An introduction to Web Services

Paul Muschamp

This paper provides an introduction to Web Services. A brief history of Web Services is given, leading into an overview of the key Web Services technologies. The paper describes the way in which the IT industry has embraced Web Services and the uses to which the technologies are currently being applied. It concludes with a view of the role of both the key developers and standards bodies in the creation of Web Services and discusses the ways in which this process needs to be supported in order for Web Services to evolve.

1. Web Services basics

Web Services is a phrase used to describe the way in which services can be exposed and used on a network, built around the use of technologies such as extensible mark-up language (XML).

Web Services concern the way in which software communicates. Software can come in many forms from a simple script on a personal computer, to an application on a networked server, through to a large operational support system running on a mainframe computer. We can view these scripts, applications and software systems as software components. Consider the following characteristics of such a software component:

- it can **describe** itself — so that other components can understand the functionality it offers and how to access that functionality,
- it can allow other components to **locate** it — so it can be used when required,
- it can be readily **invoked** whenever another component wishes to use its functions.

If such a software component were installed in a network, its services could be used by other software components. Within the last three years, a number of technologies have matured and become world-wide standards to allow software components to be deployed in this way. These components are said to be providing a 'Web Service'. An example is a Web Service that returns a credit rating when provided with a user's ID, or an order handling system that provides user and usage data to a billing Web Service which then returns the user's bill.

Web Services can also be integrated together to provide greater value-add. For example, a travel management Web Service may make use of the capabilities of Web Services providing car hire, hotel bookings and flight reservations. Or a video-on-demand Web Service may make use of a communication service that delivers the video stream (which may present itself as a Web Service).

So, why are such services called Web Services? The reason is that when exposing the capabilities of software components, we are essentially exposing software services. The goal is for these services to be widely available over the World-Wide Web (more strictly we really mean the Internet) — hence the term 'Web Services'. The concept of exposing software services is built around a service-oriented architecture [1—3].

As well as software services, Web Services can also be extended to communications networks. We have seen examples in recent years of exposing network functionality as software services [4, 5]. One of the challenges for communications service providers is to use Web Services to take best advantage of the convergence of software and network technologies, as shown in Fig 1.

In recent years, we have seen this convergence get closer to reality with the World-Wide Web and the Internet, both now uniquely linked with each other, despite the fact that the former comes from the software world and the latter from the network world. Web Services is the point at which the two worlds converge. Web Services allow services to be created and exposed with a common interface. These services can

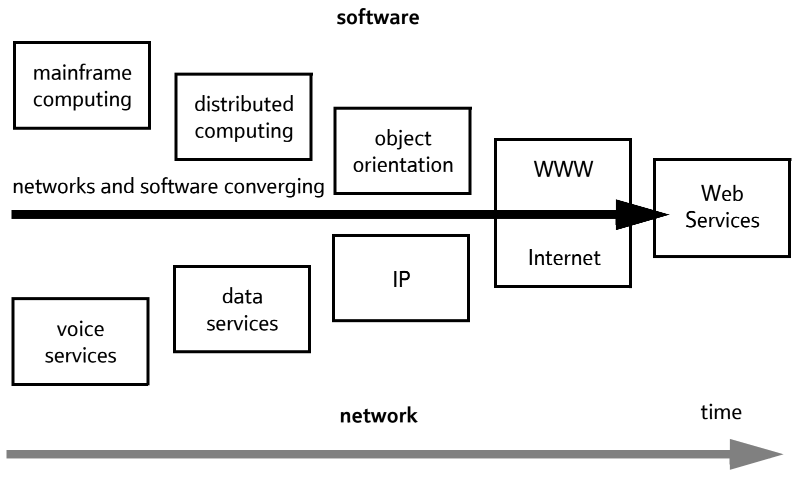

Fig 1 The convergence of software and network technologies.

be software-based or network-based and can be readily integrated together to form added-value services.

2. History of Web Services

Web Services are services that can communicate with other services over a network, using a set of standard technologies. In order for this to happen, a number of key objectives must be met. The technologies must be:

- platform agnostic — that is, infrastructure from all vendors must support the same specifications,
- language agnostic — that is, a service written in one language must implement the specifications for Web Services in the same way as that written in a different language,
- free from restrictive IPR terms — that is, the developers of the technologies must have widespread adoption as their primary goal.

The initial protagonists in the development of Web Services — IBM and Microsoft — had these objectives in mind when the ideas for Web Services were in their infancy. The development of the World-Wide Web (WWW) in the 1990s showed how the IT and communications industry can work together to create a common framework including defining basic protocols such as TCP and HTTP. However, it was the creation of XML that paved the way for Web Services.

As a widely heralded, platform-independent standard for data description, XML was a logical starting point for the job of standardised service-to-service communication. XML became a standard in February 1998, when the World Wide Web Consortium (W3C) announced that XML 1.0 had reached draft recommendation status.

Having developed the basic protocols, the next stage was to agree on a specification for a standardised message-passing protocol based on XML. Microsoft had been developing the simple object access protocol (SOAP) — it was platform agnostic, flexible, and general-purpose. In March 2000, IBM publicly backed Microsoft's SOAP and started working with the company to develop SOAP 1.1, and by the summer of 2000, SOAP was gaining wider acceptance.

At the same time, IBM and Microsoft were each working on a way to programmatically describe how to connect to a Web Service. After some discussion, protocol proposals from Microsoft and IBM merged. IBM contributed Network Accessible Service Specification Language and Microsoft offered both Service Description Language and SOAP Contract Language. In late 2000, a merged specification, Web Services Description Language (WSDL), was announced.

With SOAP and WSDL, companies could create and describe their Web Services. In March 2000, IBM, Microsoft, and Ariba started working on a solution to provide a means of discovering available Web Services, and in September 2000 Universal Description, Discovery, and Integration (UDDI) version 1.0 Specification was announced.

With SOAP, WSDL, and UDDI in place, the standards to create Web Services had arrived, and, by the end of 2000, the major IT software infrastructure vendors announced their commitment to Web Services. Oracle, HP, Sun, IBM, BEA and Microsoft, an unlikely — and thus impressive — alliance, stated their intention to support and deploy the Web Services standards in their products.

The next section addresses the way in which these software technologies work.

3. Web Services technologies

As we saw in section 1, a Web Service is one that can **describe** itself, be **located**, and be **invoked**. The Web Service description is provided by WSDL, it can be located by using UDDI, and its functionality invoked using SOAP. These three technologies are fundamentally built upon the common data description standard XML, all of which are described below.

3.1 Extensible mark-up language

XML is a standard specification for defining the content of a computer message. If a software application writes its output in XML and another application is capable of interpreting XML, then it can read the output and act on it. The way XML does this is via the use of tags. To illustrate this, if you go to the 'View' menu of an Internet browser and select 'Source' (in Microsoft Internet Explorer) or 'Page Source' (in Netscape Navigator), you will see the source HTML of the Web page you are reading. The content of the Web page will be displayed within a number of tags. Each tag will have a name, enclosed within opening and closing angled brackets, e.g. <head>. The tags 'mark up' the content — hence why HTML and XML are called mark-up languages. Figure 2 gives an example of content, marked-up using the specification for XML.

```
<customer>
   <name>Fred Smith</name>
   <address country="UK">
      <street>93 Example Avenue</street>
      <city>Ipswich</city>
      <region>Suffolk</region>
      <postcode>IP50 9BT</postcode>
   </address>
   <status purchases="2" last-purchase="12-05-99"/>
</customer>
```

Fig 2 Example XML mark-up.

This is displaying the content for a customer, as seen by the opening tag `<customer>` and the closing tag `</customer>`. The other tags are similarly easy to understand, which gives XML one of its main attributes — XML content can easily be made human-readable. More importantly, however, because the content is defined within the tags, which have a strictly defined structure, they are also machine-readable.

The importance of XML to the computing world can be compared to the importance of natural language to humans. If I wish to convey meaning to another person in a message I give to them, and I wish them to correctly interpret my meaning (and perhaps act on it), then the two of us need to use a commonly understood language. The language used must have form (spoken, written, etc), structure (words, sentences, etc), syntax (the grammatical arrangement), and semantics (meanings and connotations). Two computers communicating have the same requirement.

If the two communicating entities use a different language, the only way they can communicate is via an intermediary that does a translation between the two languages. In a complex computer system, where there are a number of communicating entities, the use of a common language can drastically reduce the cost of development and maintenance as well as improving its performance.

The software industry has spent countless millions developing software that undertakes translations between different languages. Many attempts have been made to develop and encourage the use of common languages between computer applications, most of which have failed. However, the success of the World-Wide Web, and the use of HTML as a common language, has been one of the most significant. It is the success of HTML which has encouraged the use of mark-up languages in general, and which has led to the rapid take-up and success of XML.

3.2 Simple object access protocol

With XML we have a common way to represent the content of a message sent between one software component and another. SOAP builds on this by providing the means by which one software component can invoke the functionality of another, using message passing between the two as the means of invocation.

The SOAP protocol includes the following:

- an envelope that defines a framework for describing what is in the message and how to process it,
- a set of encoding rules for expressing instances of data types within the message,
- a convention for representing the way in which the procedures (or methods) of the software component can be called, and their responses,
- a binding convention for exchanging the message using a communications protocol.

Version 1.1 of the specification for SOAP was issued by the W3C in May 2000 [6] and includes the binding convention for the HTTP protocol. Although SOAP messages can be transferred using other protocols, the fact that HTTP was chosen reflects the intention to make the protocol available over the Internet.

SOAP uses a request-response mechanism in which one software component makes a request to another software component which then provides a response. Both request and response are transported in the form of XML documents. An example of a SOAP request can be seen in Fig 3. Note, the line numbers are inserted for reference and would not appear in the request itself. This SOAP request is being made to a software

```
1. POST/StockQuote HTTP/1.1
2. Host:www.stockquoteserver.com
3. Content-Type: text/xml; charset=“utf-8”
4. Content-Length: nnnn
5. SOAP Action: “Some-URI”

6. <SOAP-ENV:Envelope
7.  xmlns:SOAP-ENV=“http://schemas.xmlsoap.org/soap/envelope/”
8.  SOAP-ENV:encodingStyle=“http://schemas.xmlsoap.org/soap/encoding/”>
9. <SOAP-ENV:Body>
10.   <m:GetLastTradePrice xmlns:m=“Some-URI”>
11.     <symbol>STOCKSYMBOL</symbol>
12.   </m:GetLastTradePrice>
13.  </SOAP-ENV:Body>
14. </SOAP-ENV:Envelope>
```

Fig 3 Example SOAP request.

component that returns a stock trading price when provided with a valid stock identifier.

Line 1 defines that the message is an HTTP POST. Line 2 defines the URL of the host provider of this service. Lines 3 and 4 define that that the message is textual of length `nnnn`. Line 5 defines the URI of the action that this request is making. Lines 1 to 5 define the SOAP header, with lines 1 to 4 being HTTP.

Lines 6 to 14 define the SOAP envelope which contains the main parts of the protocol — defining the messaging framework, the data types and the procedures (methods). Within the envelope, lines 7 and 8 define the locations of the namespace and encoding rules. Lines 9 to 13 define the body of the SOAP message. Within the body, line 10 defines the method that is being called by the SOAP request — in this case a method that returns a stock trading price for a given stock (as defined by the `STOCKSYMBOL` shown on line 11).

It can be seen that once the SOAP header plus the namespace and encoding URLs are removed, the SOAP request consists of a straightforward call to a method. Since the SOAP request is based on XML, it can be made both machine readable (by using XML's structural and encoding rules) and human readable (by using easily understood English language descriptions for the method calls and data types). Moreover, the request is embedded within an HTTP binding, allowing the request to be made across the Internet. The SOAP response message would have a similar format, as seen in Fig 4.

The main difference between the request and the response are shown in the first line (in that this is now bound to an HTTP response) and in the line showing the `<Price>` (this shows 34.5 which would be an example of the trading price for the stock in question).

SOAP provides a simple but powerful means for one software component to invoke action on another via the use of message interactions. The specification for SOAP is a world-wide standard, administered by the W3C, currently at version 1.2 [7]. All the main software platform vendors have agreed to implement this specification, which means that a software component installed on one type of platform can communicate using this method with another component on any other type of platform (see Mockford [8] and Calladine [9] for interoperability issues). It should be noted that references to SOAP no longer describe it as being an abbreviation for simple object access protocol — it is now simply SOAP.

With XML as the standard messaging format and SOAP as the means of invoking functionality, the next

```
HTTP/1.1 200 OK
Content-Type:text/xml; charset=“utf-8”
Content-Length: nnnn

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV=“http://schemas.xmlsoap.org/soap/envelope/”
 SOAP-ENV:encodingStyle=“http://schemas.xmlsoap.org/soap.encoding/”/>
 <SOAP-ENV:Body>
  <m:GetLastTradePriceResponse xmlns:m=“Some-URI”>
   <Price>34.5</Price>
  </m:GetLastTradePriceResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Fig 4 SOAP response message.

link in the chain is the way in which Web Services are described.

3.3 Web Services description language

HTTP, XML and SOAP provide the means for one software component to invoke the functionality of another over the Internet. In fact, these technologies can be used to integrate software components over any network that allows HTTP. In order to undertake this integration, the following are also needed:

- information on all available functions, including their calling parameters,
- data type information for all XML messages, including the value specifications,
- binding information about the specific transport protocol to be used,
- address information for locating the specified service.

A software developer undertaking an integration of software components might use the documentation for each component to get this information. Extracting this information for a complex integration would be time-consuming and prone to human error, thus increasing the cost of integration. Ideally, this could be undertaken automatically by integration software. WSDL has been developed for this purpose.

A WSDL file is an XML document that provides the information listed above about a software component. Using WSDL, a software component can integrate with any of the available functions of a Web Service. With WSDL-aware tools, this process can be entirely automated, enabling applications to easily integrate new services with little or no manual code.

An example of a WSDL file is shown in Fig 5. This file specifies the items listed above for a weather service providing the temperature at a given destination.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="WeatherService"
 targetNamespace="http://www.townweather.com/wsdl/WeatherService.wsdl"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tns="http://www.townweather.com/wsdl/WeatherService.wsdl"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <message name="getWeatherRequest">
  <part name="town" type="xsd:string"/>
 </message>
 <message name="getWeatherResponse">
  <part name="temperature" type="xsd:int"/>
 </message>

 <portType name="Weather_PortType">
  <operation name="getWeather">
   <input message="tns:getWeatherRequest"/>
   <output message="tns:getWeatherResponse"/>
  </operation>
 </portType>

 <binding name="Weather_Binding" type="tns:Weather_PortType">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getWeather"> <soap:operation soapAction=""/>
   <input> <soap:body
    encodingStyle="http://schemas.xmlsoap.org/soap.encoding/"
    namespace="urn:examples:weatherservice" use="encoded"/>
   </input>
   <output> <soap:body
    encodingStyle="http://schemas.xmlsoap.org/soap.encoding/"
    namespace="urn:examples:weatherservice" use="encoded"/>
   </output>
  </operation>
 </binding>

 <service name="WeatherService">
  <documentation>WSDL File for Weather Service</documentation>
  <port binding="tns:Weather_Binding" name="Weather_Port">
   <soap:address location="http://localhost:8080/soap.servlet.rpcrouter"/>
  </port>
 </service>
</definitions>
```

Fig 5 Example WSDL file.

The first section defines the various namespaces required including the URL of the WSDL file itself — http://www.townweather.com/wsdl/WeatherService.wsdl — this is the place on the Internet that a person or software component would find the description of the weather service. The second section of the file defines the messages used as input (the name of the town) and the output (the temperature value) parameters. The third section defines the operation, in this case `getWeather`, encapsulated as a port type. The fourth section declares the encoding rules for the inputs and outputs to the port type, by binding it to the operation. The final section declares the service and binds it to the port type.

WSDL adds to the list of Web Services technologies by providing the means to describe a Web Service with sufficient rigour that a software application can interpret a WSDL file and determine the necessary requirements for integration. The final piece of the Web Services jigsaw concerns the means by which we can discover and locate Web Services that have been made available for use.

3.4 *Universal description, discovery and integration*

If I wish to use a software component and I know the location of the WSDL file, I can simply point my development software to the file and implement the integration. This assumes that I know the location of the WSDL file and I know about the party responsible for the software. If I am creating a business-critical application, I will be concerned about the availability, performance, and other non-functional attributes of the service described by the WSDL file. When developing my application, I may wish to assess the merits of a number of software components supplied by different companies. UDDI provides this extension to the basic Web Services technologies by allowing the means to create a registry of Web Services. This takes Web Services into the realm of companies doing business with each other over the Internet. The UDDI specification enables companies to quickly, easily, and dynamically find and transact with one another. UDDI enables a company to:

- describe its business and its services,
- discover other companies that offer services,
- integrate with these other services.

For example, if I develop a service that relies on a credit checking function in order to validate my customers, I can use a UDDI registry to find that function. My request to the registry would be for a credit checking function and I may also provide other requirements such as cost limits, security needs, performance criteria, etc. The registry would then propose one or more companies that provide such a function, allowing me to choose my preferred supplier.

The specifications for UDDI allow the creation and use of a registry containing information about businesses and the services they offer. The information is organised as follows.

- Business entity

  A business entity represents information about a company. Each business entity contains a unique identifier, the company name, a short description of the company, some basic contact information, a list of categories and identifiers that describe the company, and a URL pointing to more information about the company.

- Business service

  Associated with the business entity is a list of business services offered by the business entity. Each business service entry contains a description of the service, a list of categories that describe the service, and a list of pointers to references and information related to the service.

- Specification pointers

  Associated with each business service entry is a list of binding templates that point to specifications and other technical information about the service. For example, a binding template might point to a URL that supplies information on how to invoke the service. It is also possible to use these pointers to access the service-level agreements that describe the contractual nature of the usage of the service. The specification pointers also associate the service with a service type.

- Service types

  A service type is defined by a tModel. Multiple companies can offer the same type of service, as defined by the tModel. A tModel specifies information such as the tModel name, the name of the organisation that published the tModel, a list of categories that describe the service type, and pointers to technical specifications for the service type such as interface definitions, message formats, message protocols, and security protocols.

It is possible to use UDDI to create private registries that reside within private networks, offering functionality to a specific set of users [9]. The specifications for UDDI are administered by the Organisation for the Advancement of Structured Information Services (OASIS) [10]. Although the first versions of these specifications have been available since 1999, the use of registries for locating Web Services remains relatively immature. In addition to UDDI registries, there are listings of publicly available Web Services [11, 12].

3.5 Bringing Web Services technologies together

The technologies described above provide the means for software to describe itself, be discovered by other software and be invoked over a network. The information technology (IT) industry has embraced the specifications for these technologies to such an extent that Web Services usage is now widespread. The coming together of an industry with a common set of goals such as this is analogous to the development of the railways. In the early days of the railways, the trains ran on different gauge tracks in different parts of the country. This was fine if you wanted to travel from Stockton to Darlington or from Liverpool to Manchester. If you wanted to travel longer distances it meant changing trains and/or using other modes of transport. Bad enough for people, much worse for freight. Because of this lack of 'interoperability', users and businesses preferred to continue to use 'old technology' (i.e. the canals). Once gauges were standardised, interconnect became easier, users were more keen to use the trains, and investors became more keen to finance the growth of the industry. This led to a phenomenal growth in railway investment and travel in the early part of the 20th century and laid the foundation for today's network (imagine NetworkRail's problems today if they also had to deal with different gauges!).

With services on the Internet, until recently we have had islands of growth (e.g. banking services, travel services, consumer goods, etc), but little common technology between them, and no easy way for entrepreneurs to easily stitch services together. Consumers preferred old technology (bricks and mortar businesses). Web Services technologies provides the standardisation that will enable services offered on the Internet to interoperate seamlessly. The more businesses use the technologies in their service offerings, the more their services will be used, thus spurning more investment and more innovation in the types of services offered. Further information about the way in which Web Services technologies work and the architectures that support them can be found in Mockford [8].

4. Integration models

The standard technologies described above allow software components to be combined to create services. An important concept of Web Services is that it supports a number of different integration models, some of which will radically alter the way in which software-based services will be developed, deployed and used.

Consider the example of a company offering a holiday booking service over the World-Wide Web. Users will access the company's Web site, provide their holiday requirements, and will be offered a number of holidays from which to choose. Elements which might be provided include flights, car hire and hotel bookings. The company is acting as a service provider, integrating the elements of the holiday from the suppliers of the flights, cars, and hotels. These elements, and the holiday service itself, can be provided as Web Services by using the technologies described earlier. Figure 6 illustrates two ways in which this can be achieved. The rectangles represent different Web Services.

In example (a) there is a fixed relationship between the holiday service and the three holiday elements. The link between the elements and the holiday service are determined when the software is designed and is known as fixed binding. In example (b) there is a dynamic arrangement between the holiday service and the hotel booking elements. The dynamic nature of this arrangement is designed into the software. When holiday service users provide their requirements, the

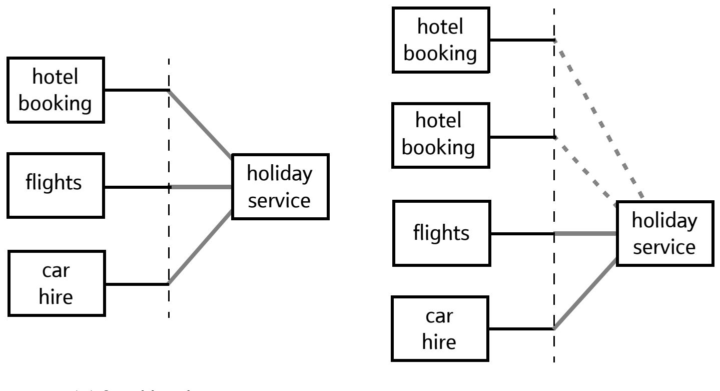

Fig 6 Two ways of achieving integration.

service can choose which of two different hotel booking elements to use (based on criteria such as price, performance, etc). The choice of the preferred element is only made at run time and is known as dynamic binding. Connections between these components will be dynamic to such an extent that they may only exist for the instant that the transaction is conducted. Web Services allow this because the standard technologies used do not require a hard-coded interface between the holiday service and the hotel booking elements.

Web Services extend this concept even further. UDDI provides the means whereby a holiday service can discover (for the first time) businesses offering hotel booking and then bind to the one offering the best service. Over time, there may be many companies offering hotel booking services — all competing with each other to offer services to holiday companies. In addition, there may be many holiday companies competing with each other to secure the services of the best hotel booking services. This may create a market for Web Services brokers, who act as intermediaries between customers and suppliers of Web Services. These concepts create new models for the ways in which businesses will trade. The vision for Web Services is that there will be millions of Web Services competing and collaborating with each other over the Internet.

5. Challenges to implementing Web Services

Whenever two computer applications communicate over a network, a number of basic conditions must be satisfied:

- the connection between them must meet set performance criteria to allow meaningful interaction — these may include security, reliability, bandwidth, etc,
- if the interaction is a business transaction, the parties must agree the offered service, the service-level guarantee, the price and means of settlement, etc,
- if the interaction is complex (e.g. part of a business process like provisioning), it may need to have knowledge of the state of other 'transactions' and it may need to act as a repository for persistent data needed by other elements,
- if the interaction involves either asynchronous behaviour or the possibility of unsolicited messages, there must be a mechanism for capturing these events and acting upon them.

Such challenges have been around before the advent of Web Services, and there are solutions available for use with other technologies. Web Services will evolve based around the way in which such solutions can be applied to the Web Services world, and in the way in which these evolve and are implemented. Figure 7 illustrates the timeline.

The first application of Web Services is seen within company firewalls where performance and security are more easily managed. Here, Web Services are used to complement existing methods for linking company applications. Since the maturity of the basic Web Services technologies in 2002, businesses have been using Web Services to cut the cost of internal integration, for example in operational support systems implementation [9].

The next evolution of Web Services supports electronic business transactions, where trading partner agreements exist and where 'extranets' are used to guarantee agreed levels of performance and security. During 2003 we have seen more and more businesses (including BT) begin to use Web Services in this way [1, 9, 13—15] — opening their trading gateways to a wider variety of trading partner, without the need for costly proprietary interfaces.

Ultimately, Web Services may get close to the vision of large numbers of Web Services that compete and collaborate over the Internet. Businesses will trade with their partners and with consumers, based on short-lived commercial arrangements. They will use Web Services as a key channel to market and will be able to quickly bring new services to market. We expect this evolution of Web Services to mature beyond 2004.

6. The role of Web Services standards

The technologies described in section 3 owe their creation and existence to the development community in companies across the IT industry. As we saw in section 2, Microsoft and IBM were pioneers in obtaining the basic agreements to work on the same set of specifications. Ultimately, however, the success of Web Services has been due to the efforts of the international standards bodies in turning the specifications into formal standards. Three of these stand out.

6.1 World-Wide Web Consortium

The World-Wide Web Consortium (W3C) was founded in October 1994 to lead the Web to its full potential by developing common protocols that promote its evolution and ensure its interoperability. W3C has around 500 member organisations (including BT) from all over the world and has earned international recognition for its contributions to the growth of the Web. W3C is responsible for administering the specifications of many of the basic Web Services technologies, including that for XML, WSDL and SOAP.

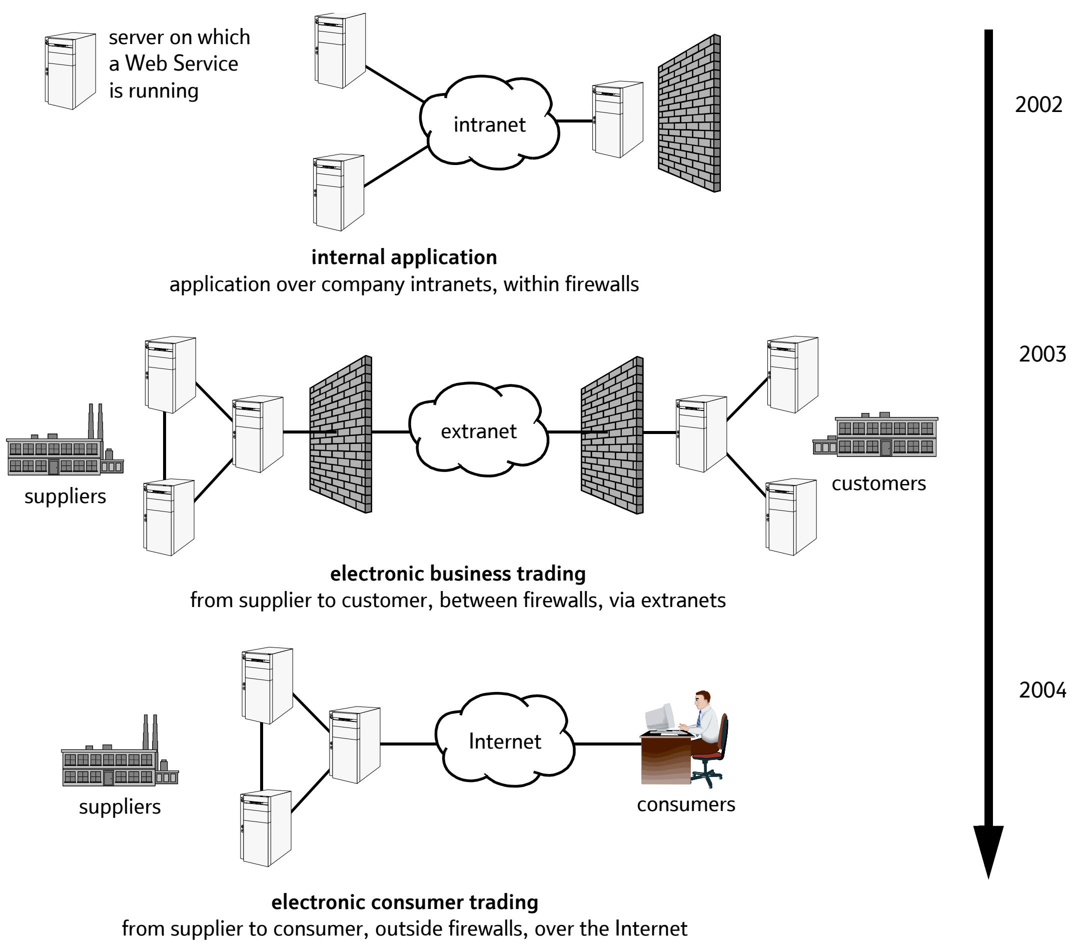
server on which
a Web Service
is running
intranet
internal application
application over company intranets, within firewalls
suppliers
extranet
customers
electronic business trading
from supplier to customer, between firewalls, via extranets
suppliers
Internet
consumers
electronic consumer trading
from supplier to consumer, outside firewalls, over the Internet
2002
2003
2004

Fig 7 Web Services implementation timeline.

6.2 *Organisation for the Advancement of Structured Information Standards*

OASIS is a not-for-profit, global consortium that drives the development, convergence and adoption of eBusiness standards. It has more than 400 corporate and individual members in 100 countries around the world. OASIS and the United Nations jointly sponsor ebXML, a global framework for the use of XML in eBusiness data exchange. OASIS operates XML.org, a community clearing house for XML application schemas, vocabularies and related documents.

Despite the efforts of the W3C in specifying and promoting many of the Web Services technologies, more recently OASIS is the organisation to which companies are turning to specify and promote the additions to the basics, such as Web Services security [16].

6.3 *Web Services Interoperability Organisation*

WS-I is an open industry organisation chartered to promote Web Services interoperability across platforms, operating systems, and programming languages. The organisation aims to provide 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. Unlike W3C and OASIS, WS-I does not actively specify standards — its emphasis is on providing:

- profiles — sets of Web Services specifications that work together to support specific types of solutions,
- sample implementations,
- implementation guidelines — recommendations for use of specifications in ways that have been proven to be most interoperable,
- sniffers — tools to monitor and log interactions with a Web Service,
- analysers — tools that process sniffer logs and verify that a Web Service implementation is free from errors.

WS-I has stated its commitment to building strong relationships and adopting specifications developed by a wide array of organisations such OASIS and W3C.

6.4 The future for Web Services standards

To achieve the vision of competing and collaborating services over the Internet, the technologies that support Web Services must evolve. The basic set — XML, SOAP, WSDL and UDDI — must continue to support the basic objectives: platform agnostic, language agnostic and availability under non-restrictive IPR terms. In addition, specifications to allow Web Services to be used in more complex scenarios need to be developed. One area where additional specifications are needed is in the area of orchestration (sometimes known as choreography). Orchestration describes the way in which separate Web Services can be brought together in a consistent manner to provide a higher value service. Orchestration includes the management of the transactions between the individual services, including any necessary error handling, as well as describing the overall process.

Orchestration is especially important to telecommunications operators (telcos). Across the world, telcos are facing declining revenues from traditional voice and data services and are looking to newer content-based, application-based, multimedia services as a way to boost revenues. Unless telcos are prepared to invest heavily in the content and applications themselves, they will need to partner with content providers and application providers in order to deliver services to users. These partnerships can lead to extended value chains, with a number of businesses taking part. Many of the interactions between these parties can be provided using Web Services which immediately means there is a requirement for Web Services orchestration.

There are currently a number of competing specifications for orchestration, from companies closely linked with Web Services and from companies and standards bodies linked with the more general area of business process management. Many of the current efforts to specify an orchestration language are competing with one another, with the risk that Web Services returns to the days of the 1980s and 1990s of non-interoperable, proprietary standards. Further ideas on the future of Web Services are developed in Davies et al [3].

7. Conclusions

The Internet and World-Wide Web have pioneered the way for integrated information and communications technologies (ICT). Building on the success of one of these technologies — XML — the ICT world has created a set of standard specifications that allow services to be deployed, integrated and used across the Internet. These standards are free to implement and are platform and language independent.

By utilising standards for the location, description and invocation of services, the idea of Web Services has been created and is currently being adopted across the industry. The vision for Web Services is a set of competing and collaborating services over the Internet, revolutionising the way that companies do business and the way that users make use of the services.

The challenge for Web Services is to continue the momentum gained from the adoption of the basic standards and to carry the same principles into their evolution.

References

1 Hill J R: ‘A management platform for commercial Web Services’, BT Technol J, 22, No 1, pp 52—62 (January 2004).

2 Boden T: ‘The grid enterprise — structuring the agile business of the future’, BT Technol J, 22, No 1, pp 107—117 (January 2004).

3 Davies N J, Fensel D and Richardson M: ‘The future of Web Services’, BT Technol J, 22, No 1, pp 118—130 (January 2004).

4 Lofthouse H, Yates M J and Stretch R: ‘Parlay X Web Services’, BT Technol J, 22, No 1, pp 81—86 (January 2004).

5 Darling J and Tye R: ‘Web Services in communication services’, BT Technol J, 22, No 1, pp 72—80 (January 2004).

6 SOAP — http://www.w3.org/TR/SOAP/

7 SOAP 1.2 — http://www.w3.org/TR/soap12-part0/

8 Mockford K: ‘Web Services architecture’, BT Technol J, 22, No 1, pp 19—26 (January 2004).

9 Calladine J: ‘Giving legs to the legacy — integration within the enterprise’, BT Technol J, 22, No 1, pp 87—98 (January 2004).

10 UDDI — www.uddi.org/

11 XMethods — http://www.xmethods.net/

12 SalCentral — http://www.salcentral.com/

13 Gahan C J: ‘URU — on-line identity verification’, BT Technol J, 22, No 1, pp 43—51 (January 2004).

14 Millar W, Collingridge R J and Ward D A: ‘Consumer vehicle telematics — an emerging market where Web Services offer benefits’, BT Technol J, 22, No 1, pp 99—106 (January 2004).

15 Stevens T and May A: ‘Improving customer experience using Web Services’, BT Technol J, 22, No 1, pp 63—71 (January 2004).

16 Kearney P, Chapman J, Edwards N, Gifford M and He L: ‘An overview of Web Services security’, BT Technol J, 22, No 1, pp 27—42 (January 2004).

Paul Muschamp joined BT in 1985 to work on the design of new integrated circuit architectures. Following a move into software development, he led a number of designs within BT’s operational support systems area, including on the PSTN Switch Manager. He spent two years in the USA as a lead designer for Concert, working on customer management and capacity planning systems. Since returning to the UK, he has held a number of posts in the BT group, including Chief Information Architect (during which he led the development of BT’s first cross-business information model), and Chief Applications Architect (during which he led the development of the 21C applications and content architectures). He has recently joined BT Exact as Head of Technology Strategy. He is a member of the Institution of Electrical Engineers and is a Chartered Engineer.