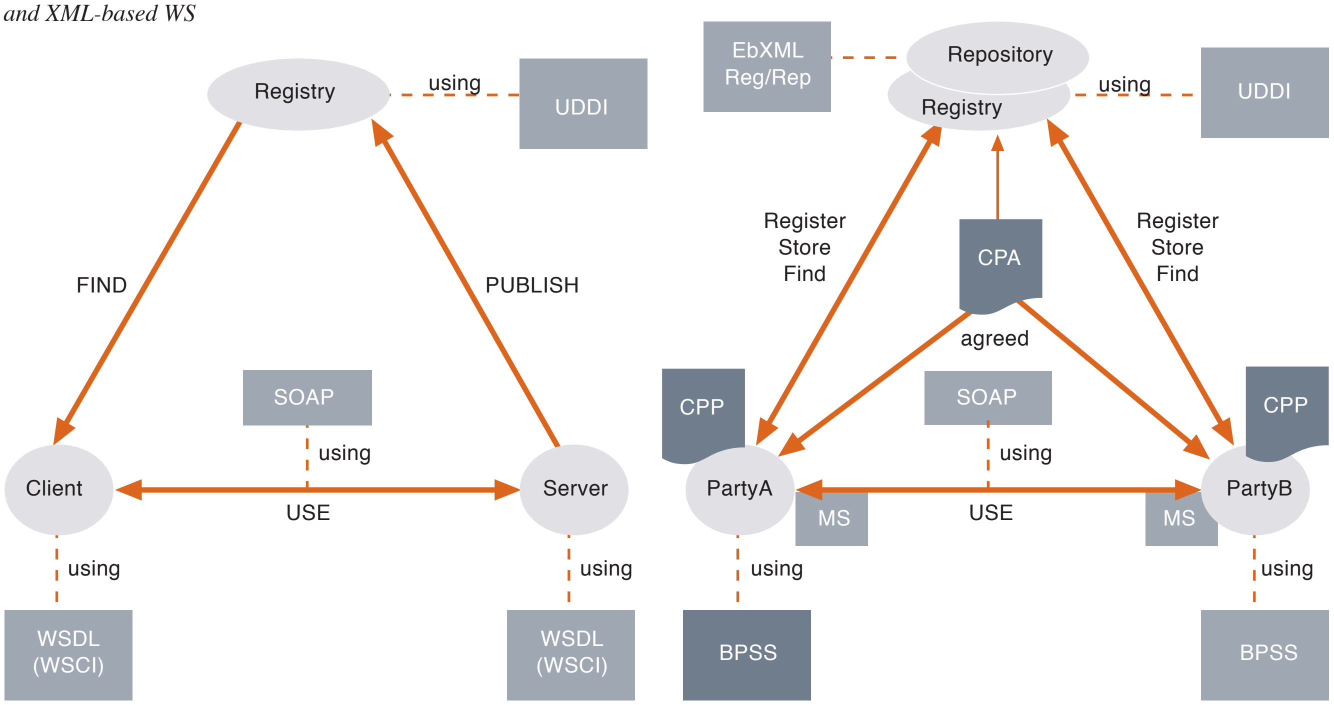
The EURESCOM project P1106 [10] has tried to identify the impact the emerging Internet standards like ebXML have on the way to do business in the telecom industries.

The main purpose of the e2-OSS Framework was to define an "e-enabled" B2B context specific for telecommunications industry. This should cover at least Business, Functional, Technology, Security aspects/dimensions and would allow for a B2B supply chain integration.

Figure 19 The different layers and the corresponding standards

Generic Stack	ebXML	IBM	Microsoft	BPMI
Agreements	CPA	TPA	???	
Orchestration	BPSS	WSFL	XLANG	
Quality of Service		WSEL	???	BPML
Service		WSDL	WSDL	
Packaging/ Transport	MSH (over SOAP)	SOAP D-SIG HTTP-R	SOAP WS-Routing WS-Security	

Figure 20 Similitude and differences between ebXML and XML-based WS



During the project it has become clear that the general B2B standards framework as outlined by Derek Coleman of RosettaNet is appropriate for e-Business Integration standards for communications suppliers. The notions of where B2B integrations relate to Enterprise integrations from a requirements perspective was also clear.

However, two major issues hampered a more rigorous analysis of how B2B and Enterprise Integration could be aligned for the e2-OSS framework.

- Lack of content standards for the telecoms vertical document and process B2B standards with which to work.

- Lack of understanding of appropriate patterns for integration within the Functional Service View of any implementation.

The project has identified where guidelines are needed for the further development of content for the e2-OSS framework and several learning points have been gained about the style and approach to design B2B collaborations using the transaction patterns outlined in UMM.

Many of the defined PIPs available in RosettaNet form a useful basis for B2B patterns for transactions to support telco processes for fulfilment and billing. However, RosettaNet's rigid definition of content standards defined for the computing and components industry renders them inappropriate for direct use. The alternative is to re-use the PIP blueprints with alternative content standards within an ebXML BPSS Collaboration.

It is clear that additional transactions are required for communications supply chain integrations and a full Business Operations Map is required as a framework within which this should done.

The project had the intention of developing a communications service provider value chain operations map that is complementary to the TMF eTOM. Early attempts at this have yet to reveal what the structure for such a map should be and this work remains to be done.

A large part of the project focussed on process modelling and the relationship of processes to components. This activity covered business processes, description languages, the use of a messaging service for B2B and the use of component services to achieve backend integration to OSS. However, a significant part of the framework relates to content and document standards.

An assumption throughout the project is that B2B integration will be achieved by the exchange of XML documents. Most likely these documents would be defined using the W3C schema definition language XSD. The issue however is the source of Business Document Definitions and Business and Technical Dictionaries.

Names for selected business documents have been suggested as a consequence of identifying Business Transactions (e.g. Trouble Ticket). However no attempt has been made to define their content.

There are potentially several sources and approaches to the creation of B2B business documents standards. The two major questions are:

1 Use statically defined business document definitions or assemble B2B document instances from components;

2 Purposely build B2B standards or re-use and extend Enterprise standards.

Table 3 outlines four possible sources of standards, which typify the approach.

The ebXML group has not yet clearly defined what it regards as an appropriate strategy for defining vertical technical content. Currently it is suggested that B2B Business Documents be based on components derived from a global tag library as proposed to the ITU in the tML initiative. Furthermore, these documents should be derived from the same components as used for Enterprise Business Documents.

This would suggest the need for a set of OSS-J Enterprise Business Document Components using the tags defined within the proposed tML tag library structure.

	Statically Defined Documents	**Dynamically Assembled**
Public Standards	RosettaNet DTDs	ebTWG Core Components
Extension of Private Standards	OAGIS BODS encapsulated in B2B document envelope	OSS-J XML value type API encapsulated in B2B document envelope

Table 3 Content Standards

In the short term (pending agreement of a telecom specific library of business document components) the only practicable route forward for communications service providers is to use and extend published "horizontal" standards such as xCBL or OAGIS.

6 References

1 *Professional ebXML Foundations*. Wros Press, 2001.

2 UN/CEFACT & OASIS. *EbXML Business Process Specification Schema v.1.0*. (Technical report) December 4, 2002 [online] – URL: http://www.ebxml.org/specs/ebBPSS.pdf

3 UN/CEFACT & OASIS. *Catalog of Common Business Processes v.1.0*. (Technical report) December 4, 2002 [online] – URL: http://www.ebxml.org/specs/bpPROC.pdf

4 UN/CEFACT. *Core Component Technical Specification v.1.7*. 21 Oct 2001.

5 UN/CEFACT & OASIS. *Registry Service Specification v.1.0*. (Project team Technical report) December 4, 2002 [online] – URL: http://www.ebxml.org/specs/ebRS.pdf

6 UN/CEFACT & OASIS. *ebXML Registry Information Model v.1.0*. (Technical report) December 4, 2002 [online] – URL: http://www.ebxml.org/specs/ebRIM.pdf

7 UN/CEFACT & OASIS. *Collaboration Protocol Profile and Agreement Specification v.1.0*. (Technical report) December 4, 2002 [online] – URL: http://www.ebxml.org/specs/ebCPP.pdf

8 UN/CEFACT & OASIS. *Messaging Service Specification v.1.0*. (Technical report) December 4, 2002 [online] – URL: http://www.ebxml.org/specs/ebMS.pdf

9 O'Riordan, D. *Business Process Standards for Web Services*. Tect.

10 *EURESCOM project P1106 Deliverable 3*. Aug & Sep 2002. (EDIN 0328-1106 & EDIN 0329-1106.)

Electronic Gateways – Forging the Links in Communications Services Value Chains *)

DERRICK EVANS, DAVE MILHAM,
ELAYNE O'SULLIVAN AND MARTIN ROBERTS

Derrick Evans (42) graduated in Physics from Imperial College of Science and Technology in 1981. He joined BT Laboratories that year and has since worked on a variety of OSS software development and solutions integrations projects. Currently he is leading a team of consultants specialising in OSS solutions design and delivery including consulting on business-to-business interface specification for telecommunications services management.

Derrick.Evans@bt.com

Dave Milham (54) is a graduate of Electrical Engineering from Imperial College of Science and Technology and in telecommunications from Essex University. He works on technical strategy in the BTexact technologies OSS engineering unit and is responsible for the coordination of BT's contributions to the OSS activities of the TeleManagement Forum, ITU-T, EURESCOM, and the IETF. He is chair for the TMF Value Chain Market Centre concerned with Telecomm B2B and manages EURESCOM Project P1106 studying Telecomm B2B. He has been awarded the coveted TeleManagement Forum Fellowship in recognition of his contribution to the industry over the last decade.
dave.milham@bt.com

A growing business trend is Internet-based business-to-business (B2B) integration of trading partners in the creation of end-to-end value chains.

The architectural approach to such integration is typified by industry initiatives such as RosettaNet (1) and ebXML (2).

For wholesale communications service providers, such architectures provide the opportunity to integrate a wide range of fulfilment, assurance and billing systems and processes such as those described in the TeleManagement Forum's eTOM (4) with those of their third-party operator, service provider and retailer customers, partners and suppliers.

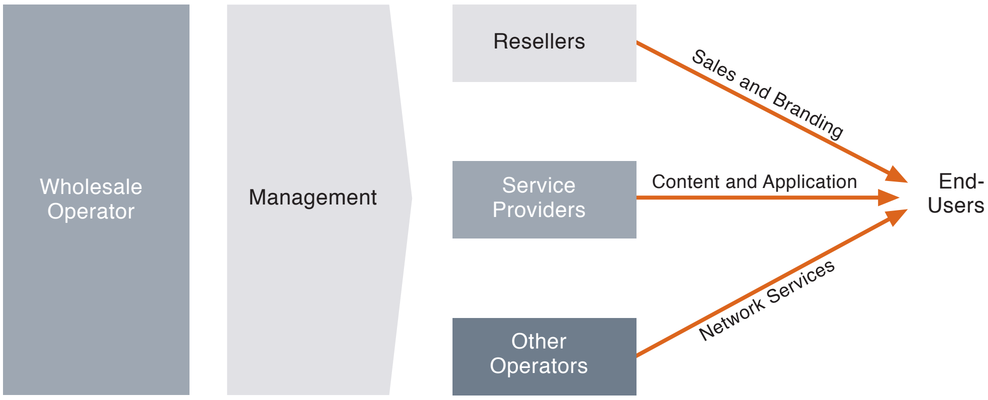


Figure 1 Emerging wholesale customers (Source (5))

Background

Traditionally, telcos have been vertically integrated companies selling their applications via their own retail divisions on their own wholly owned transport.

Where wholesale organisations existed they were principally departments created to discharge regulatory requirements for interconnection.

However, globalisation, regulation and the emergence of new Internet and multimedia service propositions have led to the emergence of many external businesses with the need for wholesale communications services (Figure 1). To meet these commercial and regulatory demands, many large telcos have introduced unbundling of their wholesale services which have become major businesses in their own right (Figure 2).

These new business relationships between wholesale providers and their partners are reflected in the TeleManagement Forum's revised Telecoms Operations Map (TOM – now known as eTOM) which has been updated to reflect the emerging e-business relationships between service providers and their partners in emerging communications service provider value chains (Figure 3).

The difficulty for wholesale carriers is how to work within this new commercial model of federated businesses of external partners and cus-

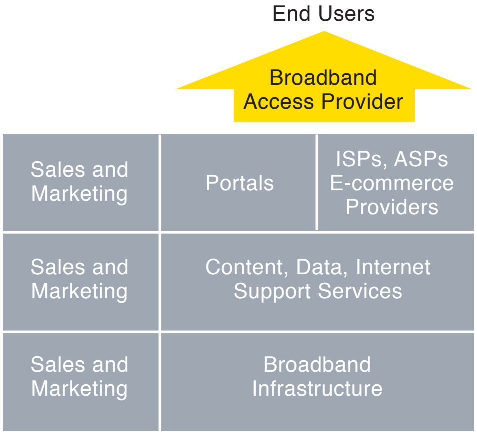


Figure 2 Commercial separation of wholesale service (Source (5))

*) *This paper has previously been published in The Journal of The Communications Network, Vol 1, Part 1, April–June 2002*

Elayne O'Sullivan (25) graduated from Cork (UCC) with a degree in Music prior to gaining an M.Sc. in Information Systems from Cardiff. Elayne now works on OSS solution design within BT specialising in the design and implementation of B2B XML interfaces for BT Wholesale. Elayne is also researching the application of ebXML developed modelling techniques to the telecommunications industry.

elayne.osullivan@bt.com

Martin Roberts (40) graduated from University of Wales, Aberystwyth, having studied Computer Science. On leaving university he worked on real-time systems in the chemical and manufacturing industries before joining BT. Since joining BT, Martin has worked on customer facing applications from the early days of CMIP interfaces and currently is the architect for the XML interfaces in use within BT Wholesale. He is editor of one of the UN/CEFACT ebtwg projects and is a key contributor to the ITU tML initiative.

martin.me.roberts@bt.com

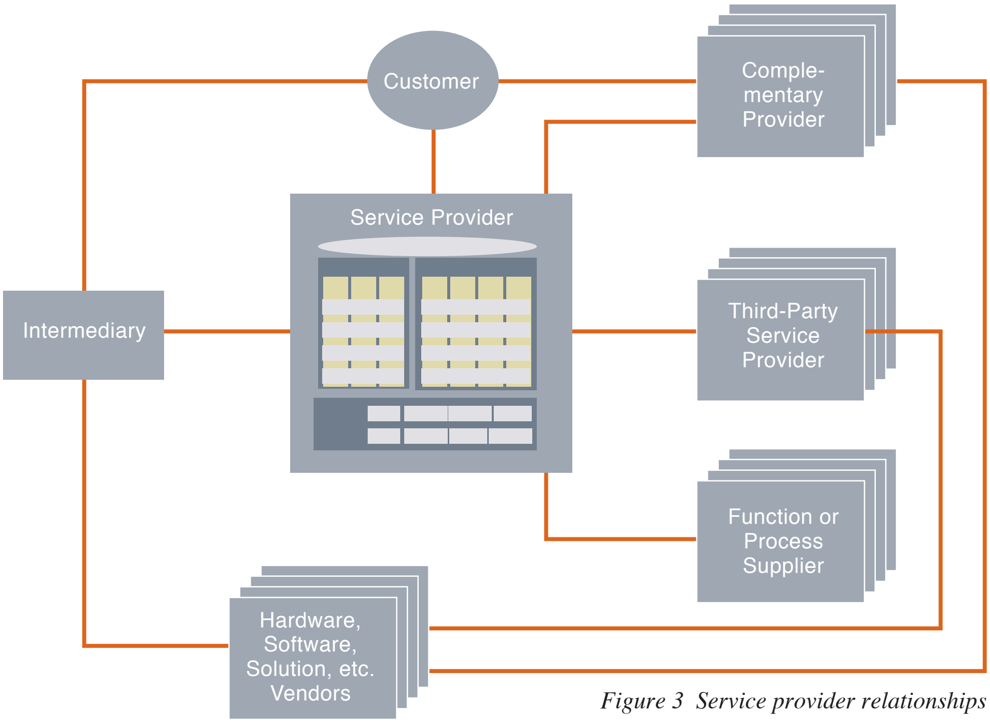

Customer
Complementary Provider
Service Provider
Intermediary
Third-Party Service Provider
Function or Process Supplier
Hardware, Software, Solution, etc. Vendors

Figure 3 Service provider relationships (Source TeleManagement Forum 2001)

tomers, while still retaining the efficiency and effectiveness benefits of the process and systems integration they had enjoyed in the past as part of a larger self-contained and integrated business.

Over recent years, integrated carriers had worked to improve customer service and contain costs through greater process efficiency and effectiveness. This had entailed the improvement of business processes through re-engineering & the realignment of processes along customer value chains through business transformation. Such processes were further enhanced with the support of integrated OSS applications such as CSS (BT's Customer Service System), Switch Manager, and Work Manager (WMS).

The benefits of such integrated system and process support have been to provide customer oriented end-to-end visibility of service delivery and assurance and varying degrees of flow-through automation for the less complex mass market products.

The challenge is now to continue to provide such visibility and flow-through with wholesale trading partners who do not and cannot share direct access to such systems and have systems and processes of their own anyway.

In terms of process integration the eTOM (Figure 4) suggests the additional processes required to manage the launch of new service propositions and operate them in conjunction with trading partners through integration of processes in the 'Marketing and Offer Management' and 'Supply Chain Development and Management' layers for new product development and the 'Customer Relationship Management' and 'Supplier/Partner Relationship Management' layers for operations.

However, the eTOM process map does not address the systems support for such integration.

Figure 5 shows a number of approaches typically deployed within BT to enable management of the customer interface with the organisation's OSS stack.

One approach is to create web-based eCRM (electronic customer relationship management) portals that provide partners with mediated web access to restricted views of customer and service information. This approach is typified by BT's eCO CRM application for BT Wholesale's customers. While this approach provides visibility via a rich graphical user interface, trading partners are still left with the problem of integration of such information with their own information systems.

An alternative approach is to use B2Bi XML messaging gateways to exchange business information as electronic documents between partners' systems across the Internet.

The remainder of this article focuses on these B2B integration technologies and architectures that may be used to loosely integrate partners' support systems and processes to achieve some level of end-to-end process visibility and flow-through.

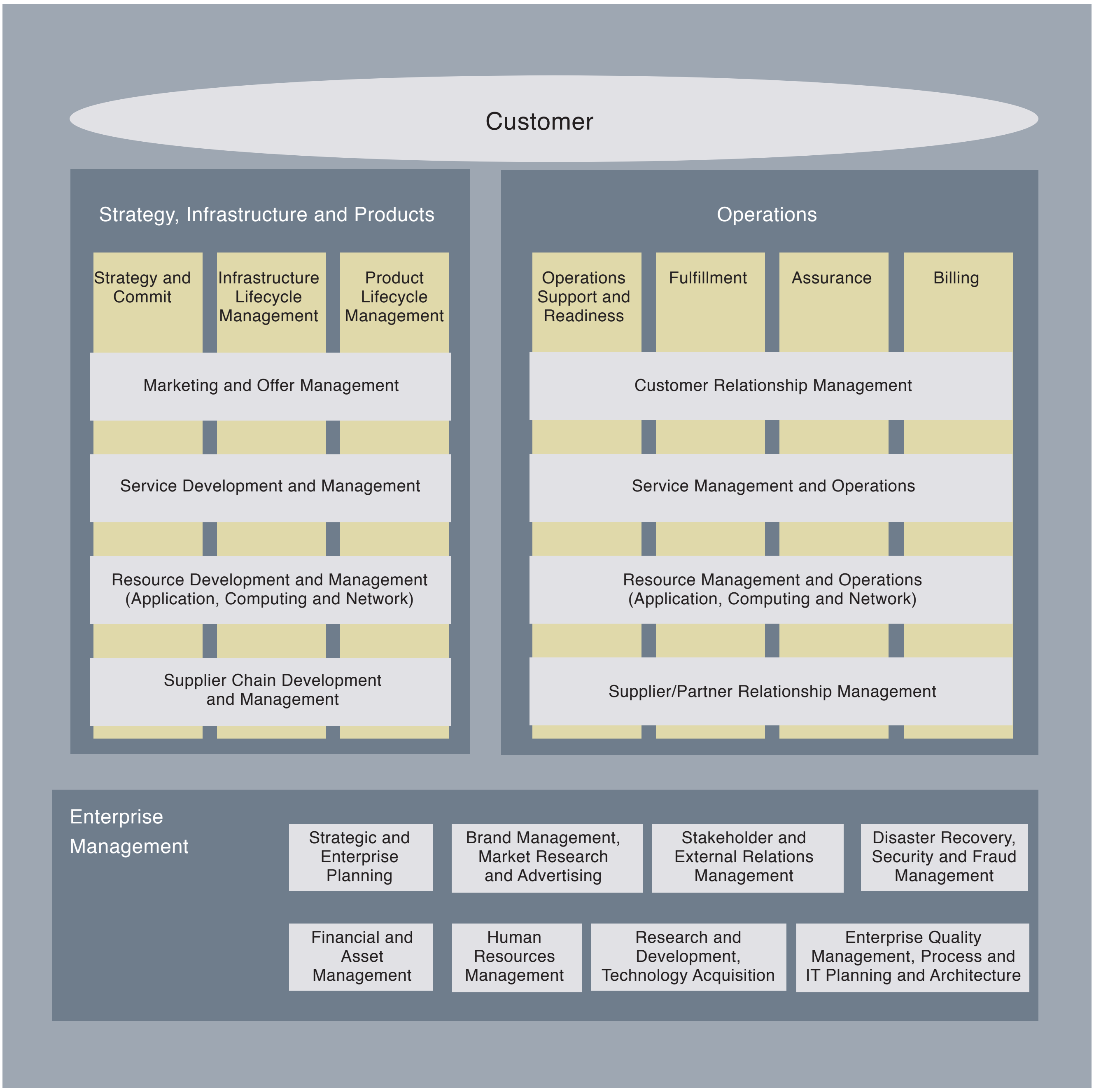

Customer
Strategy, Infrastructure and Products
Operations
Strategy and Commit
Infrastructure Lifecycle Management
Product Lifecycle Management
Operations Support and Readiness
Fulfillment
Assurance
Billing
Marketing and Offer Management
Customer Relationship Management
Service Development and Management
Service Management and Operations
Resource Development and Management (Application, Computing and Network)
Resource Management and Operations (Application, Computing and Network)
Supplier Chain Development and Management
Supplier/Partner Relationship Management
Enterprise Management
Strategic and Enterprise Planning
Brand Management, Market Research and Advertising
Stakeholder and External Relations Management
Disaster Recovery, Security and Fraud Management
Financial and Asset Management
Human Resources Management
Research and Development, Technology Acquisition
Enterprise Quality Management, Process and IT Planning and Architecture

Figure 4 Telemanagement Forum's eTOM v2.5 (©TeleManagement Forum, October 2001)

B2B Integration Concepts and Architectures

There are a variety of approaches to the integration of systems that can be categorised as follows:

- document exchange;
- exposed application;
- exposed business services;
- managed public process (PIP/transaction); and
- managed public and private process.

Each of these approaches reflects a stage of development in the evolution of e-commerce and each has its merits and applications (7). They are defined below:

Document Exchange

This is the classical approach adopted by electronic data interchange (EDI) applications in-

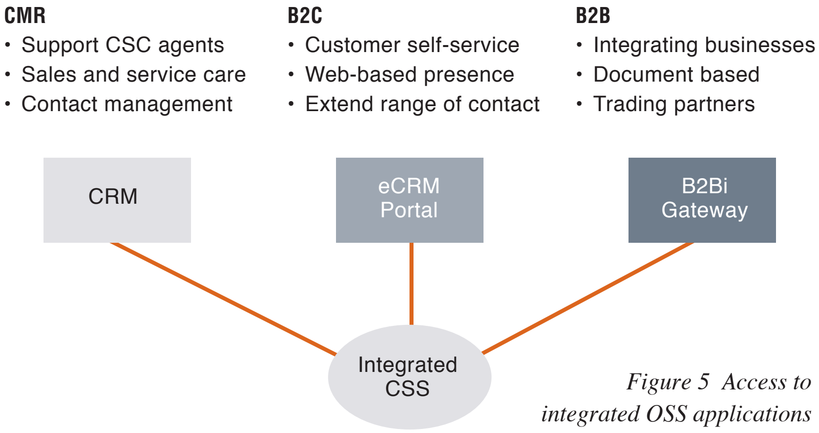


Figure 5 Access to integrated OSS applications

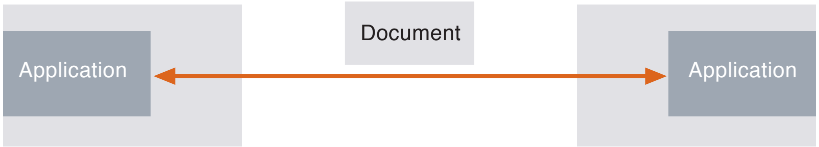


Figure 6 Document exchange based inegration of applications

volving the asynchronous exchange of structured and standardised files representing documents such as purchase orders and quotes (Figure 6).

In 'traditional' applications this exchange is secured through the use of EDI value added network services (VANS) closed user group based networks (although initiatives such as OBI had moved such EDI architectures to the Internet (6)).

The execution of the business logic to produce and exchange the documents through some form of agreed public process is often executed directly by the target applications.

While this approach provides some decoupling of applications, increasingly this approach is seen as expensive to implement and inflexible to change.

Exposed Application

With the advent of customisable software applications and the use of distributed applications, more and more applications have published application programming interfaces (APIs) that can be used by one application to invoke another (Figure 7).

When this style of integration is employed between business partners it is often secured through the use of virtual private networks. Also to deal with system availability issues, messaging middleware is often employed to decouple the two applications.

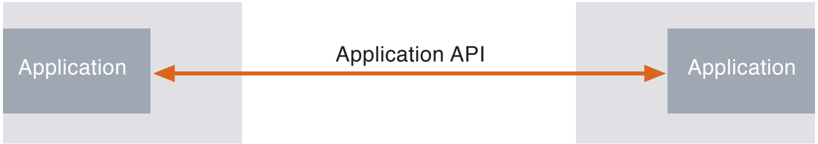


Figure 7 Exposed API integration of applications

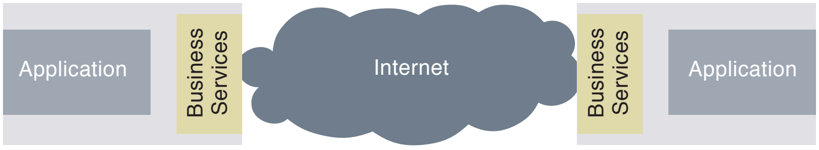


Figure 8 Exposed business service integration

While this approach can provide a very rich and functional integration between two parties, the direct integration of applications in this manner makes it very susceptible to changes in the applications of one party or another. Also one is reliant on trading partners using similar operating systems and applications technology.

Exposed Business Service

The exposed business service approach provides additional flexibility over direct API integration by presenting simplified and abstracted versions of APIs via small discrete units of code that handle specific limited functionality.

The interaction is secured over the public Internet using encryption and strong authentication via the use of digital certificates (Figure 8).

Also the differences in technology between trading partners is dealt with by using an application neutral implementation of XML messaging over the Internet http protocol in the form of a mechanism know as *SOAP* (Simple Object Access Protocol) (8).

This approach is being widely promoted under the name of *web services* and by Microsoft as part of its .NET initiative (9).

While this approach is likely to achieve great benefits for business-to-business integration there are still issues to be dealt with in using such technology widely.

The principal issues are ones of standardisation in two areas:

- security and reliability (including receipted acknowledgement of transactions and how to handle timeouts and failures); and
- vertical standards for content and business process enacted for particular industry segments.

It is possible that the first of these two may be addressed in enhanced forms of web services standards in the future. However, stricter definition of industry standard services would still be required.

Managed Public Processes

This form of integration builds on the notion of web services (although RosettaNet, as an early example of such integration, pre-dates SOAP).

Where this differs from the exposed business service approach is the industry standardisation of process and content for a defined set of business activities, such as the exchange of a purchase order request (Figure 9).

Such business activities conform to a set of standard patterns and involve not just the exchange of business documents but also standard signals to indicate the successful (or otherwise) execution of a transaction to assure a reliable and secure exchange.

The combination of a web services approach together with standardised content provides a very strong basis for the integration of trading partner processes on a transaction-by-transaction basis over the Internet.

However, for some interactions there is also an end-to-end process or choreography of such transactions to be standardised and managed. A typical example would be the UK industry agreed unbundled local loop (LLU) provisioning process that involves a multi-step approach to the provision of a metallic path.

Managed Public and Private Process

The managed public and private process form of integration is a further enhancement to Internet-based integration and deals with the need for selected B2B processes to be an end-to-end choreography of transactions (Figure 10).

This approach allows one to define a second layer of process that links individual public processes (RosettaNet PIPs or ebXML transactions). This layer can also be used to manage the end-to-end process across multiple internal applications.

Such a process is typically implemented using enterprise workflow applications with integration into applications via enterprise integration middleware. Such an EAI integration of multiple applications is becoming common practice amongst telcos (10), which adds further appeal to this form of integration.

In the same manner that public processes need to be defined and agreed with trading partners, the ‘public face’ of such choreographies need to be published.

ebXML, amongst other things, provides a standard for exchanging a description of such a choreography in the form of a *collaboration* which can be decomposed into low level activities implemented by individual *business transactions* to exchange *business documents* between partner roles (the Business Process Specification Schema (3)) as an XML based model.

With the addition of this process modelling technique, the ebXML collection of standards forms the basis for future evolution of web services based integration of trading partners’ processes and systems.

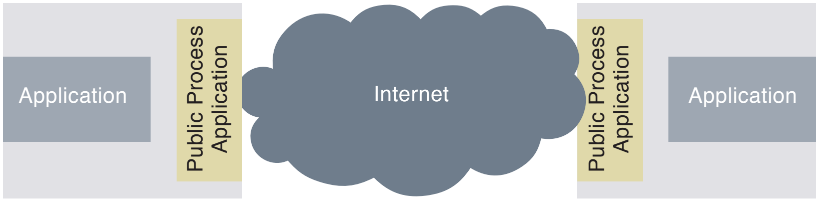


Figure 9 Managed public process integration

ebXML

The stated mission of ebXML is:

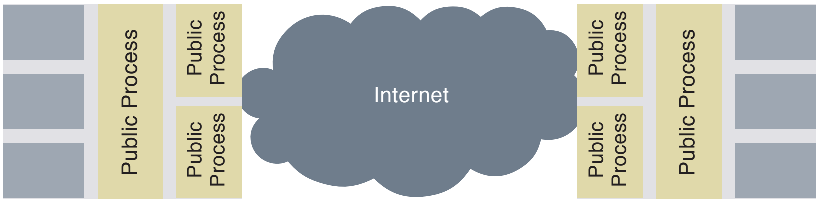
> *“To provide an open XML-based infrastructure enabling the global use of electronic business information in an interoperable, secure and consistent manner by all parties.”*

ebXML is sponsored by UN/CEFACT and OASIS, and is a suite of specifications designed to enable enterprises of any size and in any geographical location to conduct e-business over the Internet. As such, the emphasis has been on a modular implementation of the specifications on a variety of platforms and scales to fit all pockets.

The key specifications of ebXML were published in May 2001 and are:

- EbXML Requirements Specification v1.06;
- ebXML Technical Architecture Specification v1.04 (the overall ebXML conceptual architecture);
- Business Process Specification Schema v1.01 (the schema for the exchange of B2B process designs);
- Registry Information Model v2.0 (the general repository for recording B2B processes and agreements);
- Registry Services Specification v2.0 (the API definition for access to the registry);
- Collaboration-Protocol Profile and Agreement Specification v1.0 (the format for the exchange of trading partner profiles including the B2B services supported and the means of forming B2B agreements); and
- Message Service Specification v1.0 (the messaging protocol for the exchange of ebXML business documents).

Figure 10 Managed private and public process



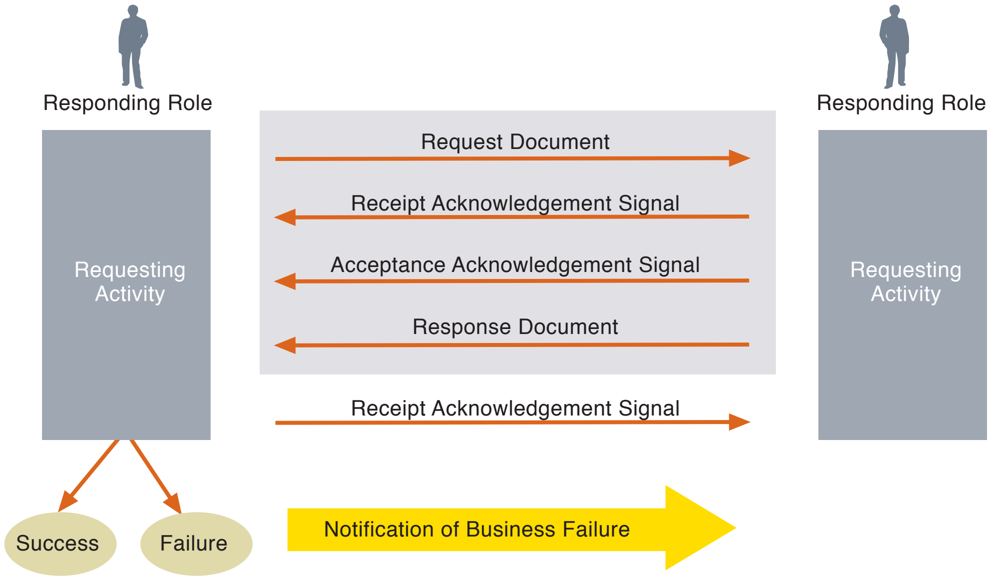

Responding Role
Responding Role
Requesting Activity
Request Document
Receipt Acknowledgement Signal
Acceptance Acknowledgement Signal
Response Document
Receipt Acknowledgement Signal
Requesting Activity
Success
Failure
Notification of Business Failure

Figure 11 Exchange of business documents and signals in executing a transaction between partner roles (Taken from (3))

Of these specifications, the most commonly implemented one is the message service for which there are now version 1.0 implementations from most B2B gateway vendors.

RosettaNet has also stated that the messaging service for ebXML is a likely candidate for the next generation of the implementation framework (RNIF V3).

Figure 11 describes the ebXML concept of messaging-based integration.

Each trading partner fulfils a *role* and executes their respective business processes on their systems and the activities of each partner are synchronised by execution of shared *business transactions* involving the exchange of *business documents* (typically in the form of XML files) such as purchase order requests and confirmations.

In addition to the exchange of business documents, other messages are exchanged as signals of successful execution of phases of a transaction.

The end-to-end sequencing of such transactions and the conditional paths between each step are described as *collaboration* within the business process specification schema that may be visualised in a number of ways. A typical approach is to use an activity graph as shown in Figure 12.

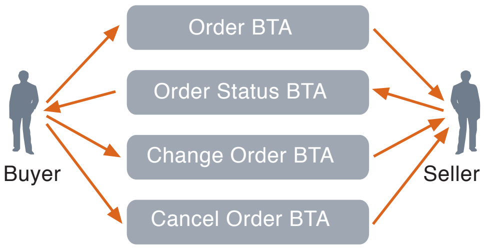


Figure 12 Example list of business transactions between two trading partners that form a business collaboration

Using a combination of standard business transactions and documents within such collaborations forms the basis for a wide variety of B2B processes to be created in support for complex multiparty supply chains not just for procurement but also for assurance and other telco processes.

Challenges Ahead

The Challenges

While the path set by the eTOM and ebXML is clear, there are many hurdles to be overcome before such architectures are widely adopted.

Apart from the messaging service the implementations of the other ebXML standards are not widely available. The ebXML messaging service is now undergoing (as of October 2002) its second phase of interoperability testing with 12 vendors from around the world (13).

For selected ebXML standards (such as BPSS) there are competing alternative standards. Also, the wide promotion of web services and .net as the answer to ebusiness is likely to confuse the general marketplace as to the need for ebXML-based solutions. This will undermine confidence in investment in such standards and delay adoption until the market position is clearer.

Despite the efforts of ebXML, RosettaNet and others to produce cost-effective architectures for e-business integration there is still significant cost in realising solutions based on technologies. In particular, the integration to legacy applications and the understanding of how to translate between the published B2B document and process standards and individual partners' own internal arrangements add significantly to cost.

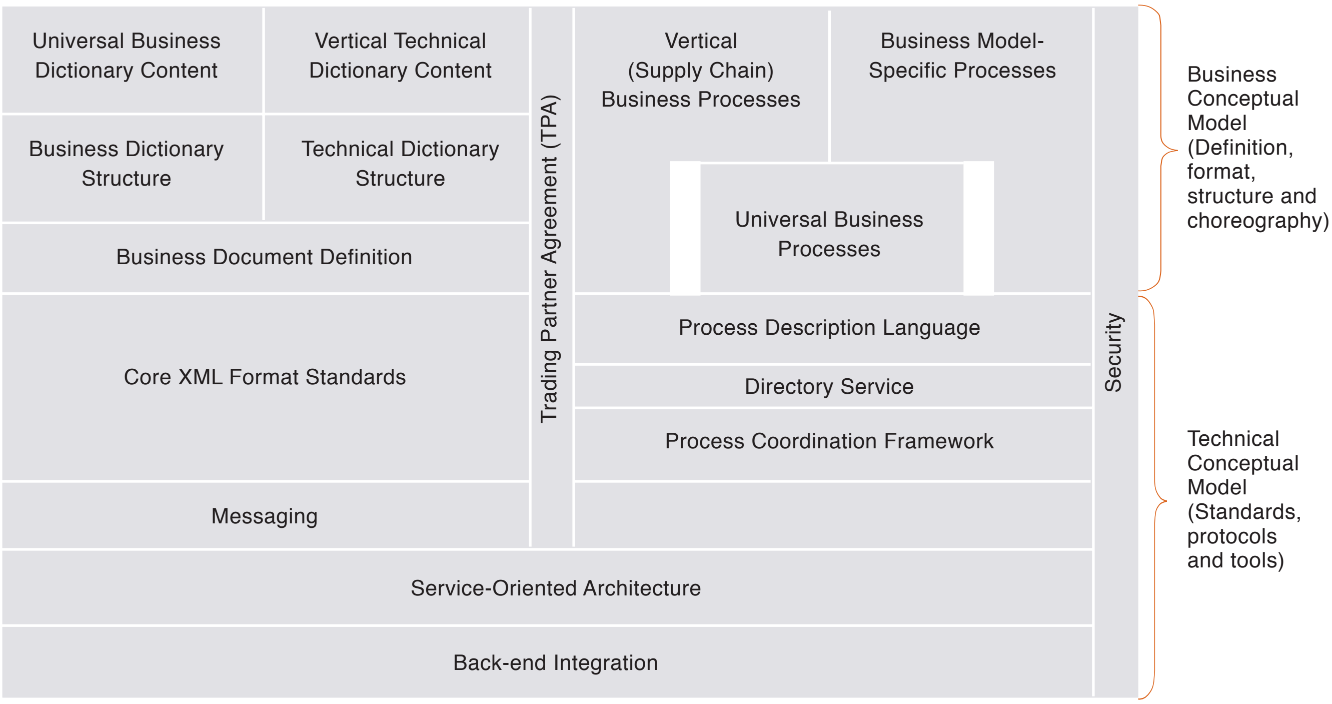

Universal Business Dictionary Content
Vertical Technical Dictionary Content
Business Dictionary Structure
Technical Dictionary Structure
Business Document Definition
Core XML Format Standards
Messaging
Trading Partner Agreement (TPA)
Vertical (Supply Chain) Business Processes
Business Model-Specific Processes
Universal Business Processes
Process Description Language
Directory Service
Process Coordination Framework
Service-Oriented Architecture
Back-end Integration
Security
Business Conceptual Model (Definition, format, structure and choreography)
Technical Conceptual Model (Standards, protocols and tools)

Figure 13 B2B standards conceptual framework (11)

As can be seen from Figure 13, while ebXML and others provide ready standards for the messaging and process integration of enterprises, a major component of any B2B framework is vertical, business specific content.

RosettaNet has dealt with this for the electronics and IT industry by publishing standard transactions (PIPs) with defined names, patterns and XML content.

Other industries, including telecommunications companies, have made similar efforts on selected initiatives (such as number portability) on older EDI standards but have yet to come up with comprehensive vertical process and data standards for all potential telco B2B integrations on newer Internet-based e-commerce platforms.

Without such industry-defined content, trading partners are left with the expense of defining their own proprietary bilateral implementations.

Strategies for Meeting the Challenges

As stated earlier, work has already commenced in implementing the various ebXML standards and proving their interoperability.

Through adopting a modular approach, ebXML has further left the path clear for the market to decide which of the standards are most appropriate, and only time will tell what the final populated framework will look like in terms of products and standards.

The increased use of standard commercial applications and their integration via EAI middleware will also help open up the legacy environment to B2B integration, and reduce cost of integration.

As a further cost-reduction aid, some B2B vendors provide ‘lightweight’ pre-configured versions of their gateway products that larger companies can distribute to their smaller partners with pre-implemented integrations and simple file-based APIs.

In terms of further kick-starting the adoption of B2B integration by smaller trading partners, the leading operators may well need to provide assistance and support in the development and implementation of e-business standards suitable for telcos.

This support should also include ‘partner enablement resources’ comprising specifications, and testing environments for development of e-business integrations prior to live operation.

In terms of the wider telecommunications industry in the UK, BT and other operators have already formed The Telecommunications Industry B2B Forum (formerly known as the *TelcoAPI Forum*) (12). The intent of this group is to develop and promote telco specific B2B process and data standards and promote them through the ITU.

The approach of the group has been to adopt XML business documents from adjacent industries (such as purchase orders) and extend them to cover the particular requirements of service orders for telecommunications services. It is the intention of the group to promote these standards more widely, develop standards for other processes as well as develop procedures for the configuration management of these interfaces.

Applications Within BT

Overall, BT Wholeale's vision is to deploy B2B technologies as an enhancement to its eCO services, as part of a wider transformation of its OSS infrastructure, and support as wide a range of customer and supplier facing processes as is practicable using such technology.

Within BT Wholesale efforts to employ these eBusiness architectures have already been underway for over two years.

Currently the BT Wholesale eCO application supports XML-based ordering interfaces using a proprietary Access Framework mechanism developed to allow trading partners to exchange business documents using http post and get commands to a public hosted web server secured with digital certificates.

The framework has been reused for XML interfaces for LLU and ADSL provisioning and there are currently efforts underway to look at support for trouble ticketing.

With the emergence of early implementations of the ebXML messaging service, BT is also looking to enhance these early implementations with phased replacement of the Access Framework with an ebXML messaging gateway as a more standards based and functional approach.

BT Retail is also active in the use of B2B technologies with its suppliers (in particular Cisco) via use of RosettaNet standards and moreover, BT Wholesale is looking to reengineer and streamline its network plan and build processes with major network equipment vendors using B2B gateways and standards.

Conclusions

Emerging e-business standards such as RosettaNet and ebXML have been prompted by the emergence of the Internet as a platform for e-commerce.

As wholesale providers develop a wide range of trading relationships with partners, suppliers and customers the opportunity presents itself to engage in a much wider series of *collaborative commerce* activities than just the buying and selling of goods and services.

The TeleManagement Forum's eTOM begins to suggest just how wide the scope of these activities could be and the opportunities for OSS integration via B2B technologies.

However, there are significant standardisation, cost and maturity issues to be overcome prior to wider spread adoption of these applications.

Nevertheless, the links are coming hot out of the forge: time to start building the chain.

References

1 *RosettaNet*. August 16, 2002 [online] – URL: http://www.RosettaNet.org

2 *ebXML*. August 16, 2002 [online] – URL: http://www.ebxml.org

3 *Business Process Project Team*. ebXML Business Process Specification Schema, Version 1.01, 11 May 2001. URL: http://www.ebxml.org

4 *Telemanagement Forum*. eTOM The Business Process Framework for the Information and Communication Services Industry – GB921 v2.5. URL: http://www.tmforum.org

5 Uglow, S, Gambhir, A. *Wholesale: New Markets for Communications Carriers and Service Providers*. OVUM, Oct. 2000.

6 *Open Buying on the Internet*. August 16, 2002 [online] – URL: http://www.openbuy.org/

7 Chappell, D et al. *Professional ebXML Foundations*. Worx Press Ltd., 2001. (ISBN 1-861005-90-3)

8 *Simple Object Access Protocol (SOAP) 1.1*. W3C Note 08 May 2000. URL: http://www.w3.org/TR/SOAP/

9 *XML Web Services*. August 16, 2002 [online] – URL: http://www.microsoft.com/net/defined/xmlservices.asp

10 Gerrese, J. iCE: The 'Cool' Operation Support System and Interactive Customer Empowerment Engine. *The Journal of the Institution of British Telecommunications Engineers,* 2 (3), 2001.

11 Coleman, D. *B2B Standards*. RosettaNet Presentation.

12 *The Telecommunication Industry B2B Forum*. December 4, 2002 [online] – URL: http://www.telcoB2B.org.uk/.

13 *The Drummond Group*. October 1, 2002 [online] – URL: http://www.drummondgroup.com/html-v2/pr_10-01-02.html

Next Generation Infrastructure for Electronic Collaboration

ØYVIND AASSVE

Øyvind Aassve (36) is an IT architect at Telenor Networks, where his main focus is IT architecture and business-to-business issues. He holds an MSc in Information Systems (2001) from the University of Texas at Arlington. Before joining Telenor Networks he worked as a programmer at eTech Solutioncorp, and has previously also worked extensively with enterprise software as a business analyst and project coordinator at Scandinavian PC Systems Group (now Visma Software). Mr. Aassve has participated in Norsk Edipro's Infrastructure project where he led the working group "Description of collaboration models in an open infrastructure" and participated in two other groups.

oyvind.aassve@telenor.com

The Infrastructure Project

The Infrastructure project was initiated by Norsk EDIPRO to facilitate the adoption of electronic collaboration especially among small and medium sized Norwegian enterprises by providing an open infrastructure for collaborating applications and enterprises. The vision is to achieve flexible and seamless integration between applications of enterprises throughout the supply chain, and to manage this in an easy, cost efficient and universal way.

This article provides an overview of the results of the Infrastructure project's May deliveries from the two working groups "Description of collaboration models in an open infrastructure" led by Øyvind Aassve, Telenor Networks, and "Application Readable Models in an open infrastructure" led by Arild Nybakk, Scandinavian Transport Systems.

Background

There is an acknowledged and universal need today for organizations to be able to collaborate electronically, preferably throughout the supply chain. This requires an open infrastructure of collaborating applications that can be implemented by all participants in the supply chain without long and complex implementation projects. The business world is dynamic, and an electronic infrastructure is required to support easy and cost efficient change of business partners as well as business protocols.

With this in mind the Norsk EDIPRO Infrastructure project embarked on a journey to establish a sound infrastructure – the goals being the enabling of full application integration on a semantic level, resource effective and manageable enough to be adapted by small and medium sized enterprises. By "full application integration" we mean in this context the enabling of two or more collaborating business applications to produce, interpret and process the contents of exchanged message data on the basis of the semantics of a common reference model containing all information relevant for business collaborations within the domain in question. The applications to be integrated would typically be end-user applications like ERP systems, logistics systems and financial systems – in other words the critical production systems of a company or an organization.

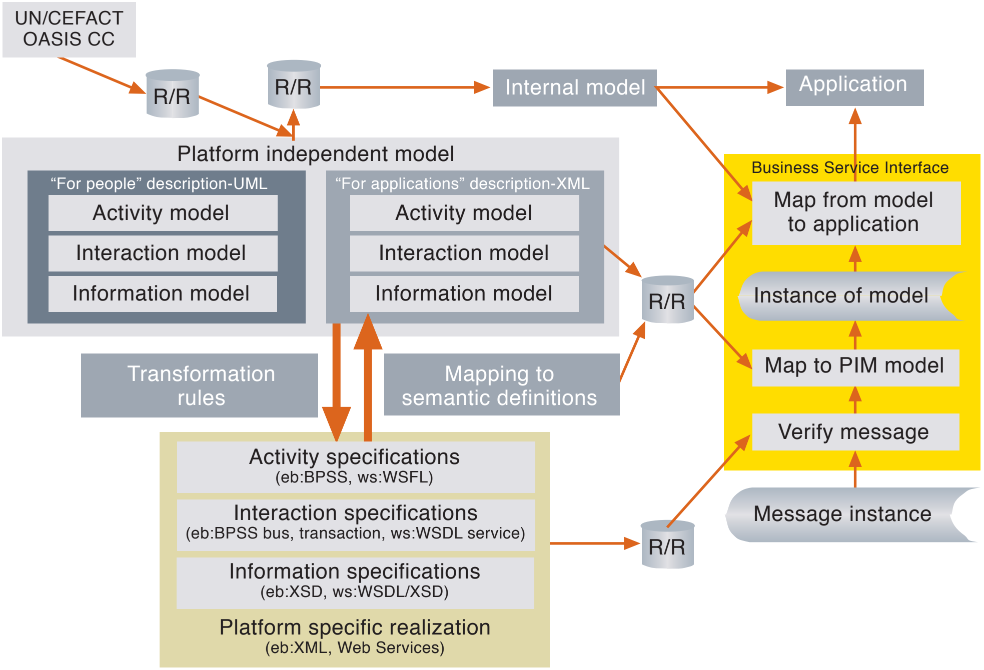
In order to succeed with electronic collaboration there is a set of process and information models that need to be aligned. The infrastructure needs to ensure that not only the users in the different organizations are interoperable, but also systems and applications. For this a set of semantic definitions need to exist that can make Organization A's proprietary e-commerce systems communicate with Organization B's proprietary e-commerce system. To enable a flexible and easily extensible collaboration, it is necessary to align different descriptions. First of all the collaboration should be built on a *domain model*. A domain model is typically developed by industry organizations and attempts to standardize processes and information across an industry to facilitate electronic collaborations. Further, each company's description of their internal business processes will constitute the *business process model*. Organizations in a given industry will typically have different internal process models for a given function, but they should be related to the same domain model. Then at the final level two or more organizations, belonging to the same or different domains, can go together to specify how the specific collaboration processes should be implemented between them in a *collaboration model*.

The Proposed Architecture

A central issue in defining an architecture is to embrace the rapid changes we see in technology and business environments today. A natural choice would therefore be to use the model-driven approach for our architecture. With a model-driven architecture we decouple design and realization so that our fundamental business process design can be realized in the technology of choice, earlier that might have been EDI/EDIFACT or CORBA, today it could possibly be XML Web Services or electronic business XML (ebXML). This also allows you to describe your collaboration process in the same manner regardless of which platform you want to base the realization on. The central issue is to keep the design in a platform independent version in order to be able to easily migrate to an alternative realization as business needs change and technology evolves.

Figure 1 shows the complete architecture model. An important requirement for our work has been that the Platform Independent Model (PIM) must

Figure 1 The architecture model



be available in two forms, one "for people" version (the left box of the Platform Independent Model) that can be read and understood by humans – preferably including non-IT professionals; and one "for application" version (the right box of the Platform Independent Model) which is optimised to be read and processed by applications. The "for people" version is described using a set of UML diagrams, while for the "for application" version these diagrams are implemented using a lighter version of the OMG standard XML Metadata Interchange (XMI). This "XMI Light" version was developed by SINTEF for the project, because full-length XMI was found to be too verbose.

An extension of the Resource Description Framework (RDF) – DARPA Agent Markup Language (DAML/RDF) was also evaluated for the "for application" version. RDF is central to W3C's efforts to create the semantic web. DAML/RDF does not however handle activity diagrams, mainly due to lack of attention to describing business and collaboration processes.

With a collaboration process now modelled in UML and then converted into XMI Light, the next step is to select a set of transformation rules that can transform the PIM into a Platform Specific Realization (PSR). The idea here is that there will be one set of transformation rules for each realization, and all you need to do is select the rule set for the realization of your choice.

Central to the model is also the Business Service Interface whose purpose is to map the domain semantics of the platform independent model with the semantics of the internal application model. This results in a significant improvement from the former field-to-field mappings in EDI, because it enables us to communicate message contents as specified by specific realization documents (XSD schemas, BPSSs, etc.) and interpret these contents based on the underlying common semantic domain model. Telecommunication Markup Language (tML) is a possible domain model for the telecom industry.

Description Techniques

Most of the processes to be implemented in the architecture described above will be stored in public or private repositories. To achieve successful interoperability it is crucial that the processes are described in the same manner by all the participating business partners.

Figure 2 shows the three different submodels/ UML diagrams that have been selected to capture the complete business collaboration.

- The activity model (UML Activity diagram) shows what activities/processes to be performed by each role in a collaboration. This flow of operations and information is often referred to as choreography or orchestration. The realization will be based on BPSS for ebXML and WSFL for XML Web Services.

- The interaction model (UML Component diagram) illustrates which roles collaborate over the different interfaces. Realizations will be based on the BPSS for ebXML and WSDL for XML Web Services.

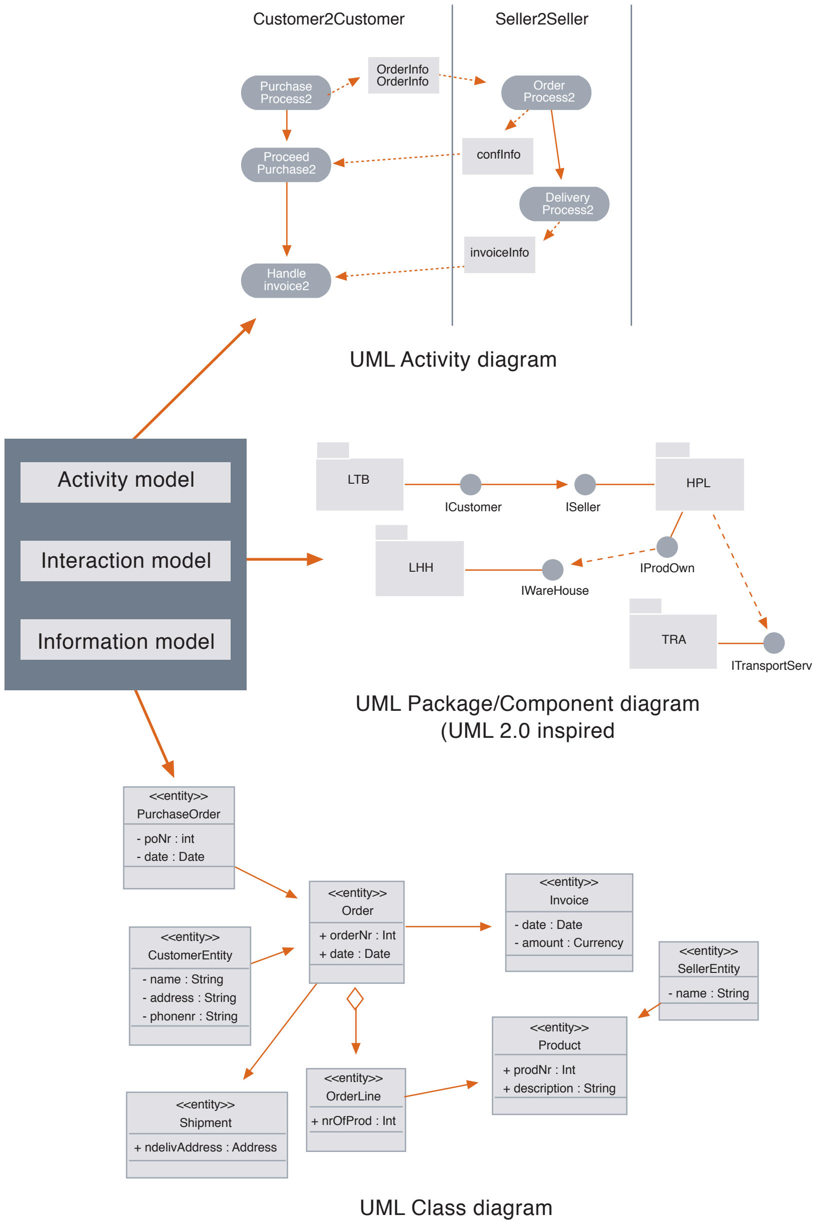


Figure 2 UML diagrams are used to describe the submodels in the architecture. In the "for application" model these diagrams will be described through XMI

- The information model (UML Class diagram) depicts the information included in the collaboration in the form of classes, attributes and relationships. The information model is also the basis for expressing the information contained in the flow-objects in the Activity Model. Realizations will typically be based on XML Schema and in some instances on WSDL for XML Web Services.

The three complementary submodels contain attributes that together represent necessary aspects of a collaboration. A minimum is to describe the activity and information model to define the collaboration process.

The models can be presented in either a conceptual or a more detailed version. The activity model in the figure above is detailed in the sense that it contains flow objects, while a conceptual model would only depict the process flow. Flow objects are objects that represent the information that flows between the participating roles in the processes, and they contain the contents and structure of the messages that are sent between the collaborating parties.

An issue regarding the activity model is also the representation of state in the system. International initiatives like ebXML has introduced Business Entity Types (Business Object Types) to help record the state of a collaboration as it progresses. As the sub-processes are completed the "state" of the collaboration reaches new levels. State then becomes a function of the progress of the collaboration and can be useful for the collaborating partners to synchronize. This can be achieved through a business entity (business object) whose purpose it is to update the collaboration status at all times. Change of status happens when one party completes an activity that is being synchronized through an information flow.

The Semantic Connection

The relationship to semantics and semantic information is critical to enable the collaborating partners to have the same understanding of the content in the platform independent model.

The word semantic means "science of the meaning of words". In our work a semantic definition is a precise and unambiguous explanation of the meaning associated with an element in a collaboration model.

Any application is built upon a "semantic universe" expressible through an activity and information model – the meaning content of the application is carried by the model. The part of this universe which – directly or indirectly – is involved when an application participates in an electronic collaboration, must be – or be made into – a part of the semantic universe of the other application(s) taking part in the collaboration.

This is the purpose of the Business Service Interface (BSI). The BSI downloads external schemas (ex. a potential partner's process realization document) from different Registries/Repositories (R/R) and makes the semantics of these available for the internal applications of the owner of the BSI. The BSI is basically responsible for two mapping jobs. The first one is the alignment of the semantics of an organization's internal applications to the semantics of the platform independent model. The other one is the mapping from the realization document specifications to the platform independent model semantics – this is partly done during set-up, partly on-the-fly as necessitated by changes or extensions in the realization documents. For the latter mapping from Platform Specific Realizations to Platform Independent Model an application of the W3C standard Resource Description Framework (RDF) is chosen, containing either Xpointer expressions or Universal Unique Identifiers (UUIDs) as the concrete mapping mechanism.

Applications participating in a collaboration will have to map the semantics of any operation, information item or other business entity to be used in collaborations within the domain in question. We consider the semantics of such a business entity to be encapsulated in what we have labelled a *semantic carrier*. A semantic carrier can be thought of as a location or reference point uniquely identifying the semantics of the business entity. It is important to note, however, that the carrier is located in the context of a model for the business domain in question, making it more than just an object with a given identifier – the semantics of the carrier is the combination of the semantics of the component as such and the semantics of the model context in which it appears. This means that we need to contextually identify the different elements carrying semantic information in the model. The details will not be presented in this article, but as an example, in order to represent the information model semantically it may be necessary to semantically locate information on class, attribute, association and code list identifier level.

Platform Specific Realizations

The project's assumption is that XML Web Services and ebXML will become the most popular candidates for Platform Specific Realizations. The specifics of XML Web Services will be better explained in other articles of this issue of *Telektronikk*, but I find it worthwhile mentioning here that in order to implement full collaborations through Web Services further capabilities than those provided by WSDL, SOAP and UDDI today are necessary. In order to implement Web Services as part of larger business processes IBM has put forward the Web Services Flow Language, which is already supported by a number of tool vendors. Microsoft has come up with their XLANG language which serves similar purposes. IBM and Microsoft are currently working together to possibly combine their efforts so that the Web Services community only needs to relate to one orchestration standard.

ebXML was developed in a joint initiative of the United Nations (UN/CEFACT) and the Organization for the Advancement of Structured Information Standards (OASIS) between November 1999 and May 2001. The ambitious goal was to create an infrastructure for a single global electronic market. ebXML is composed of the following four elements:

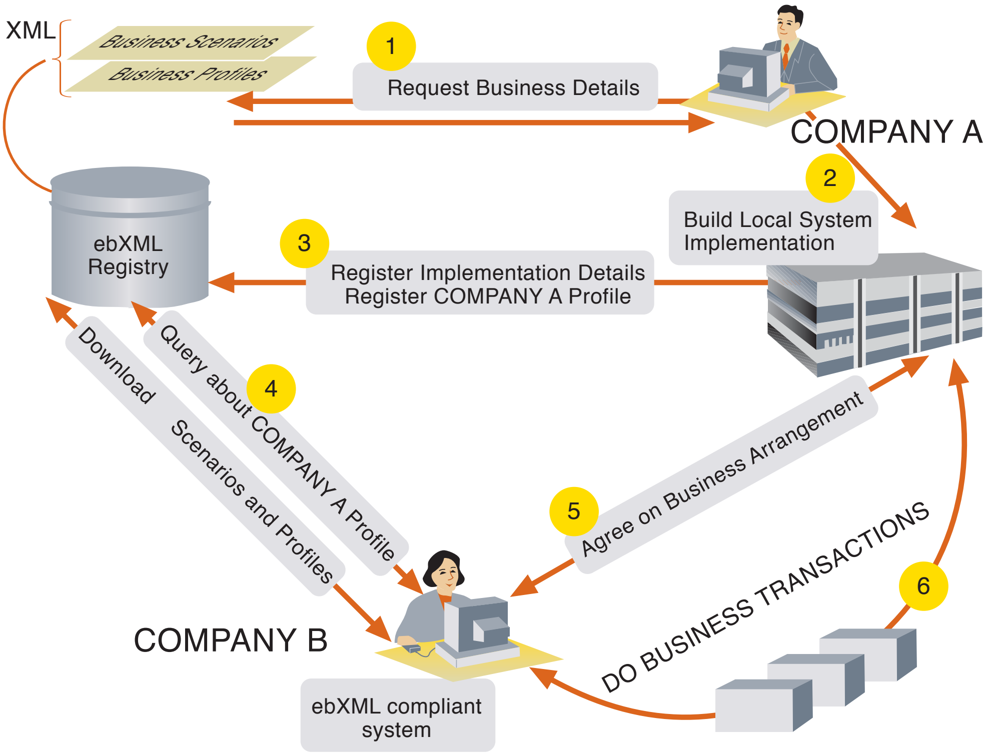

XML
Business Scenarios
Business Profiles
1
Request Business Details
COMPANY A
2
Build Local System Implementation
ebXML Registry
3
Register Implementation Details
Register COMPANY A Profile
Download Scenarios and Profiles
Query about COMPANY A Profile
4
5
Agree on Business Arrangement
DO BUSINESS TRANSACTIONS
6
COMPANY B
ebXML compliant system

Figure 3 The ebXML architecture

Business Process Specification Schema (BPSS): The Specification Schema provides the definition of an XML document that describes how an organization conducts its business. While the CPP/CPA deals with the technical aspects of how to conduct business electronically, the Specification Schema deals with the actual business process. It specifies such things as the overall business process, the roles, transactions, identification of the business documents used, document flow, legal aspects, security aspects.

Trading Partner Information: The Collaboration Protocol Profile (CPP) provides the definition (DTD or XML schema) of an XML document that specifies the details of how an organization is able to conduct business electronically. It specifies how to locate contact and other information about the organization, the types of network and file transport protocols it uses, network addresses, security implementations, and how it does business (a reference to a Business Process Specification). The *Collaboration Protocol Agreement* (CPA) specifies the details of how two organizations have agreed to conduct business electronically. It is formed by combining the CPPs of the two organizations.

Messaging Service: Provides a standard protocol neutral way to exchange business messages between organizations. SOAP was finally selected as the standard.

Registry: A database storing items that support doing business electronically. Examples of items in the registry might be XML schemas of business documents, definitions of library components for business process modelling and trading partner agreements.

The process of establishing a business collaboration based on ebXML will typically be as follows (Figure 3): based on a review of the information available from an ebXML Registry Company A can build or buy an ebXML implementation suitable for its anticipated ebXML transactions. After enabling their own applications for ebXML transactions Company A creates and registers a CPP with the Registry. Once Company A is registered other companies can look at Company A's CPP to determine whether it is compatible with their own CPP and requirements. If it is, the potential partner should be able to negotiate a CPA automatically with Company A. In addition there will normally be meetings to specify the business conditions for their collaboration.

This should have given you a taste for the possibilities of business collaborations in the near future. The full documentation of Norsk EDIPRO’s Infrastructure project including description and rationale behind the different choices is available at http://www.edipro.no/index.php?id=47796&cat=1323. “Description of collaboration models in an open infrastructure” is for the time being only available in Norwegian.

The Self-Service Channel – “Dine sider” (“Your pages”)

ROBERT LANDSEM

Robert Landsem (28) graduated as Computer Engineer from Sør-Trøndelag University College (HiST) in 1997. He worked for two years at Enitel a.s, and is currently working as an Internet architect in Telenor Plus.

robert.landsem@telenor.com

1 Introduction

The purpose of this document is to show how to use Microsoft .NET to integrate with Java EJB services using XML Web Services technology. This document is based upon experience from the eChannel (eKanal) project at Telenor. Using this kind of technology is rather new and exciting, and not many sites have been implementing this kind of technology yet.

2 About the Application

The eChannel project has developed a web site “Dine sider” (“Your pages”), which is a self-service channel web site for our private customers with fixed line phones. On this site authenticated users can manage their subscription online and order/cancel products and additional services. For instance they can get consultation on which type of subscription or service they should get, copies of previous invoices and incurred cost since last invoice. The site also provides a lot of functionality for non-authenticated users. Orders and cancellations of products/services are automated by utilising the appropriate services supplied by our middleware platform.

“Dine sider” has approximately 70,000 visitors, and 20,000 users logged on every week. That makes us one of the biggest web sites with this kind of technology in Europe. The site also has extensive use of personalized information and campaigns, e.g. you do not receive campaign on broadband Internet connection if you already have this product. All of this is made possible with ASP.NET (C#), XML Web Services, SOAP and Commerce Server.

3 Background

Eighteen months ago, “Dine sider” was implemented using Active Server Pages, communicating with a Java middleware platform in another business area in Telenor. For business related reasons we had to leave their middleware. The responsibilities of the middleware platform are abstracting functionality and information located in various internal production/ legacy systems.

Telenor was at that time working on a new middleware platform. Their goal was to abstract functionality from our legacy systems. The technology used for implementing was already decided to be Java.

Telenor was also striving to gain an intimate knowledge of its customers and to personalize and up-sell its services. The previous solutions were not focused on logging what customers were doing when visiting the web site. We required our new solution to better keep track of how users were browsing and using the web application. Another challenge we experienced with the Active Server Pages implementation was the time and cost implications of rapid changes to the user interface. This because the Active Server Pages contained both HTML and user interface controls.

4 Technology

Several technologies for realizing our web front-end and functionality were evaluated.

- Java
- Siebel
- Microsoft .NET

We required a solution where we could easily make adaptations and on-line integration. Siebel technology was not found to be the best solution for us. Microsoft .NET was the chosen vendor based on our previous experiences with Microsoft’s products and excellent support for XML Web Services.

XML Web Services are the cornerstone of the .NET platform. The hope for the XML Web Services technology is that it will bring us into a new era of programming. Rather than relying on the assembly of platform dependent components or objects, the XML Web Services technology is based on the reuse of distributed, platform independent services. These services and their corresponding contracts are exposed using (UDDI) catalogues publicly accessible.

5 Web Service Overview

A web service is a mechanism for invoking functionality from another application or system on any platform using standard Internet technologies.

XML Web Services exchange data using XML and transport data using HTTP, for example. The most common Web Services language (protocol) is currently Simple Object Access Protocol (SOAP). The protocol does not, however, specify how messages are going to be built and handled.

Important standards for Web Services are Web Service Description Language (WSDL) and

XML Schema Definition (XSD) documents. The WSDL documents contain the services interface definitions: the services messages, the services ports and the services encoding. Messages are described as RPC names and XSD types. Ports are described as IP and port number. Bindings are, in our case, always SOAP RPC encoded. The XSD documents contain the messages type definitions, in an XML format.

The WSDL defines the interface and is used to create an XML Web Services proxy. This proxy is then used to interact with the Web Service.

On the service layer there is typically a Web Services handler to process the incoming Web Services call. The handler translates the call from SOAP XML into the language of the services and then executes the method on the service. The response from the method call is then translated back to SOAP XML and returned to the proxy. The translation from and to XML is referred to as serialization/deserialization or, sometimes, marshalling/unmarshalling.

The roles of the above parts are shown in Figure 1.

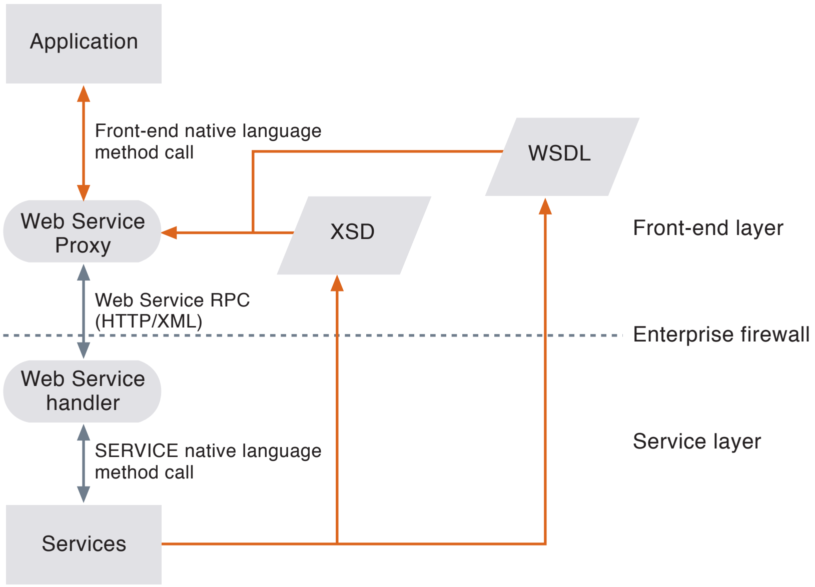


Figure 1 Use of WSDL/XSD documents from the service layer on the front end

6 SOAP Overview

The Simple Object Access Protocol (SOAP) is an XML syntax for exchanging messages. It is designed to exchange structured and typed information on the Web, for example. Because it is XML, it is both language and platform independent. SOAP allows applications implemented in different languages and on different platforms to communicate. At present, SOAP has been implemented in over 60 languages on over 20 platforms. SOAP is a fundamental part of Microsoft .NET and it will be important for developers moving to Microsoft .NET to understand what SOAP is and how it works.

Let us assume that we have a very simple corporate database that holds a table specifying customer reference numbers, names, addresses and telephone numbers. You now want to offer a service that enables other systems to perform a lookup on this data. The service should return a name for a given telephone number.

The prototype of the service will look like this in C#:

```
string getCustomer ( string
customerPhoneNumber );
```

The SOAP developer will now encapsulate the database request logic for the service in a method in any kind of programming language, and then set up a process that listens for the request to the service. The request is in SOAP format and contains the service name and any additional parameters. The listener process decodes the incoming SOAP request and transforms it into an invocation of the appropriate method. It then encodes the result of the method call into a SOAP message (response) and sends it back to the requester. Figure 2 shows the arrangement.

A SOAP-encoded RPC dialogue contains both a request message and a response message. A demonstration of how a simple service returns the name of the owner of a telephone number is shown on the following page.

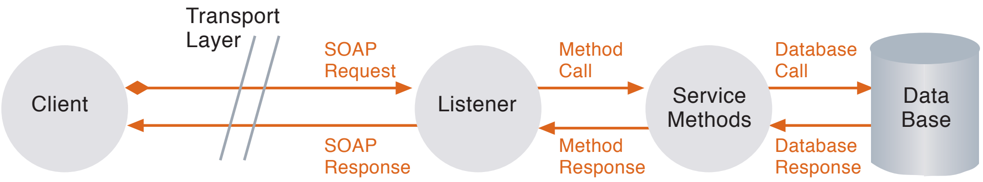


Figure 2 Illustration of a request from a client down to a listener on the service layer

1 Method Signature

```
string getCustomer ( string customerPhoneNumber );
```

2 Request

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
     <SOAP-ENV:Body>
        <ns1:getCustomer
         xmlns:ns1="urn:MySoapServices">
           <param1 xsi:type="xsd:string">22041968</param1>
        </ns1:getCustomer>
     </SOAP-ENV:Body>
</SOAP-ENV:Envelope>  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
```

3 Response

```
<?xml version="1.0" encoding="UTF-8" ?>
  <SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
       <SOAP-ENV:Body>
          <ns1:getCustomerResponse
           xmlns:ns1="urn:MySoapServices"
           SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/ encoding/">
             <return xsi:type="xsd:string">Robert Landsem</return>
          </ns1:getCustomerResponse>
       </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

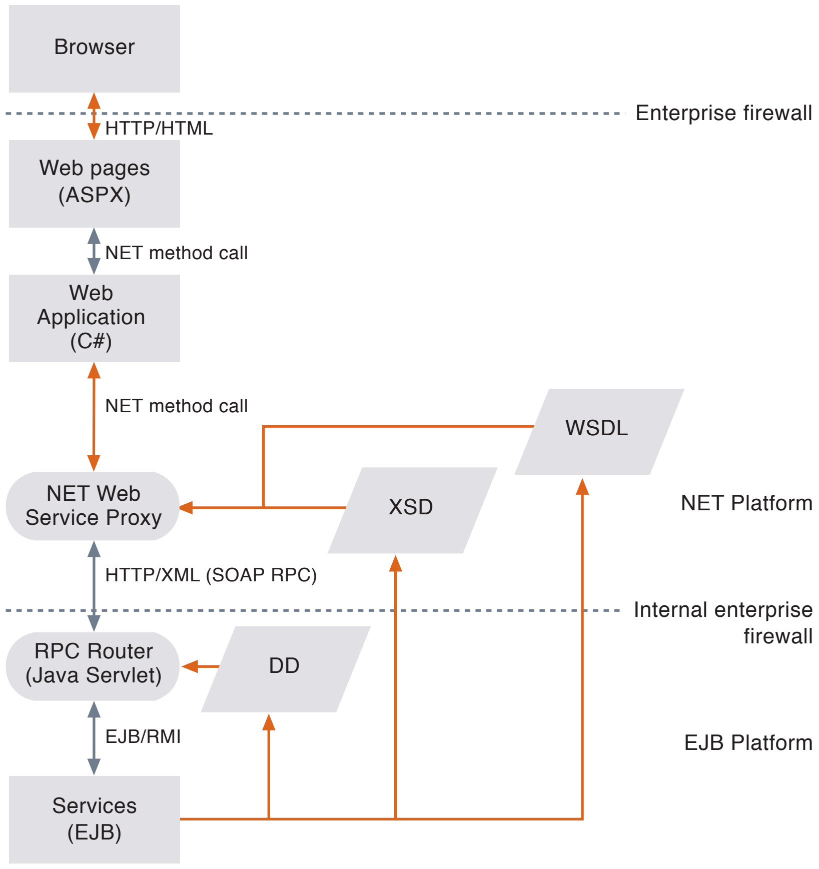


Figure 3 Overview of the architecture used by Telenor

7 Our New Web Site

For this particular project we used WSDL and XSD to describe the services provided by the service layer. These documents were used as input for the Microsoft .NET front-end system in order to automatically create the Web Service proxy. The proxy wrapped all the Web Service details into C#, and the Web Application developers used the XML Web Services as normal method calls within their code. The WSDL and XSD are only used once to create the proxy at design time. At runtime the WSDL and XSD documents are not used.

The chosen XML Web Service handler is the open source Apache SOAP. The access point is a servlet, RPCRouter, which translates the SOAP XML to Java and performs EJB lookup and a remote invocation through reflection of Java objects. The RPCRouter handles all the Web Service requests. The servlet uses Deployment Descriptors (DD) to translate (serialize/deserialize) from XML to Java and vice versa. The DD are Apache specific translation maps (XML to Java) described using a special XML syntax. Our implementation includes Deployment Descriptors (DD).

8 Lessons Learned

We had some problems converting some data types from Java to C#. Caution should be shown with different handling of data types on different platforms.

Another important issue is Apache SOAP's use of Deployment Descriptors. They introduce an unnecessary step in the development of Web Services. DD is a proprietary definition and could be replaced with WSDL and XSD documents.

Within three months in October 2001, Telenor implemented a customized web site based on Microsoft .NET platform.

9 Conclusion

When it comes to exposing objects over the Internet, current XML Web Service technology is an excellent (and a lot simpler, extensible and manageable) alternative to DCOM or CORBA. It is also much easier to implement.

Even though XML Web Services were primarily chosen because of difficulties integrating heterogeneous systems, there is no reason not to use XML Web Services to integrate homogeneous systems.

By implementing XML Web Services, companies can automate many business operations, thereby creating new operating efficiencies and more efficient ways of doing business.

And with XML Web Services standards already incorporated into their IT systems, companies are better prepared to take advantage of upcoming opportunities to transform their business.

Web services are useful for both B2B and for internal application integration, e.g. the development of applications that use Web Services to connect to trusted business partners.

10 Further Reading

Microsoft. November 12, 2002 [online] – URL: http://msdn.microsoft.com/soap

Schjefstad, K. *White paper – Implementing Web Services in a heterogeneous environment*. MSO Norway, 4 Dec 2001.

IBM on Web Services. November 12, 2002 [online] – URL: http://www-106.ibm.com/developerworks/webservices/

Mayo, S. *Five Early Birds' Business Case Studies*. IDC Web Services, Aug 2002.

11 Acronyms

SOAP	Simple Object Access Protocol
HTML	HyperText Transport Protocol
RPC	Remote Procedure Call
XML	eXtensible Markup Language
XSD	XML Schema Datatypes
WSDL	Web Service Description Language
EJB	Enterprise Java Beans
RMI	Remote Method Invocation
JDBC	Java Database Connectivity
DAO	Data Access object in the Origo project
DD	Deployment Descriptor

Web Services in Action: The User Profile Web Service

ANNE MARIE HARTVIGSEN AND DO VAN THANH

This paper describes the concept of the User Profile Web Service (UPWS) – an XML Web service that allows users to store preferences and personal data in a central profile and at the same time as it allows the user's different applications and services access to it. A realisation of this service would offer great enhancement of user mobility since it would allow for personal configuration of a user's services independent of time, place and device.

Anne Marie Hartvigsen (25) finalised her Master in Information Systems at Agder University College in June 2002, with the thesis "Studying Emerging Technologies in Telenor – Using Web Services to Provide Universally Accessible User Profiles". She also holds a Cand.Mag. in social sciences from the Norwegian University of Technology and Science (NTNU). Formerly a visiting researcher at Telenor R&D, working in the PANDA (Personal Area Network and Data Applications) research group, she is now employed as a systems developer at the Telenor spin-off Xymphonic Systems.

anne.hartvigsen@xymphonic.com

Do Van Thanh (44) obtained his MSc in Electronic and Computer Sciences from the Norwegian Univ. of Science and Technology (NTNU) in 1984 and his PhD in Informatics from the University of Oslo in 1997. In 1991 he joined Ericsson R&D Department in Oslo after 7 years of R&D at Norsk Data, a minicomputer manufacturer in Oslo. In 2000 he joined Telenor R&D and is now in charge of PANDA (Personal Area Network & Data Applications) research activities with a focus on SIP, XML and next generation mobile applications. He also holds a professor position at the Department of Telematics at NTNU in Trondheim. He is author of numerous publications and inventer of a dozen patents.

thanh-van.do@telenor.com

1 Background and Context

This section describes the motivation behind our work on the User Profile Web Service (UPWS) and gives an overview of related work in the problem area.

1.1 User Mobility and the World Wide Web

The background for this paper is two important trends. The first trend, mobility, concerns users' increasing ability to communicate anytime and anywhere. The huge popularity of mobile communication clearly shows how this kind of freedom and flexibility is rapidly becoming an important user demand. The other trend is the huge popularity of the World Wide Web. It has emerged as the by far most popular application on the Internet, and in harmony with the mobility trend can now be accessed not only trough PCs but also using smaller, mobile devices, like mobile phones and personal digital assistants. Users already have access to the Web almost anytime, anywhere and from any device, and there is every reason to believe that this trend will continue to evolve towards more and more flexible user options.

Today's users experience many different services. It may be software application services residing on the user's own devices, or it may be software application services that reside on the Web and that the user accesses through a Web browser. Users also utilise communication services such as voice communication and messaging – on both mobile and fixed networks.

1.2 Inhibitors of User Mobility

Today however, the user's mobility is limited by the fact that her data are bound to the devices and/or services that she uses. The first of these problems means that if a user needs to use *the same service or application on different devices* this will often require synchronisation of data between the devices (see Figure 1). Some products, e.g. Outlook, offer synchronisation facilities so that a user automatically can update data after changes have been made on one device. However this solution is still tedious, and if the user has more than two devices, the task of keeping all devices updated can be a complex one.

If the service resides on a Web server this reduces the problem, since data will be stored on a server and accessed from all the different devices accessing the service. However, the second problem still remains: Data are bound to the service, so that *different services that might use the same data will not be able to share them* (see Figure 2). Thus the user must maintain an extra set of data for each additional service, and all data changes must be made once per service. A typical example of this problem is having many

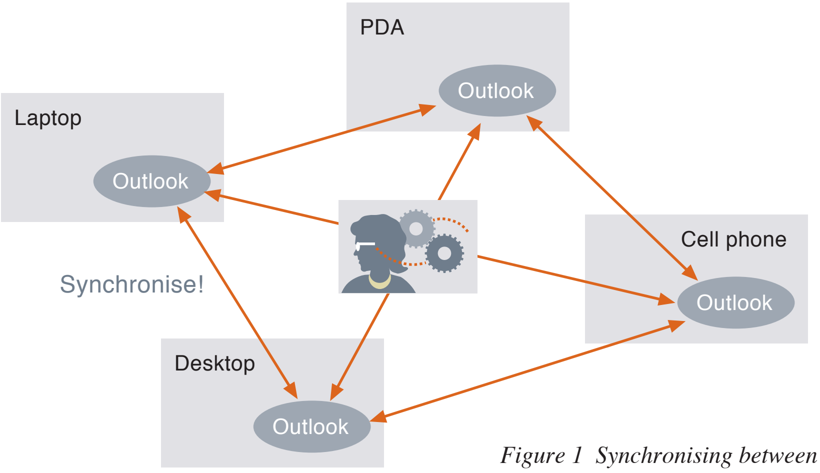


Figure 1 Synchronising between devices

sets of contact information – one on the mobile phone, one in the e-mail client, one in the chatting application, etc. Some of these data are overlapping, and allowing the different services to share the data would simplify data maintenance for the user and give the user more freedom and mobility.

All of this does not only count for user data, but also for preferences and settings – either application specific or more general (e.g. privacy preferences). There are several settings that would be convenient for the user not to have to configure again for each service to be used.

Also, it is important that services can be both application services and other, e.g. communication services, and thus this paper tries to look at a broad area taking in all categories in a user's daily use of IT services.

1.3 A Unified User Profile

Instead of binding user and configuration data to different devices and services, we propose here that the appropriate binding is between the data and the user. A user only needs one set of personal data, and once a user has configured an application or service with preferences and settings other services should be able to reuse as much as possible of these preferences and settings. The only way to achieve this goal is to capture all the relevant data in a user profile and then let the different services access this profile – as shown in Figure 3. These data can then be used for configuration of the service and as a repository of personal user data such as e-mail addresses, phone numbers, favourite bookmarks, etc.

This profile must be available as a service to the user's different services, and there are some crucial requirements associated with this:

- The profile must have one *general* part and one service *specific* part for each service
- The profile must be *extendable* so that a new service can be added instantly at any time
- The profile must be *easily* accessible from *any* device and service

These requirements are all crucial for the user profile to succeed and later in this paper we will explore how these requirements can all be fulfilled using XML Web Services. But first we will take a look at existing initiatives towards solving the problems we have described here.

1.4 Existing Solutions and Initiatives

Several initiatives try to solve different problems related to those described at the beginning of this section, and in this section we sum up some of the major activities that are going on.

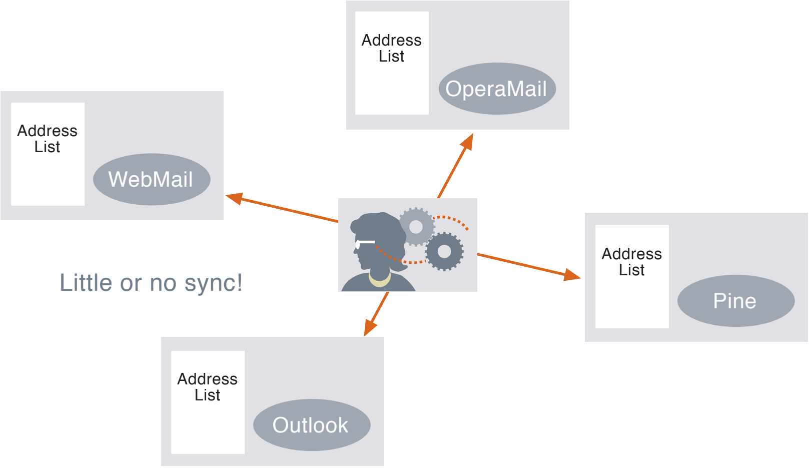


Figure 2 Keeping redundant data Stores

1.4.1 Virtual Home Environment

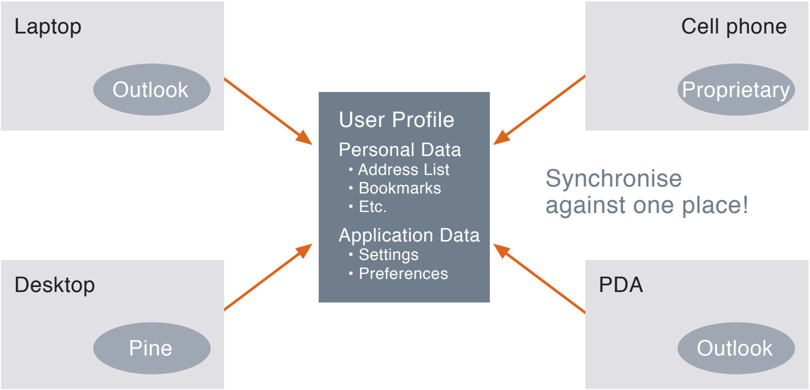
3GPP's Virtual Home Environment (VHE) initiative recognises the user's need to experience the same environment on the road that they have in their home or corporate environment. It is a part of the IMT-2000 and UMTS specification, and as such the environments targeted are limited to 3G mobile networks. Still only in the specification phase, VHE aims to let a foreign network emulate the behaviour of a user's home network.

1.4.2 CC/PP

There is a lot of work going on concerning the utilisation of user profiles to enhance services, and particularly in the area of mobility management for wireless and mobile networks, most often focusing on conveying situational context information about the environment, device and network, and only to a lesser extent personal preferences and needs.

The Composite Capabilities / Preferences Profiles (CC/PP) is an example of this perspective, focusing primarily on the capabilities of the network, devices and software and not so much on

Figure 3 Central user profile solution



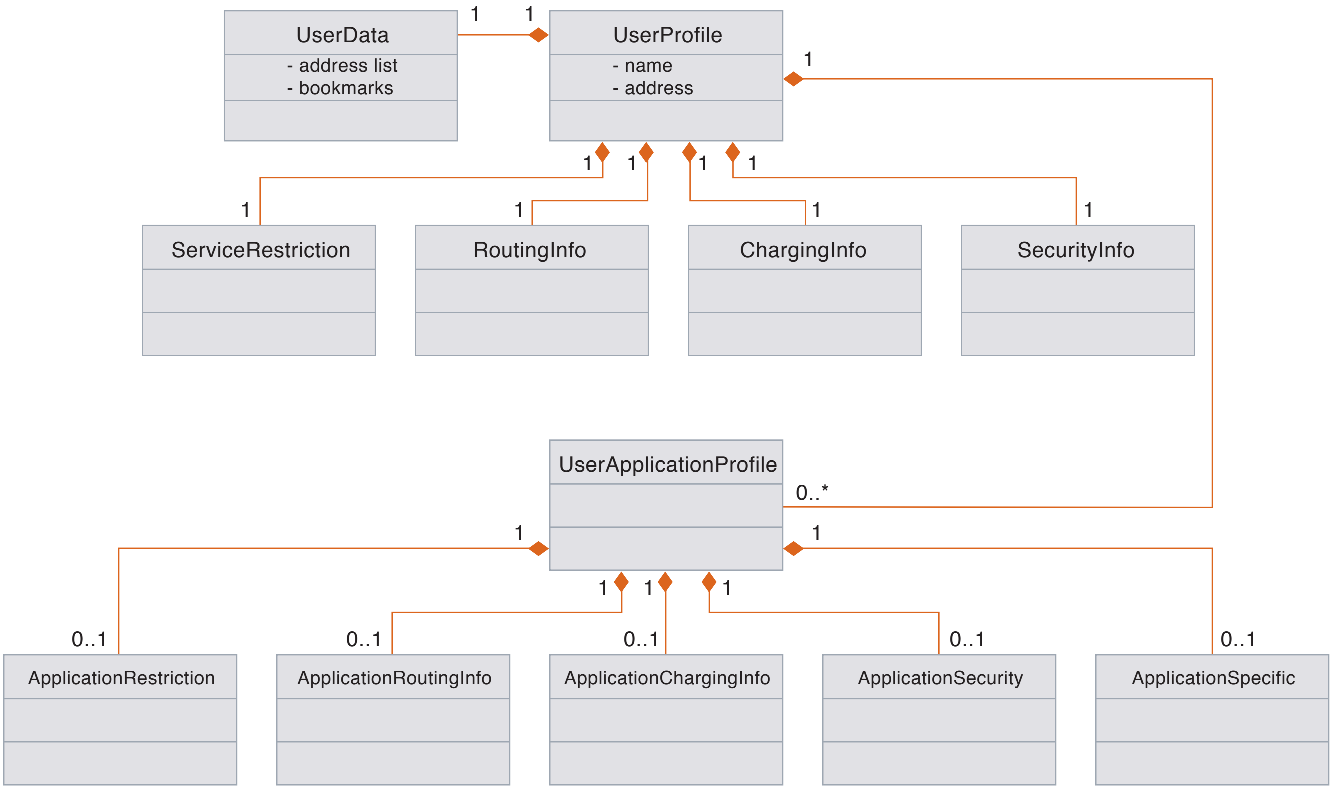


Figure 4 User Profile Data Model

user preferences. Software services on the Web may be accessed from devices with different capabilities – limitations in bandwidth and display may for example benefit from receiving a lightweight version of the service, while a user with a powerful PC and broadband would benefit from receiving every feature that the service can offer. CC/PP addresses this issue by defining a way to specify what a user agent (defined in terms of hardware and software platform and user agent application such as a Web browser) is capable of doing. Using CC/PP Web clients and services can negotiate service content and content delivery mechanisms to best fit the capabilities of the user agent. CC/PP also aims at defining user preferences, although this is not specified at this time. Thus CC/PP does not yet take care of user preferences in a satisfactory way.

1.4.3 P3P

The Platform for Privacy Preferences (P3P) focuses on user preferences in terms of privacy protection levels on Web sites. The user must install a P3P client on the device she is using to access the Web page, and then the client and the Web site can negotiate on the content. The ultimate P3P implementation would be integrated with the Web browser on the client side, but the solution is still bound to the device the user accesses a given service (Web site) with, and also to the client (Web browser) she uses.

1.4.4 .NET myServices

.NET myServices is the only existing solution similar to our UPWS. The aim of this service is to offer users the ability to store personal information in a central repository, which can be accessed by other applications such as an XML Web service. The concept relies heavily on .NET Passport service for single sign-on. However, our proposed framework goes beyond the scope of .NET myServices, since users have no opportunity to register different services as a part of their profile. Also, this project is still under development, and is not offered as a commercial service yet.

2 The User Profile Web Service

The data model in Figure 4 shows a general outline of the user profile, described in UML. It shows how each user must have a general UserProfile where user specific data such as name, address, contacts, etc. can be stored. In addition each user can have several UserApplicationProfiles where application specific data are stored. The content of ApplicationSpecific need not be understandable to any other than the service to which it belongs.

2.1 Realisation Alternatives

There are many possible ways in which the user profile could be realised and made available to the user's services and applications (from now on called *clients*).

The user could for example carry all the data with her. This could be done with the help of a smart card, but that would limit the solution to devices with the ability to read smart cards. The lack of card readers and security mechanisms makes this an unrealistic option. Besides the solution would depend on the user not losing or damaging her card.

Instead we propose that the profile should be stored in a place reachable from all devices and services in a programmatic way. The Internet and the World Wide Web seem like a natural choice, due to the ubiquity of the Web. By placing the profile on a Web server it is accessible from many fixed and wireless networks. There are however two problems that must be overcome. First of all the Web is more suited for retrieving data than submitting data back – which must be done to update the profile. It is possible to use HTTP Post but this is not an ideal solution. Secondly, Web access is traditionally done through a Web browser, while in our case we need the service to be programmatically accessible from the different clients.

One solution to these problems could be to use existing approaches to distributed computing, such as COM+, CORBA, IIOP and Java RMI. There are however some serious problems connected with these approaches too, because they put restrictions on the implementation details of the clients. First of all the client is often forced to use the same technology as the service – e.g. a CORBA service cannot be accessed using COM+. Secondly the chosen technology can limit possible limitations because they require the server and client to be implemented in a certain way. For example, for a client to use Java RMI it must be implemented in Java. The consequence of this is that the service will have to offer several different interfaces, or it will automatically deprive some clients of the possibility to access it without having to write a complex, expensive and non-reusable interface to the service.

Another problem with these technologies is that they use tight coupling, which makes the solutions less flexible and less suited for using the Web as the platform. Add to this that the technologies are quite complex and heavy to implement, plus the fact that firewalls are not handled very elegantly, and it is no wonder that universal services like our proposed user profile does not exist on the Web today.

2.2 XML Web Services

In this context XML Web services (see What is a Web Service in this issue of *Telektronikk*) constitutes a simpler technology for offering services on the Web. There is a couple of factors that make Web services the most suitable technology for realising our service:

- *Universal accessibility*. The XML Web services specifications offer a higher level of abstraction than the other technologies mentioned here (see How to Build a Web Service in this issue of *Telektronikk*). By using XML-schema definition of data types and allowing any language and platform to map these to its own representation it is possible to generate clients on any platform

- *Loose coupling*. This is the programming model most suitable on the Web platform, since services will never be 100 % reliable. It also ensures that changes on either side do not affect the other party's solution, as long as the interface is kept as it is. It also provides flexibility, since it becomes easy for a client to switch to a new service.

- *Simplicity*. There is no doubt that skilled developers are still needed, but the amount of development that needs to be done is smaller, since the technology is simpler. This of course also means reduced functionality, but the plan is to let the standard evolve as needs arise, instead of including everything from the beginning. This wisdom originates from the success of the Web, which has evolved exactly in this way. Therefore the SOAP protocol is designed with extensibility in mind.

- *Industry support*. With big players like Microsoft, IBM and BEA evangelising Web services and implementing Web services standards in their products, there is little doubt that the new technology is something more

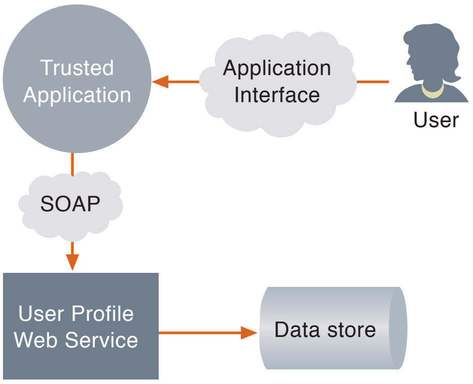


Figure 5 UPWS architectural overview

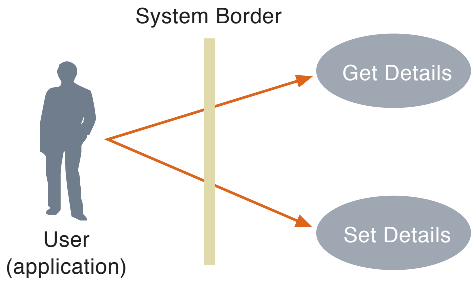


Figure 6 Use cases

than a fad that will be gone in a year or two. The broad support means that most vendors are already implementing the standards, so that developers on many different platforms can get readily generated interface code without having to write much more than the business logic. It also means that the goal of 100 % interoperability between different platforms is realistically obtainable.

As a Web service the user profile is accessible to any service that is connected to the Internet and able to talk and understand SOAP (on HTTP or any other transport protocol that the service provider chooses to support). Since retrieving and updating the profile will be quite a costly process in terms of time and bandwidth, the most suitable way to use the profile would be for the client to download the relevant parts of the profile at the beginning of a user session. If changes occur the profile can be updated at the end of the session or whenever a user decides during a session.

Mechanisms must be in place to authenticate both the user and the client of the profile service. Authentication can be as simple as basic authentication, but regarding that sensitive and personal data will be transferred it would also be desirable to use some kind of digital certificate and to encrypt sensitive data.

The end user can access the profile through a service that uses the Web service interface, since the Web service itself cannot have a user interface. The resulting architecture is shown in Figure 5. Thus the different clients will provide the user interface, and if it is necessary the user profile provider itself can provide a client to allow for easy administration of the profile.

The different clients will simply fetch user profile details and write back changed details. On a high level this results in the two use cases shown in Figure 6.

Set Details will have two different scenarios – one general and one for the first registration of the service.

Both scenarios require the user and client to authenticate, so that the UPWS can give them access to the appropriate parts of the user profile.

3 Use Scenarios

By using the UPWS a user can

- access an updated contact list from any service requiring names, e-mail addresses, phone numbers, fax numbers, etc.;

- access her favourite Web links from any Web browser on any device;

- automatically configure an application whose settings are stored in the profile;

- automatically fill in Web forms;

- add new services to her profile on the fly.

4 Implementation

There are several issues that must be resolved when implementing the profile:

- Storage and retrieval of user profile data;

- 3rd party application access to and modification of user profile data;

- User access to and modification of user profile data;

- Security;

- Providing the service;

- Consuming the service.

4.1 Storage and Retrieval of User Profile Data

When deciding how to store and manage XML data there are at least four alternative ways to do it: In a file system, in a native XML database, in a modified object database or in a relational database. File systems and native XML databases are appropriate for storing XML documents rather that XML data, the latter being more structured. Object databases have been identified as a natural storage for XML, but have yet to prove their usefulness in this context.

When storing structured data, it is generally recommended to use a relational database and then extract the information from the database to XML when needed. Structured data will benefit from the relational model when it comes to retrieval, searching and aggregation of data. While this forces a need to transform, or decompose the original XML document, decomposing an XML document to be stored to a relational database is not all that difficult. Also, many relational database vendors are implementing thin XML serialiser wrappers that enable them to generate XML documents on demand from relational data. So even if data will be coming in and going out as XML, e.g. in the form of SOAP messages, they can and should be stored as relational data.

One obvious advantage of using relational databases is that the technology is mature, stable and ubiquitous, and a whole range of tools exist for working with relational databases.

The service will extract the relational data into XML documents before providing applications and users access to them. It will also have to extract data from XML documents to update the database. Depending on the implementation, more or less of this functionality can reside in the database itself. Different implementations will have different ways of accessing data. The .NET development, for instance, will probably use ActiveX Data Objects for .NET (ADO.NET) to connect to data stores, while a J2EE implementation will probably use JDBC (see Figure 7).

4.2 Third Party Application Access to and Modification of User Profile Data and Functions

We have already concluded that XML Web services is the best alternative for realising the application.

The purpose of the User Profile Web Service is to give any other application access to the profile data. After getting the data from the profile, an application can configure according to the settings defined in the profile. The application should not have access to the whole profile, merely the parts relevant for the type of terminal and service in use, and according to user specifications. This will be specified in the ApplicationRestriction (see Figure 4). Based on the information retrieved from the profile service, the application can provide the user a personalised service. The application must also be able to update the UserApplicationProfile that belongs to the calling application.

Since the service will be provided as an XML Web Service, the API will be a SOAP interface, described in WSDL format. The third party application uses the WSDL file to understand the interface and implement a client proxy through which it can communicate with the Web service. Through this interface the application can both fetch the user profile data and return new or changed data to the user profile service (see Figure 8).

4.3 User Access to and Modification of User Profile Data

The most convenient way to provide the user access to her data is through the applications that will be using the user profile service. To ensure data privacy and integrity it is however important that different applications have limited read and write access to the data, otherwise a malicious application might change the user data without the user's knowledge and consent. That is why restrictions are defined in the ApplicationRestriction class.

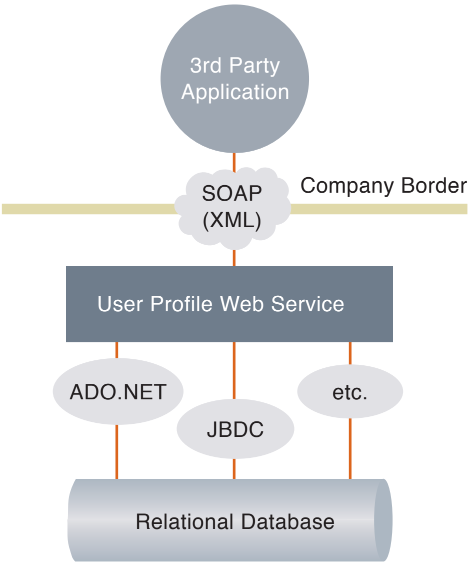


Figure 7 Data storage and access

Figure 9 shows how the user accesses the User Profile Web Service through the user interface of a third party application.

Applications must be trusted by the Web service in order to be allowed access. Depending on the trust level, different parts of the profile might be available to the different applications. At least one application should be completely trusted by the Web service, so that the user can manage all her data through this application. The user profile provider itself could offer this application, or a trusted third party could offer it.

Third party applications could either be installed on the user's device, or be accessible through Web interfaces such as HTML and WML. When delivered to the user as a Web application, an application could either provide one user interface (e.g. only HTML interface), or several user interfaces (e.g. both WML and HTML interface). In the case of several interfaces, these could be provided as different, static interfaces, or as one dynamic interface, displaying differ-

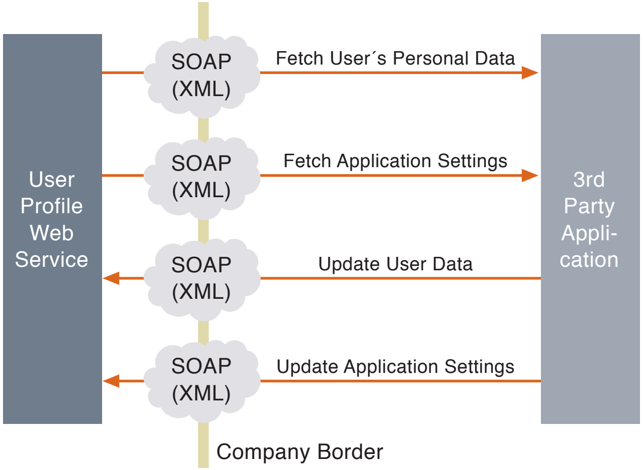


Figure 8 Third party application access to the user profile service

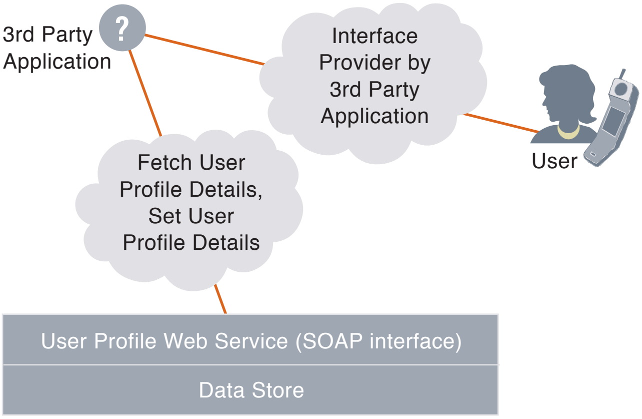

3rd Party Application
?
Interface Provider by 3rd Party Application
User
Fetch User Profile Details, Set User Profile Details
User Profile Web Service (SOAP interface)
Data Store

Figure 9 Application using the UPWS for personalisation of its services to the end user

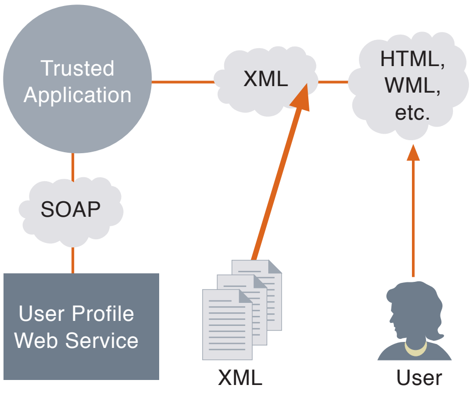


Figure 10 Dynamic user interface using XSL

ently depending on the accessing device. To achieve the last scenario the interface could be described in XML, and XSLT style sheets could be used for generating the most suitable interface display (see Figure 10).

4.4 Security

Security is extremely important in this service. Some of the data stored in the profile are personal, sensitive information that the user must be able to trust will not be read or altered by anyone else than the user herself and the applications approved by the user and the profile provider.

In addition to password authentication, security precautions would probably involve some use of digital certificates to further authenticate the client, and to encrypt sensitive data. This can be done using a secure key exchange protocol like Kerberos or HTTPS based on PKI key exchange. Applications should only gain access to relevant data, in order to protect the service from misuse.

Security is a complex issue, depending very much on the context in which it is implemented. Therefore it is not appropriate to specify a detailed security design here.

4.5 Providing the User Profile Web Service

When the data store and business logic of the service is implemented on the provider's platform, some kind of XML Web services framework is needed in order to make the service available to others. This can be done either with Microsoft's .NET framework, or with a J2EE implementation that supports XML Web service protocols. As soon as this is done, the service is available on the Web. The service can be invoked by sending the appropriate SOAP messages to the appropriate URIs, and what these are is specified in the WSDL document, which is also available on the Web.

Next, potential users of the Web service (third party applications) must be aware of the service. One way is telling partners and other chosen actors about the service, and where they can download the WSDL document. While a good and secure way of testing the service to begin with, it does not meet the requirements of global ubiquitous accessibility from any application. To achieve this the service must be published in a global registry that can be searched by possible clients. XML Web Services specifications therefore also include the UDDI registry for universal registration and discovery of XML Web Services.

Publishing the service to the UDDI registry does not only mean that the service must be 100 % stable and available, it also means that it must implement security precautions ensuring that the service will only be accessed by applications that are authenticated and authorised to use it. A UDDI broker might offer some security and guarantees, but the service provider must also consider which security mechanisms that are necessary to implement in the service.

Users of the service can either be approved automatically online (e.g. with the help of authentication and contract services provided by the broker), or contracts can be negotiated manually, "off-line". Which model is best suited is a mix of security concerns, business model (e.g. are the users supposed to pay for using the service) and available solutions (e.g. how much is already offered by the broker).

4.6 Consuming the User Profile Web Service

As discussed in the previous section, a client can discover the service through a UDDI registry or by direct communication with the provider. After discovering the service and obtaining the necessary access keys (passwords, certificates or other things that the XML Web service might require), the client must build a proxy to communicate with the Web service. This is generated automatically from the information in the WSDL file and integrated into the client's software project. Now the client can call the proxy methods as if they were local methods. How these methods should be called and how the results should be handled is entirely up to the client application.

If the client for instance is granted access to the user's bookmarks, it will probably see a method called something like getBookmarks. By calling this method the client would obtain the user's bookmarks. The client application is then free to do with them whatever it wants, but it would probably choose to store them in the user's local bookmarks folder. If the client application were the Internet Explorer Web browser, it would probably store them in the "Favourites" folder belonging to the user.

4.7 UML Application Design

The Object Management Group (OMG) specifies a modelling language called Unified Modelling Language (UML) (Object Management Group, 2001), which aims at enabling implementation independent design of applications. This modelling specification has become very popular and is implemented by a number of vendors, e.g. Rational and Telelogic.

To provide an overview of the user profile service design, three kinds of UML diagrams are shown here. First some use cases are presented to show how other applications will interact with the service. Then data objects are shown in a class diagram. Lastly the use cases are elaborated in sequence diagrams, depicting how the different data objects will interact in each use case.

4.7.1 Use Cases and Class Diagram

The use cases show how actors may interact with the user profile service. In use case diagrams the actor, or user, can be either an end user, or another system (application) using the service.

Figure 6 shows a simple use case diagram. It illustrates the main use of the system, namely reading user profile data, and saving new or modified data.

To be more accurate it is necessary to design a data model that shows what kind of data the user profile service will contain.

Telecom user profiles would constitute a good starting point for the service. But telecom user profiles as defined have many limitations. The user profile is intended for customisation of the main service, namely voice communication or telephony, and its supplementary services, e.g. call forwarding, call answering, etc. It is also stored within the operator's system and is not available to third party applications or services. In order to allow the users access to multiple applications and services anytime, anywhere and on any terminal, the content of the user profile needs to be extended to fulfil the following requirements:

- For each user the user profile must be expandable to incorporate the preferences and settings for any additional application or service that the user requires.

- For each application the user profile must contain the information necessary for the presentation of the application on the terminal types requested by the user.

- For each application the user profile must contain usage restrictions.

- The user profile must incorporate personal data such as address book, telephone list, bookmarks, etc.

In accordance with these requirements, a data structure for the user profile is proposed in UML (Unified Modelling Language) in Figure 4.

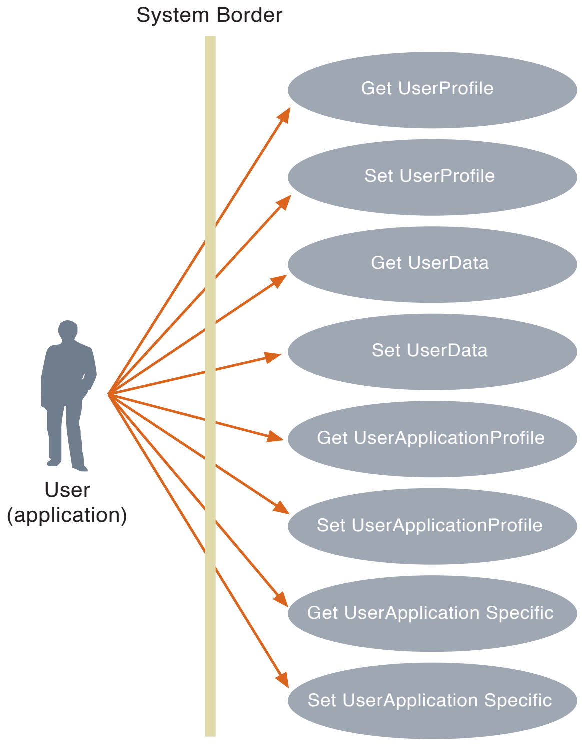
The *UserProfile* has six components: *UserData*, *ServiceRestriction*, *RoutingInfo*, *ChargingInfo*, *SecurityInfo* and *UserApplicationProfile*.

UserData contains for example addressList, bookmarks, etc.

ServiceRestriction has attributes such as:
- Roaming restriction
- Time restriction
- Credit limit
- Maximum number of terminal addresses for group registration for incoming applications
- Incoming screening
- Outgoing screening
- List of subscribed services

RoutingInfo has attributes such as
- Forwarding activation status
- Registered terminal address for incoming applications
- A linked-registered terminal address
- Default terminal address for incoming applications
- Routing by applications originating area
- Routing by calling party identity
- Time-dependent routing
- Routing on "busy" condition
- Routing on "no answer" condition
- Default duration (or number of calls) for incoming applications registration

Figure 11 Detailed use case diagram


System Border
Get UserProfile
Set UserProfile
Get UserData
Set UserData
Get UserApplicationProfile
Set UserApplicationProfile
Get UserApplication Specific
Set UserApplication Specific
User
(application)

ChargingInfo has attributes such as

- Default charging reference location
- Charging option selected
- Temporary charging reference location
- Advice of charge activation status

SecurityInfo has attributes such as

- Authentication procedures subscribed
- Security options subscribed
- Type of authentication procedures activated
- Max number of failed authentication attempts
- Password

The *UserProfile* may contain zero or more *UserApplicationProfiles*. The purpose of the UserApplicationProfile component is to enable customisation of an application. For each application (run in a service session), there may hence be assigned zero or one *UserApplicationProfile*. The *UserApplicationProfile* may contain zero or one *ApplicationRestriction*, *ApplicationRoutingInfo*, *ApplicationChargingInfo*, *ApplicationSecurityInfo* and *ApplicationSpecInfo*. It is therefore possible to specify the restrictions, routing, charging, and security options for each application.

The *ApplicationSpecInfo* is a component that contains application specific data. Greater flexibility is achieved in this way. An application is however not required to have its own *UserApplicationProfile*. For applications that do not have their own profile the main *UserProfile* is applied at the initiation of the application. The user should have access to a service that permits him to interrogate and modify some of the attributes of his *UserProfile*. His access rights are linked to and used by an access control procedure.

While all these classes are important in a working telecom user profile application, some of them fall outside the scope of this thesis. Focusing on the usage of XML Web services in order to provide access to data, the classes most important in this setting are UserProfile, UserData, UserApplicationProfile and UserApplicationSpecific, since these must be accessible from the third party application.

RoutingInfo, ChargingInfo, ApplicationRoutingInfo and ApplicationChargingInfo deal with routing of calls and the operator’s basis for billing the customers, and will not be discussed further here. For the sake of simplicity these classes are omitted in the rest of the design.

ServiceRestriction in telecom profiles is mainly used for time- and place restrictions, and the ApplicationRestriction can serve the same purpose. However, ApplicationRestriction can also be used for restricting the application’s privileges; i.e. regarding access to the user profile data such as bookmarks and personal data. Security and ApplicationSecurity can be used for authenticating and authorising the user and application.

Figure 11 shows a more detailed use case diagram. The user of the service must be able to fetch and alter some of the objects in Figure 4, namely the UserProfile, UserData, UserApplicationProfile and UserApplicationSpecific. The other classes are for internal use, and will not interact directly with the user.

In the next section, these use cases are elaborated into sequence diagrams, showing what must happen on the inside of the system once a use case occurs.

4.7.2 Sequence Diagrams

Sequence diagrams are drawn for each of the use cases, and often as several scenarios for each use case. The purpose is to show how the system’s objects must interact when an actor is using the system.

Figure 12 and Figure 13 show the sequence diagrams for get and set UserProfile. After the user is authenticated, application restrictions are checked to prevent the application from altering data that this particular application does not have write access to. If the application is allowed, the user profile object will be updated accordingly and a confirmation is sent back to the calling application.

Get and set UserData are similar to get and set UserProfile, and are only shown in the appendix.

Figure 14 shows how UserApplicationProfile is fetched. Each application should have access to

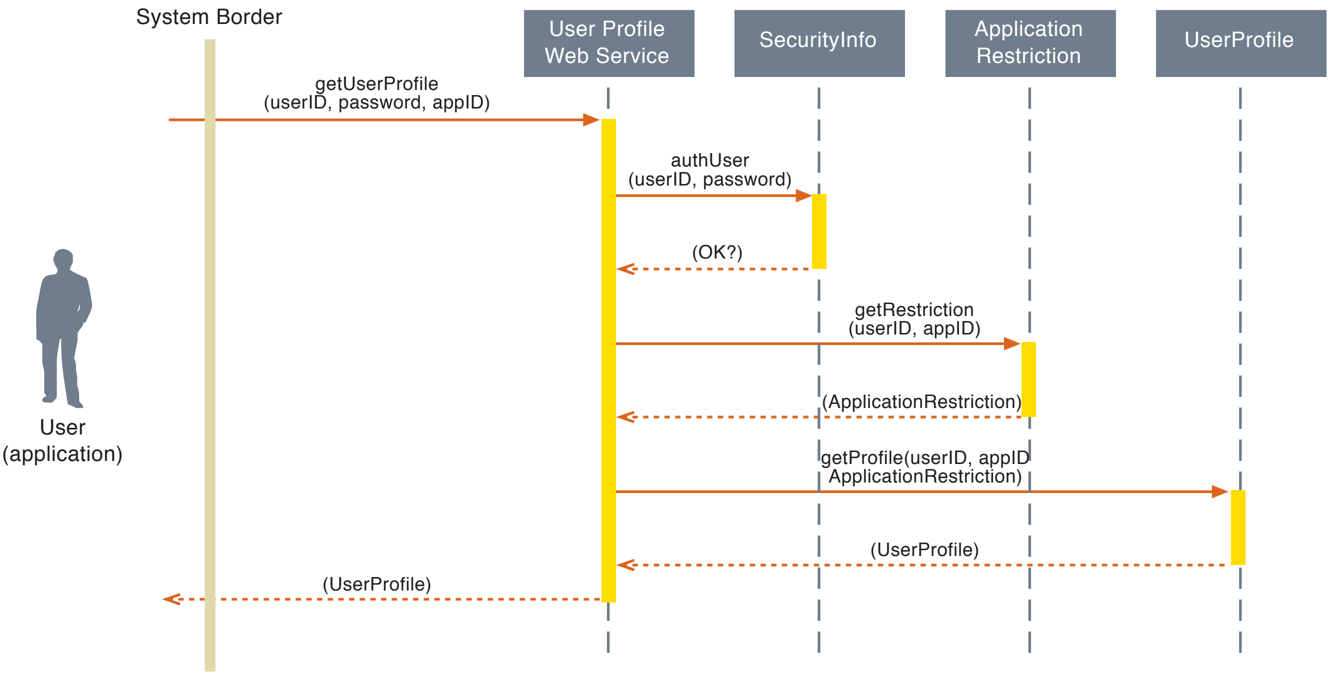


Figure 12 Sequence diagram: get UserProfile

all data in its specific UserApplicationProfile object, so there is no need to check restrictions. Otherwise get and set UserApplicationProfile is similar to get and set UserProfile.

Get and set ApplicationSpecific are similar to get and set UserApplicationProfile and are only shown in the appendix.

The preceding diagrams have shown the most common use case scenarios. Another scenario for the use cases is when the user or application is not yet registered in the user profile.

Figure 15 shows how a New Application scenario could be handled.

The procedures for registering a new user will not be drawn here, since it depends on how the company offering the user profile chooses to include new users. Telecom companies could offer the service to existing customers, since

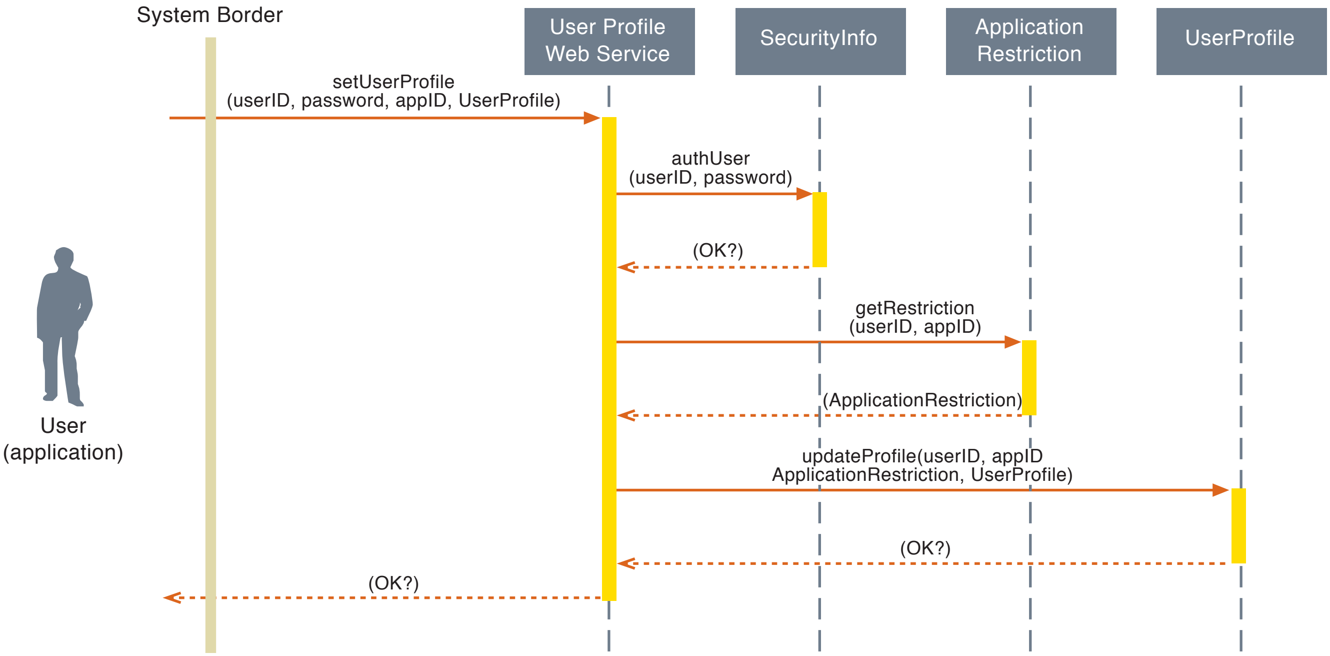

System Border
User Profile Web Service
SecurityInfo
Application Restriction
UserProfile
setUserProfile (userID, password, appID, UserProfile)
authUser (userID, password)
(OK?)
getRestriction (userID, appID)
(ApplicationRestriction)
updateProfile(userID, appID ApplicationRestriction, UserProfile)
(OK?)
(OK?)
User (application)

Figure 13 Sequence diagram: set UserProfile

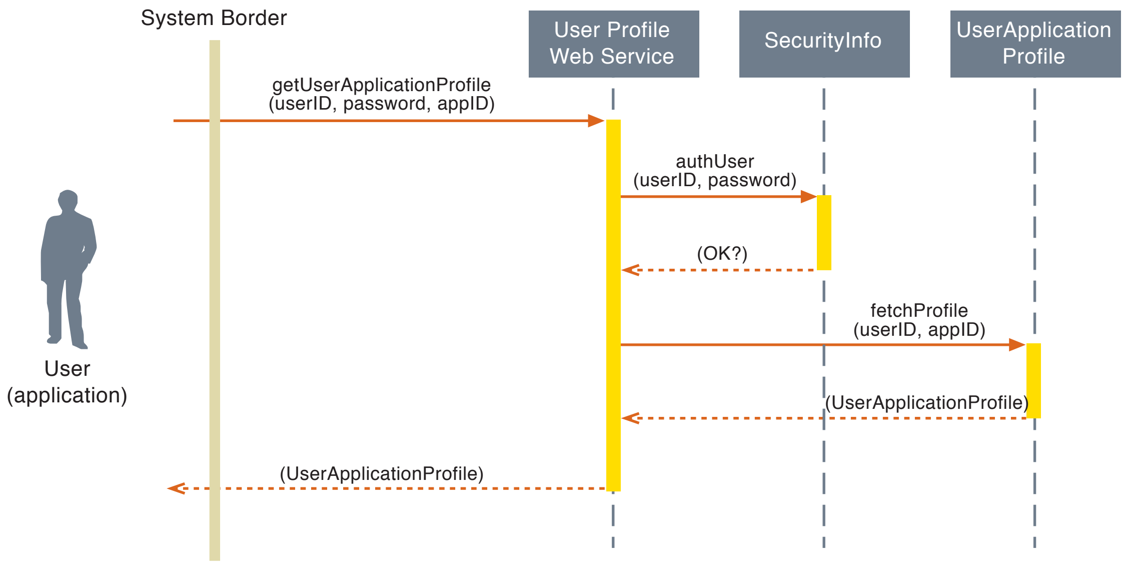


Figure 14 Sequence diagram: get UserApplicationProfile

data about them would already be stored. In that case, an activation procedure where the customer agrees on the service terms may be all that is needed to register the new user.

5 Business Model

An actor that can be trusted both by the user and the service provider must host a service like this. The user needs to be certain that her data will

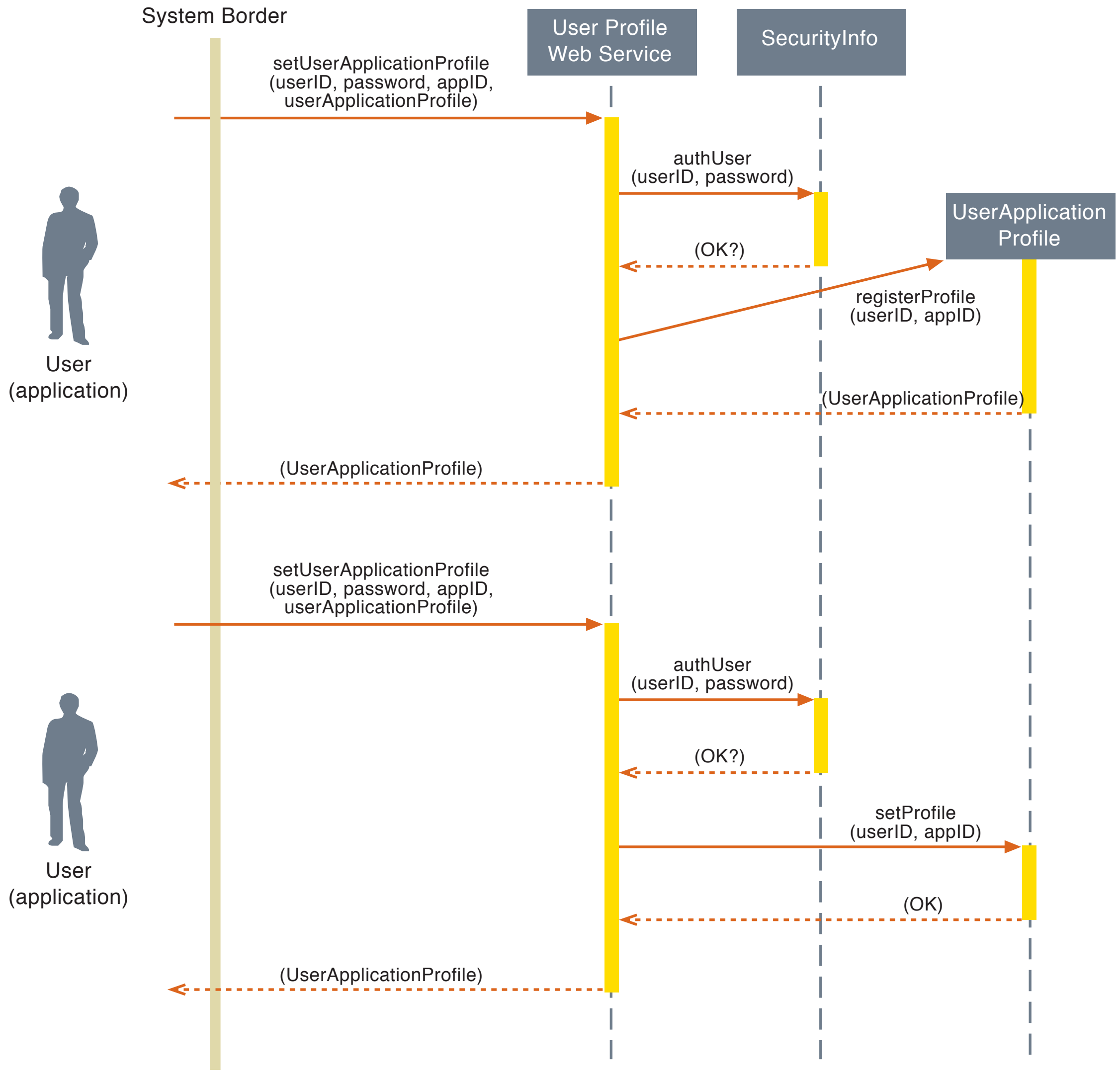

System Border
User Profile Web Service
SecurityInfo
setUserApplicationProfile (userID, password, appID, userApplicationProfile)
authUser (userID, password)
UserApplication Profile
(OK?)
registerProfile (userID, appID)
User (application)
(UserApplicationProfile)
(UserApplicationProfile)
setUserApplicationProfile (userID, password, appID, userApplicationProfile)
authUser (userID, password)
(OK?)
setProfile (userID, appID)
User (application)
(OK)
(UserApplicationProfile)

Figure 15 Sequence diagram: set UserApplicationProfile, new application scenario

not get lost or accessed by unauthorised parties. The service provider on the other hand would also need to be sure that the service specific data would not be altered without permission from the service, and that the user data are correct. Telecom providers already maintain extensive user profiles to configure their communication services, and the user data are validated in the sense that correct information about real users is ensured. As such telecom companies or other companies with this kind of customer information, and also government administrations have a starting point for providing such a service. To secure customer rights it is possible to imagine that several such institutions would cooperate in providing the user profile.

Several payment schemes can be imagined. First of all it is a question of who should pay for the service – the user, the service provider or both. Secondly it would be possible to charge either per use or per time unit (e.g. monthly fee). Probably a mixture of the different possibilities could be offered, but we will propose one model here.

We assume that a telecom company chooses to offer a user profile service to its customers. These customers already pay a monthly subscription fee, and the service could be offered as added value to certain subscription types, or in return of an additional monthly fee. To ensure that the service will actually be useful to the customers, any application can access the service for free. Subscribing to this service then, the user would be able to use it with any of her services that have chosen to implement the service. Since clients are easy to make, many service providers may consider this a cheap way to add significant value to the service.

6 Issues to Resolve and Further Work

This paper has merely presented a draft of a possible Web service: The user profile Web service. There are, however, several issues related to the user profile that need to be resolved before a real and commercial Web service can be offered.

6.1 Legal Considerations

Storing user data demands a permit from authorities, and the ability to obtain such a permit is an important issue to consider. A provider must also be prepared to prove that user data will not be available to third parties without the user's full and cognisant consent.

6.2 Privacy Considerations and User Acceptance

Apart from the legal considerations there is a distinct ethical issue about storing and managing people's personal data. In the kind of service provided here this issue becomes even more visible, since the scope is to include as many of the user's data as possible. Users are concerned about privacy, particularly on the Internet (Cranor et al. 1999). Although the goal is to provide the user and the user's services with a valuable service that will make data management easier on all parts, it is easy to see how motives can be questioned, and users are and should be sceptical towards putting all their data in the hands of one, big actor. It becomes extremely important that the provider is regarded as a fair and trusted actor. This is yet another reason for several actors to ally – the notion of several actors working together may increase credibility.

6.3 Technical Considerations

Our tentative data model is not complete, and particularly there is a need to discuss which data should be user specific and which should be application specific. One should also consider the use of application categories to enable sharing of data between different applications in the same application domain. There are also many implementation specific aspects that must be discussed, such as security architecture and which platform to use.

7 Conclusion

We have described an application called the User Profile Web Service. This application allows users to store preferences and personal data in a central profile, with the purpose of granting different services access to it. In this way the proposed application allows the user's services to self-configure.

The enabling technology for the User Profile Web Service is XML Web Services. Using XML Web Services allows services on any platform and device to take advantage of the user profile, as long as they can use the Web services protocols and compliant transport protocols (e.g. HTTP or SMTP). A realisation of this service would offer great enhancement of user mobility since it would allow for personal configuration of a user's services independent of time, place and device. However, the work is in an early stage and there are several issues that need to be resolved first – both on the technical level and business considerations.

8 Further Reading

Cranor, L, Reagle, J, Ackerman, M S. (1999). *Beyond Concern : Understanding Net Users' Attitudes About Online Privacy*. AT&T Labs-Research Technical Report TR 99.4.3. August 16, 2002 [online] – URL: http://www.research.att.com/resources/trs/TRs/99/99.4/99.4.3/report.htm