White Paper

Preparing Your Enterprise For Web Services

IONA Technologies PLC

December 2001

Summary

This paper discusses eight of the most important topics needed to prepare your enterprise for Web services. Understanding these topics will lead to successful Web services implementations, and may also provide a competitive advantage to you and your enterprise.

The intended audience for this paper includes IONA technical professionals, such as Systems Engineers and Solutions Architects who need to describe the Web services vision to prospective customers, as well as prospective customers who want to better understand and prepare for Web services.

This paper covers the following topics:

1. Understanding how Web services can be used for business advantage
2. Knowing which standards can safely be used
3. Experimenting with and mastering the basic Web service technologies
4. Choosing a platform vendor that meets your long-term needs
5. Defining a services-oriented architecture and adopting a component-based development approach
6. Developing your own Web services and integrating the Web services of others
7. Defining your Web services publishing strategy
8. Anticipating and planning for the business issues

This paper concludes with a brief summary of these topics.

iii

Table of Contents

v

1 Understanding Web Services

1.1 What Is A Web Service?

What is a Web service? Simply put, a Web service is a software construct that exposes business functionality over the Internet. In the context of a Web service, "expose" means:

- Identifying valuable business processes within the enterprise.
- Defining loosely-coupled, service-oriented interfaces to those processes.
- Describing those interfaces in a Web-based, industry-standard format.

1.2 Major Types of Web Services

There are two types of Web services, typically referred to as remote procedure calls (RPC) and document-oriented messaging.

1. RPC is like traditional synchronous middleware such as EJB, CORBA, and DCOM.
2. Documented-oriented is akin to asynchronous protocols such as MQSeries, JMS, and so on, in that it typically kicks off a process flow but doesn't wait for the entire process flow to complete before returning to the client.

1.2.1 RPC-Oriented Web Services

A traditional RPC-oriented Web service is where a request for a specific function is sent from the client to the server and the client waits for the reply. The server executes the request, and sends the reply to the client who is waiting for the response, as shown in Figure 1.

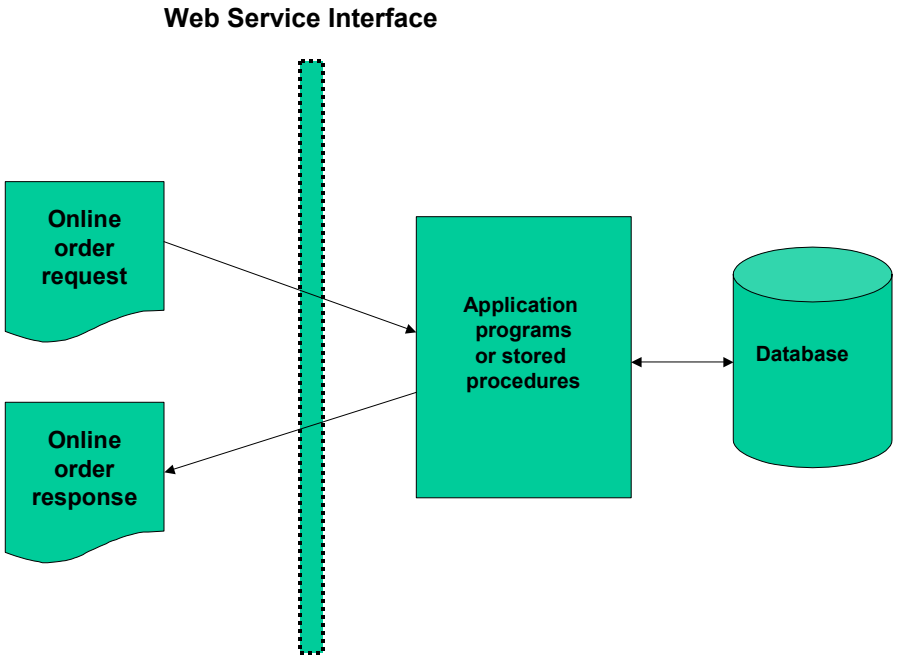

Figure 1: RPC-Oriented Web Service

Some RPC-oriented Web services are already accessible on the web. Examples of stock quoters, exchange rate calculators, and so on, can be found at www.xmethods.net, but most of them are too fine-grained to be realistic and will not scale well on the Web.

Still, there exist other Web services that are of the right granularity and that are ready for integration into your applications. These include services for credit card validation, user authentication, and so on.

Although RPC-oriented Web services are important to applications, we all recognize that enterprise applications also require asynchronous Web services to implement business processes and B2B collaborations. These are described in the next section.

1.2.2 Document-Oriented Web Services

Typical document-oriented, asynchronous Web services allow a client to send a document to a server without the client having to wait for the service to complete. The server receives the document and initiates a business process that is made up of several simpler services.

The scenario illustrated in Figure 2 shows a typical e-business process for order fulfillment, where the company receives a purchase order, sends an email acknowledgement to the customer, optionally uses a Web services to check credit worthiness, ships the product to the customer, and sends an invoice.

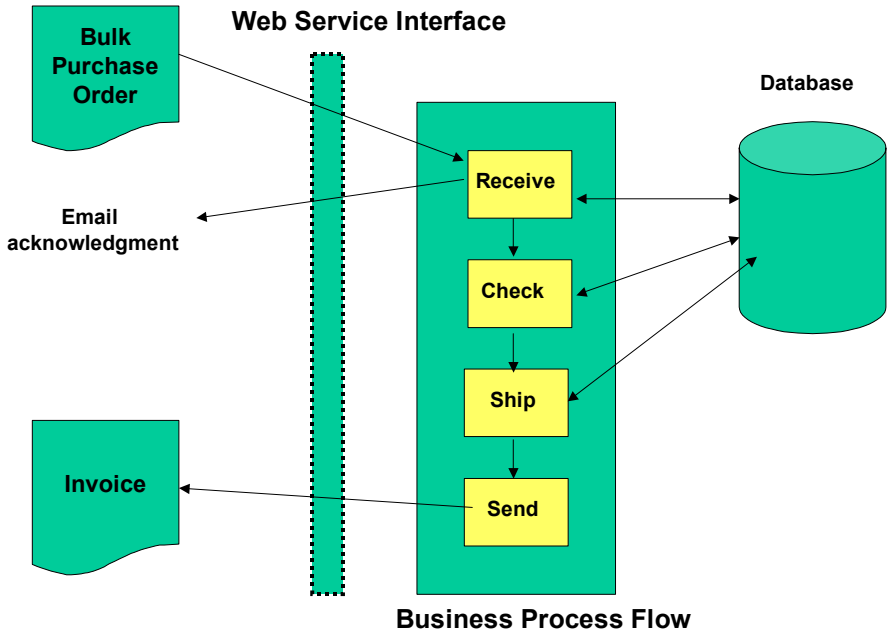

Figure 2: Document-Oriented Web Service

This example shows the benefits of composite document-oriented Web services that are made up of several simple RPC-oriented web services. This composite Web service fully automates a business process to improve the operational efficiency of the business.

Consider what might happen to a company that doesn't use a document-oriented Web service to automate the entire order fulfillment workflow. Instead, this company uses an RPC-oriented Web service that receives the purchase order, notifies an employee of the order, and sends an email acknowledgement to the customer. What is the likelihood that the order sits on the employee's desk for an extended period of time without being fulfilled? Obviously, in that case there would be some dissatisfied customers.

1.2.3 Why Not Traditional Middleware?

We have seen that Web services can support RPC and asynchronous messaging over the Web, but you might ask, "Why bother with web services when I have my favorite procedure-oriented middleware (for example, DCE, CORBA, DCOM, EJB, and so on)? Isn't that sufficient to solve all of my integration needs via remote procedure calls?" As well, others might say, "I have my favorite messaging oriented middleware (for example, MQ Series, JMS, and so on). Shouldn't that suffice for all of my document-oriented integration needs?"

To answer these questions, we must understand that there are a several problems with using traditional middleware on the Web:

1. Middleware provides a great implementation vehicle for services, but none of them is a clear winner on the Web. The fact that no single type of middleware is ubiquitous on the Web limits their use to the integration of services inside the enterprise, and precludes their use for the integration of services over the Web.

2. Middleware requires compatible architectures from all participants in order to succeed, which makes it difficult for an enterprise to use traditional middleware to integrate applications between itself and trading partners over the Web

3. Middleware requires object model-specific protocols such as DCOM, JRMP, IIOP, and so on. Enterprise system administrators are not willing to allow these protocols and transports to pass through corporate firewalls.

4. All of the major middleware vendors are heavily investing in and supporting Web services by parsing and serializing XML. The middleware vendors have been content to do this since it allows them to scale their approach to a wider network.

1.2.4 Web Service Benefits

There are several reasons for using Web services to expose your Enterprise Services.

1. Web services are accessed via ubiquitous web protocols (HTTP) and data formats (XML)

2. Web services support a single technology for business integration inside and outside the firewall. Web services allow companies to integrate in-house applications, publish existing application assets to other companies, and leverage the application assets of others. It is clear to most that business processes and applications will be integrated within and across enterprises using Web services.

3. Web services support multiple computing platforms, which makes integrating applications over the Web easier. Web services allow enterprises to do business with trading partners, customers, and suppliers regardless of the computing platforms and programming languages that are involved.

4. Web services allow businesses to do business more efficiently by extending existing systems to those of trading partners and customers without having to replace existing back-end infrastructure such as ERP systems, packaged applications, and so on.

5. Web services can be used for new business models that call for giving, renting, or selling services.

6. Web services complement traditional middleware platforms. Web service technologies can be used to integrate applications over the Web, while the traditional middleware technology can be used to efficiently implement and integrate applications inside the corporate firewall. And of course Web services

4

can be used to expose the results of middleware-based integration inside the firewall.

2 Knowing The Standards

After achieving an understanding of Web services, the next step to prepare your enterprise is to stay abreast of the Web service-related standards, of which there are many. This section explains which of these standards are likely to be useful to you and your enterprise.

2.1 Starting To Use The Basic Standards

The basic Web services platform is illustrated in Figure 3. This platform is made up of four basic standards: XML, SOAP, WSDL, and UDDI.

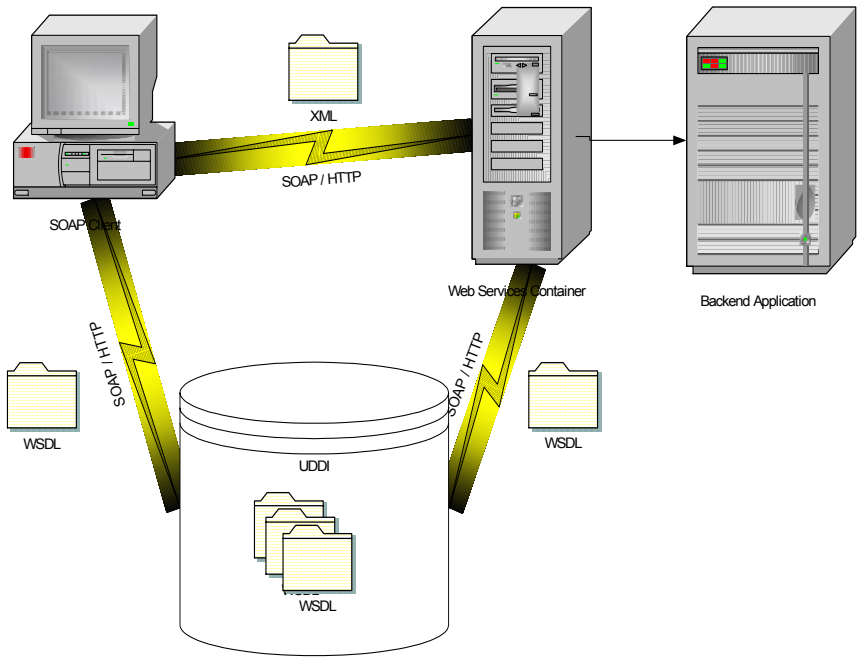

Figure 3: The Basic Web Service Standards

As Figure 3 shows, eXtensible Markup Language (XML) is used for moving structured data across the Web as documents. XML provides a common language for data definition, allows Web services to process data in a variety of useful ways, and greatly reduces the need for custom parsers, translators, and so on.

Simple Object Access Protocol (SOAP) is a protocol specification that defines a uniform way for invoking RPCs using HTTP/XML, and for sharing documents as multi-part MIME attachments. SOAP combines with HTTP/XML to provide three benefits:

5

- HTTP messages are able to get past firewalls that pose problems for other transports.

- XML data formats are easily extendable and well supported with commercial parsers, and so on.

- SOAP is a widely interoperable protocol on the Internet; second only to TCP/IP.

Web Services Description Language (WSDL) defines how a SOAP client can wrap a method call in XML, send it over HTTP to the server, and then process the response. On the server-side, WSDL defines enough information to allow the Web service container to parse the XML request, make the procedure call on the backend application, and wrap the result in XML to send back to the client.

Universal Discovery, Description, and Integration (UDDI) is a specification for a repository of Web services. Web services are described as WSDL entries in the UDDI repository. UDDI provides a mechanism for service providers to register WSDL description for the services they want to publish, and for Web service clients to find WSDL descriptions for the services they want to invoke.

2.2 Staying Abreast Of The Emerging Standards

Although the basic Web services standards provide the RPC and messaging protocols, they don't address some of the traditional platform services that you would expect to have, such as security, transactions, and workflow. Still, there are several emerging standards that are being proposed to address these additional platform services. We list six of the more prevalent ones in this section, but it is important to note that there are many others.

1. Security Assertion Markup Language (SAML): provides single sign-on for Web services. The OASIS XML-Security Services Technical Committee is developing SAML. The committee defines SAML as a framework for exchanging authentication and authorization information. More information about SAML is available at xml.coverpages.org/saml.html.

2. XML Key Management Specification (XKMS): provides authentication and digital certificates. This is a joint submission by Microsoft, VeriSign, and webMethods to the W3C. It consists of two major components: (1) XML Key Information Service, and (2) the XML Key Registration Service. More information about XKMS can be found at www.w3.org/TR/xkms.

3. Transaction Authority Markup Language (XAML): provides transactions support for potentially any number of Web Services interacting over the Web. IBM, Bowstreet, Oracle, HP, and Sun are jointly developing XAML. The XAML group has stated that it intends to submit the proposal to a standards body. More information about XAML can be found at www.xaml.org.

4. Business Transaction Protocol (BTP): BEA proposal for transactions over the Web. Submitted by BEA to the OASIS Business Transactions Technical Committee. This specification covers the problem of long-lived transactions

6

over multiple enterprises. More information about BTP can be found at xml.coverpages.org/ni2001-03-08-b.html.

5. XLANG (pronounced "slang"): a notation for the specification of message exchange behavior among participating Web services; this Microsoft proposal for describing a business process has been submitted to OASIS. It uses a derivative of Backus Naur From (BNF) to build a "Service Description." The service descriptions are based on WSDL. Like the BTP proposal, it aggregates Web Services. But unlike BTP, XLANG has a close relationship with WSDL. More information on this proposal is available at www.oasis-open.org/cover/xlang.html.

6. Web Services Flow Language (WSFL): an IBM proposal for describing a business process. This has also been submitted to OASIS. There is serious discussion about combining the best of XLANG with WSFL. The Gartner Group expects the two proposals to harmonize and be submitted to W3C by the end of 2001. WSFL describes business processes in terms of composition and choreography. See xml.coverpages.org/wsfl.html for more information.

As you can see, the emerging standards are a jumble of acronyms identifying a number of proposals for standards that are in some cases competing. For example, Microsoft's XLANG and IBM's WSFL have both defined XML dialects for coordinating complex interactions between multiple Web services.

2.3 Defining Your Web Services Standards Strategy

With all of these standards, what's a company to do? The short answer is that enterprises should start using the basic standards, should resist the temptation to wait for all of the emerging standards to be in place in the hope of having the perfect implementation, and should stay abreast of the vertical standards in their respective industries:

1. The basic Web service standards provide all the capabilities for standardized access and invocation in heterogeneous environments. You can prepare your enterprise for Web services by starting to use these basic standards:

    - XML, SOAP, and WSDL are all safe options to be using right now. XML is supported practically everywhere on the Internet, and all major vendors are heavily investing in SOAP and WSDL.

    - UDDI is still evolving and IONA's implementation will address some of the questions involved in its use. First, it is supposed to allow programs to search for services like people search for items using a browser. But can a program make decisions like a person with a browser? Second, there are trust issues to resolve. For instance, UDDI can dynamically find a service, but it doesn't tell you whether the people publishing (or consuming) that service can be trusted.

2. There are numerous emerging standards for transactions, security, and workflow. Many of them are starting to undergo a streamlining phase. That being the case, you should track the new and emerging standards, but wait until they stabilize enough for quality implementations before using them. If

7

your enterprise requires these services immediately, there are workarounds. For instance,

- Security can be implemented by passing the username and password as parameters in the SOAP request and using HTTPS as the transport. This approach uses HTTPS for encryption and LDAP or some other mechanism for authentication.

- Transactional semantics can be implemented by an invoked Web service using the capabilities of the traditional middleware. This approach supports a single transactional Web service whose implementation is entirely inside the firewall.

- Workflow can be implemented by composition of service implementations. That is, you can use traditional middleware to implement a Web service that invokes business functions in a workflow.

3. At present, there are a limited number of vertical industry standards. Still, now is the time for IT executives to watch for, and start driving these standards into their respective industries. Some industry initiatives are underway and we expect to see vertical standards emerge in the insurance, manufacturing, and health care industries in the near future.

3 Mastering The Technologies

Prepare your enterprise for Web services by developing the right technology skills for your IT team:

- Start mastering the new Web services technologies to understand how they can be used for business advantage. Internal advanced technology groups should review the extensive information and downloads that are offered at the major vendors' web sites (www.xmlbus.com) and then begin experimenting with the technologies.

- Most enterprises should begin experimenting with the technology to make sure that their IT teams get the right technical skills to utilize SOAP. At a minimum, they must know how to use SOAP within Visual Basic, Java, C++, and so on.

- To publish Web services, your IT group should know how to: implement a Web services listener, understand SOAP messages, generate SOAP responses, provide a WSDL contract for your service, and advertise your service via UDDI.

- To consume Web services your IT team should know how to: use UDDI to locate services, interpret a WSDL contract for the service, generate appropriate SOAP requests, and interpret appropriate SOAP replies.

- It is recommended that you give give developers at least a foundation of Web services training; some education will result in a quicker start. Courses can be found at the major vendors' web sites (www.iona.com).

4 Choosing A Platform

The platforms are rapidly maturing, as Web services become the standard for making remote procedure calls, and for manipulating documents over the Web. In selecting a platform, there are number of items to consider. We list the issues of most interest to developers and also note some of the deployment and administration issues to consider:

1. APIs for constructing and parsing XML messages will enhance productivity by eliminating the need for you to write custom parsing and formatting logic. At a minimum, APIs must support reading and writing XML streams from your programming language of choice.

2. Most major platforms have a software developer kit (SDK) that wraps SOAP functionality inside a class library. Although these SDKs are not required to use SOAP, they can make implementing Web services easier.

3. Toolkits should provide you with automated wizards for generating WSDL interfaces, Web service containers, and client proxies from existing enterprise applications implemented in J2EE, CORBA, DCOM, .NET, and so on. These wizards can also reduce the need for you to learn all the details of the specifications.

4. Platforms should provide you with adapters (for example, SAP, Baan, PeopleSoft, and so on) for simplifying legacy integration and optionally may even provide wizards to generate WSDL interfaces and other code from legacy APIs.

5. Platforms should provide you with SSL integration for sending SOAP requests over HTTPS.

6. Platforms should provide a means to manage, deploy, monitor, and dynamically test Web service end-points.

7. Platforms should provide a means to peek into SOAP messages. Further, the provided tool should be able to create and send SOAP messages to destinations. This “Spy” functionality could also extend to providing profile and performance information.

8. Platforms should offer a way to deploy, re-deploy, un-deploy, activate, and deactivate Web services.

9. Platforms should support browsing a UDDI repository and other discovery/location services.

5 Adopting A Service-Oriented Architecture

Although a service-oriented architecture is not strictly required, enterprises that adopt one, as illustrated in Figure 4, will find that it easy to incorporate Web services into their applications, while those enterprises that write monolithic applications in C or COBOL will find it difficult to incorporate Web services.

9

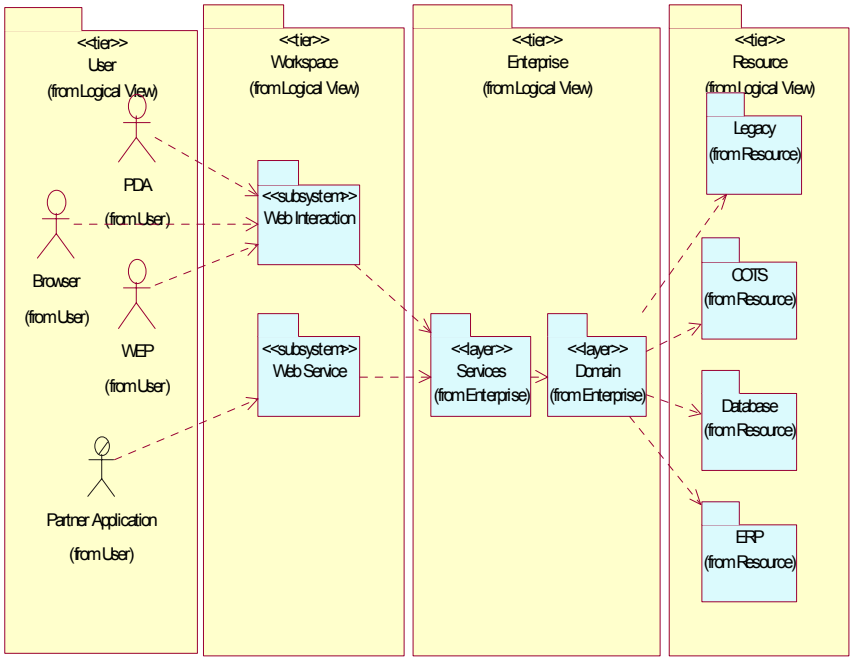

Figure 4: The Service-Oriented Architecture

The service-oriented architecture is divided into four logical tiers:

User Tier: The User Tier consists of the various external devices (or the different channels) that are used by customers, trading partners, and so on, to communicate with the enterprise. The User Tier may include presentation devices such as a Web Enabled Phone (WEP), a PDA, presentation software such as a Web Browser, and partner applications that are using services programmatically.

Workspace Tier: The Workspace Tier mediates between the devices and programs that reside in the User Tier and the services that execute in the enterprise Tier. The Workspace tier can be implemented with a number of technologies, such as servlets, Active Server Pages (ASP), Java Server Pages (JSP), Common Gateway Interface (CGI), and so on. The Workspace Tier is made up of two subsystems: one that supports traditional Web applications and one that supports Web services. In either case, the Workspace Tier receives a request from the User Tier, invokes a service in the enterprise Tier, and formats a reply that is appropriate for various devices or programs that reside in the User Tier.

Enterprise Tier: The Enterprise Tier implements the business logic and is further subdivided into two layers: the services layer which provides high level interfaces to the Workspace Tier, and the domain layer that maps the objects in the problem domain into and out of their persistent storage in the Resource Tier. The Enterprise Tier is typically implemented using modern component programming models (for example J2EE, CORBA, DCOM, and so on), but for simple services the Enterprise Tier may consist solely of legacy adapters that access the Resource Tier. Although the Enterprise Tier of the services-oriented architecture may be quite expensive to

10

implement, once it is implemented, the additional cost of deploying a Web service is greatly reduced.

Resource Tier – The Resource Tier is composed of databases, legacy applications, and so on.

6 Publishing and Consuming Web Services

6.1 Publishing Enterprise Applications

With the introduction of a service-oriented architecture, the first question that comes to mind is, What happens to the existing enterprise applications that have been implemented using DCOM, CORBA, J2EE, and so on? Must we discard them and start all over again? The simple answer is no.

Enterprise applications implemented using component-based approaches such as CORBA, EJB, and so on, typically encapsulate their business functionality inside EJB session beans, CORBA objects, adapters, and the like. This functionality can then be reused and exposed as a Web service by implementing the Web service subsystem (see Figure 4 on page 10) that handles the reception of the SOAP message, the translation of the message, and the invocation of the method in the services layer.

Vendors have already recognized that adding a Web service interface to an existing enterprise application will be a common development scenario, and have provided tools to generate the WSDL Web services subsystem that handles incoming requests, as well as the WSDL file that describes the service. Depending on the vendor, these tools can generate implementation-specific code for CORBA interfaces, EJB session beans, Java classes, and JMS messages.

Adopting a service oriented architecture and using modern component-based environments such as J2EE and .Net results in significant productivity gains because there are tools that can generate the Web service subsystem and the WSDL file that is required to implement the Web service from existing components in these environments.

6.2 Publishing Legacy Applications

So far we have defined a service-oriented architecture and seen how to add Web services to enterprise applications. The next question is, how does an enterprise add a Web service interface to a legacy application? This question is critical, because enterprises have already invested a lot of time and money in legacy systems.

Legacy systems can be incorporated into an enterprise application running on J2EE, .NET, and so on, through an adapter. The adapter may take XML as its input, convert the XML into calls on the legacy system, and return the output parameters as XML.

11

Enterprise applications that call the adapters can then be exposed as Web services, using the steps that were outlined for publishing enterprise applications. The services layer of the enterprise tier accesses the legacy adapters, as shown in Figure 4 on page 10.

6.3 Consuming Web Services

In general, Web service clients can invoke Web services to re-use business functionality implemented by others, to collaborate with existing trading partners in private communities, and potentially to dynamically discover new trading partners. Although the service-oriented architecture illustrated in Figure 4 on page 10 shows partner applications consuming the Web services published by the enterprise applications, the Web services can also be used by the enterprise applications themselves.

In such a service-oriented architecture, we're used to thinking about the Enterprise Tier getting its data from database or legacy applications, but what if the Enterprise Tier were to get its data from a Web service instead? Suddenly, we see that a service-oriented architecture can also be constructed from multiple Web services that work together to provide data and services for the application.

The Web services client needs the WSDL file for the service it is to invoke. The Web services client can get the WSDL file from a UDDI repository, from an LDAP directory server, from the local file system, from a compile time constant, and so on. The Web services client uses the WSDL file so it can start communicating with the Web service. The Web services client can then send a SOAP message over HTTP to the Web service and interpret the XML reply.

The SOAP messages used by the Web services client can be created by any of a number of vendor SDKs. In addition, many platform vendors support the creation of web client proxies from WSDL files. Thus Web service client developers can get additional productivity gains by using these generated proxies to hide the details of XML formatting, sending a Web service request, and interpreting the XML response.

But note that an enterprise that implements a Web service need not provide an SDK or any other development tool to consumers; all they need to provide is the WSDL file.

7 Defining your Web Services Publishing Strategy

Enterprises should take into account a number of practical considerations when publishing their Web services. By addressing these practical concerns, Web services will have a simpler design, easier error handling, and improved performance:

1. Consider the frequency of message exchange. Exchanging a few coarse-grained messages is more efficient in a network environment than exchanging

a lot of fine-grained messages. Fine-grained messages result in increased network traffic and make handling errors more difficult.

2. Consider network overhead and bandwidth issues. The fact that creating a SOAP message has some overhead should be kept in mind. There is also some cost to consider in terms of reading and writing SOAP messages.

3. Create higher-level, coarse-grained interfaces. Sending and receiving more information in a single request is better than sending information bit-by-bit. Provide the client with access to a specific business service, rather than getting and setting a specific data value. Enabling business to take place via a single message exchange is the best way to design a Web services interface.

4. Use document-based messaging. There is no reason for the Web service client to have to know the name of the remote methods. In document-based messaging, the Web services subsystem receives the document, invokes the appropriate methods, and responds. Using document-based messaging results in loose coupling of applications.

8 Anticipating the Business Issues

Web services technology issues are quite simple compared to the business issues involved in connecting seamlessly with customers and partners. IT directors will undoubtedly have a greater challenge in addressing the business issues than in acquiring the right IT technology skills. Thus, the business issues must be anticipated and planned for in advance.

Below are few of the business issues to consider. There are many other potential business issues and no one knows the answers to all of them, but armed with a knowledge of Web services, you can anticipate and plan for them in advance. Some of the issues are:

1. Will your systems use dynamic registry lookup? If so, how will you know whether the Web services that you are accessing can be trusted? Conversely, how will you know whether the business partners accessing your Web services can be trusted?

2. How will you enforce service level agreements? For instance, if your primary partner's Web service isn't available when you need it, do your business policies require you to retry after a time? Or will you want to fail-over immediately to an alternate service?

3. Will your enterprise use Web services to improve the operational efficiency of the business that you already do? Or will your enterprise use Web services to create new business models and open up new revenue opportunities by publishing the application assets that you already own?

4. With vertical industry standards for Web services yet to arrive, will you ask that your trading partners adopt these standards? Will you wait for your trading partners to more fully adopt Web services? Or will you simply wait for the vertical industry standards to become generally accepted?

13

5. How will you and your partners test Web services before going into production?

9 Summary

This paper presented eight topics that will help you prepare your enterprise for Web services in a way that may prove to be a business advantage. In summary, the enterprise that wants to use Web services to best advantage will:

1. Understand how Web services can be used for business advantage. Among their uses are ease of in-house application integration, improvement of operational efficiencies, and creation of new revenue opportunities.

2. Stay abreast of the Web service-related standards so that you will know which ones can be safely used and which ones to avoid for the time being. Start using basic standards, and avoid the temptation to wait for perfect implementations of standards.

3. Master the Web service technologies. Advanced technologies groups within the enterprise should review the extensive information and downloads offered at the major vendors' Web sites, and begin training on, and experimenting with, those technologies.

4. Choose a vendor that offers a platform that meets your needs. Most significant platforms have a software developer kit (SDK) that wraps the SOAP functionality in a class library, and that has wizards that generate listeners, build WSDL interfaces, and create proxies.

5. Adopt a service-oriented architecture and modern component-based environments such as J2EE and .NET. For enterprises to succeed at Web services, they need to embrace the concept of services-oriented architectures and component-based development so that they develop granular, flexible services instead of monolithic applications.

6. Start developing your own Web services and consume Web services provided by others. Use Web services to tie existing systems together, publish higher level business compositions, and interact with suppliers and partners for authorizing credit card transactions, conducting credit checks, and so on. In some cases, you may even look for new revenue opportunities by publishing your own application assets for incorporation into other enterprise applications.

7. Start to think through your Web services publishing strategy, for example, decide what to expose, to whom, and so on. Instead of having monolithic applications, start to break these monolithic programs into coarse-grained services. This will make publishing, integrating, and syndicating existing applications much easier.

8. Plan for the discussion of Web services' effect on business issues. Connecting seamlessly with customers and partners raises many issues that will require IT directors to anticipate and to address concerns from business people elsewhere in the enterprise.

14

10 Appendix A: the Orbix E2A E-Business Platform

IONA understands that the success of Web services depends on two major aspects of contemporary application development. The first is integration of existing applications and IT assets. The second is continuing application development in the context of having integrated IT assets that can now be leveraged.

IONA's e-Business platform has been expanded and renamed Orbix E2A™ – which stands for "End to Anywhere™" and denotes the need for enterprises to be able to connect their applications to any channel, format, or technology, now and in the future. Orbix E2A consists of a Web Services Integration Platform and an Application Server Platform.

10.1 Orbix E2A Web Services Integration Platform

The Orbix E2A Web Services Integration Platform is built on IONA's ART™ (Adaptive Runtime Technology) and is composed of:

- Web Service Broker
- Support for vertical XML standards
- Business process orchestration
- Application integration framework and integration broker
- Application Adapters
- Enterprise services

15

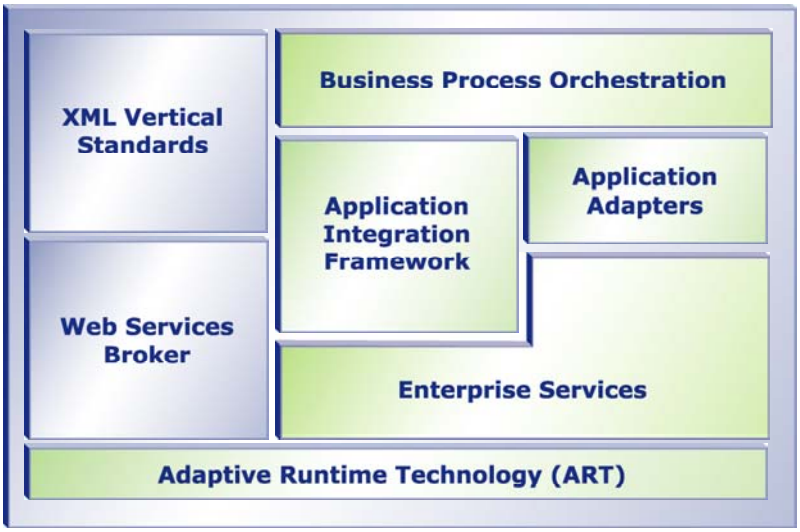

Figure 5: Orbix E2A Web Services Integration Platform

The purpose of the Orbix E2A Web Services Integration Platform is the integration of existing applications without the need to write a lot of code. It contains technology that developers can use to leverage existing ERP data stores with minimal effort.

10.2 Orbix E2A Application Server Platform

The Orbix E2A Application Server Platform built on IONA's ART™, using well-known, standards-based distributed object technologies:

- CORBA
- EJB
- Common Services
- Mainframe Connectivity
- Messaging and Security

16

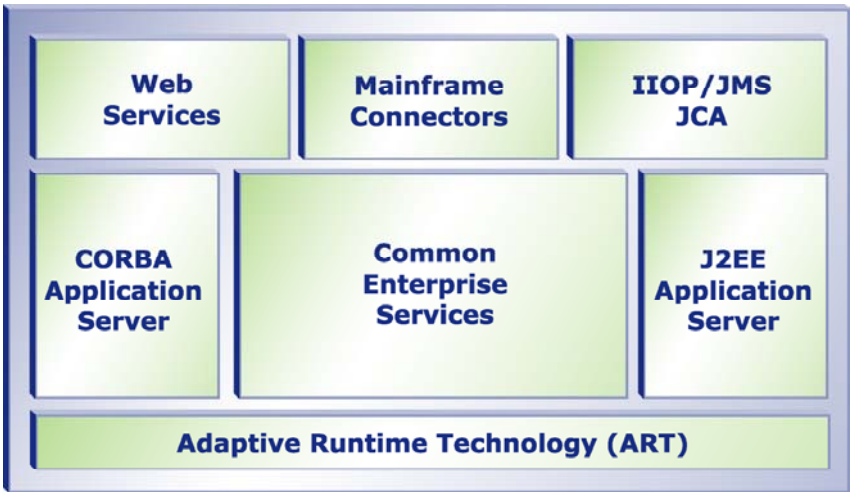

Figure 6: Orbix E2A Application Server Platform

The Orbix E2A Application Server Platform complements the Orbix E2A Web Services Integration Platform by providing the infrastructure to build new enterprise applications. Using the Application Server Platform, developers can build new applications that leverage Web Services provided by the Web Services Integration Platform.

Building on the community standards that IONA has been involved in for ten years, such as SOAP, XML, J2EE, and CORBA, the Orbix E2A e-Business Platform has no dependency on implementation language, operating system, or platform.

The E2A e-Business Platform addresses the business need to reach more customers with less effort. The platform covers both existing applications (for example, ERP applications and data stores) as well as new development projects, satisfying the varying needs of e-business customers.

17

11 Further Reading

1. IONA Technologies. *IONA E2A Application Server Platform White Paper*, December 2001.

2. IONA Technologies. *IONA E2A Web Services Integration Platform Product Brief*, December 2001.

3. IONA Technologies. *Architecting Web Services White Paper,* December 2001.

18

12 Contact Details

IONA Technologies PLC
The IONA Building
Shelbourne Road
Dublin 4
Ireland
Phone: .....+353 1 637 2000
Fax: .....+353 1 637 2888

IONA Technologies Inc.
200 West St
Waltham, MA 02451
USA
Phone: .....+1 781 902 8000
Fax: .....+1 781 902 8001

IONA Technologies Japan Ltd
Akasaka Sanchome Bldg 7/F
3-21-16 Akasaka
Minato-ku, Tokyo
Japan 107-0052
Phone: .....+813 3560 5611
Fax: .....+813 3560 5612

Support: .....support@iona.com
Training: .....training@iona.com
Orbix Sales: .....sales@iona.com
IONA's FTP site .....ftp.iona.com

World Wide Web: www.iona.com

www.xmlbus.com

19