White Paper

Architecting Web Services

By Mike Rosen, Chief Enterprise Architect, IONA Technologies,

and John Parodi, Principal Writer, IONA Technologies

IONA Technologies PLC

December 2001

M2758

Summary

This paper discusses the architectural considerations and requirements for support of Web services and Web service applications. It reviews basic Web service architectures and technologies, examines the granularity of Web services, and describes how higher-level business functions are exposed or presented as Web services.

The paper then introduces an enterprise-scale architecture for building Web service applications and places that architecture into the context of the total enterprise. Finally, the paper shows how this enterprise architecture for Web services addresses the architectural requirements, by virtue of its success in actual customer implementations.This paper is derived from a Webcast presentation by Mike Rosen, Chief Enterprise Architect with IONA Technologies. Mike's broad knowledge of enterprise architecture comes from hands-on implementation experience with B2B, EAI and distributed technologies, including Java, XML, CORBA, COM and messaging. He is author of several books, including ***Designing E-Business Systems and Architectures: A Manager's Guide***. E-mail him at mike.rosen@iona.com.

Table of Contents

Summary iii
1 Introduction 1
2 Definition of Web Services 2
2.1 Web Service Characteristics 4
2.2 Creating Higher-Level Business Functions 5
2.3 Document Based Processing 7
2.4 The Importance Of Business Models 9
2.5 Additional Requirements 10
3 Fundamentals of Distributed System Architecture 10
3.1 Tiers and Responsibilities 14
3.2 The Process Automation Trap 16
3.3 Web Services Enterprise Architecture 18
3.4 Using Web Services for B2B Interaction 20
3.5 Using Web Services for EAI 20
4 Industry Trends and Evolving Standards 22
5 Architecture Review and Conclusions 24
6 Further Reading 26
7 Contact Details 27

1 Introduction

Any good architecture is constructed in response to specific requirements, and in accordance with basic architectural principles. Some of the basic architectural principles discussed in this paper are:

Separate concerns: The purpose of a strict separation of concerns is to keep independent things independent, so that a change in one part of the system does not adversely affect other parts. The most familiar example of this principle is the separation of interface and implementation. At the architecture level, this can be viewed as a separation of presentation functions from business functions.

Another critical separation of concerns, for the purposes of this discussion, is the need to keep the business logic independent from the specific technology (EJB, CORBA, DCOM, and so on.) that is used to implement it. The fundamental idea is that a business solution should not be specified in terms of IT constructs such as messages or objects.

Accommodate the future: Architecture should provide flexibility, so that future application requirements can be satisfied easily. In building a flexible architecture, we try to identify both the future application requirements and areas that are likely to change. We isolate those likely changes with architectural "flex points." Tracking industry trends helps us to identify some of these areas of potential change. Clearly, the constantly changing technology landscape means that the architecture must also be flexible in the face of new technologies as they emerge and mature.

Align With Industry Standards: Industry standards provide enormous value to IT development, not only by providing standard solutions to difficult problems, but also by providing choice and technology commoditization, which let customers avoid vendor lock-in. Architecture must identify and incorporate appropriate standards, both current and emerging.

Architect for the Enterprise: Architecture strives to promote consistency and re-use. Thus, the difference between architecture and enterprise architecture is one of scope. Whereas application architecture typically applies to a single application or family of applications—for example, life insurance policy underwriting and related functions—enterprise architecture is concerned with providing a consistent infrastructure throughout the entire enterprise, to be used by many different types of applications. Enterprise architecture must also promote the development of business functionality in such a way that it is easily reused by those many different applications. So, to continue the underwriting example, even if the original underwriting function was designed for life insurance, a forward-looking enterprise architecture would allow it to be applied to auto and homeowners policies as those functions are added over time.

Business Drivers: Perhaps the most important architectural principle is that the purpose of architecture (and of IT as a whole) is to support the enterprise's business drivers, that is, the business strategies and goals. In consulting with CIO's, CEO's, and IT Directors over the past few years, the following common, primary business goals for IT have emerged:

- Become more competitive in this fast moving e-business environment, meaning much faster time-to-market with new applications.
- Continue to improve the quality of those applications.
- Reduce cost of the applications, meaning not only the development cost but the costs of maintenance and operation as well.

2 Definition of Web Services

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.

Service-oriented interfaces can be organized into Service-Oriented Architectures, which define systems in terms of reusable business services rather than business data, so that business processes can be used in many different applications. Commonly, smaller units of functionality are recombined into several different, larger business processes. Building systems based on such an architecture means that changes in business data do not require changes in cooperating or existing systems.

Loose coupling applies to several aspects of system design, including synchronicity, interface, data, and technology. Systems that are loosely-coupled in time have an asynchronous or event-driven model rather than a synchronous model of interaction. Interfaces can be designed with loose coupling in mind, so that minor changes and enhancements in services do not require updates to client software. The data passed between applications can also be described in a loosely-coupled, extensible way; XML allows backward-compatible and flexible enhancements to data schemas. Loose coupling allows different parts of the system to work together, but to remain independent so that changes in one part do not necessarily require changes elsewhere.

In this same context, "over the Internet" means:

- Publishing a description of the interface so that it can be discovered and used. The description of the interface is expressed in an industry-standard, XML-based format called Web Service Description Language (WSDL). WSDL supports a complete description of the operations available and the parameters required to use those business services. In addition, it describes how to bind to those services, specifying the protocols and endpoints required. Once a Web service is described, its description is published in a repository that conforms to the UDDI (Universal Description and Discovery Interface) standard. Clients can query the repository to discover an appropriate service. A client might access a service that is already known, or might search for a service based on a category, depending on the application. B2B partners will typically look up known services with known partners. Private consumers will be more likely to search for services by category, for example stock quotes or currency conversion, where the semantics of interactions are well understood. Over time, we expect to see a federation of repositories organized around industry verticals, like insurance or health care.

- Accepting requests and returning replies. The request is a message formatted according to the Simple Object Access Protocol (SOAP), which is a modular protocol built on top of XML. On the Internet, SOAP messages are sent using HTTP or HTTP/S (although SOAP is actually protocol-independent). SOAP's XML foundation is important because XML provides an extensible mechanism for describing messages and content. This extension capability allows for a loosely-coupled relationship between sender and receiver, which is especially important over the Internet where two parties may be in different organizations or enterprises. In addition, XML is represented in ASCII text, which can easily pass through corporate firewalls over the HTTP protocol.

- Bridging between the outward-facing interface (accessible via standard Web protocols) and internal implementation of the service (typically using standard as well as proprietary enterprise protocols). The SOAP request is received by a run-time service (a SOAP "listener") that accepts the SOAP message, extracts the XML message body, transforms the XML message into a native protocol, and delegates the request to the actual business process within the enterprise. These run-time capabilities may be hosted within a *Web services container*, which provides scalability, load balancing and other enterprise qualities-of-service for the Web service itself.

So these six items serve as a first approximation of the requirements for a Web services architecture, as illustrated in Figure 1.

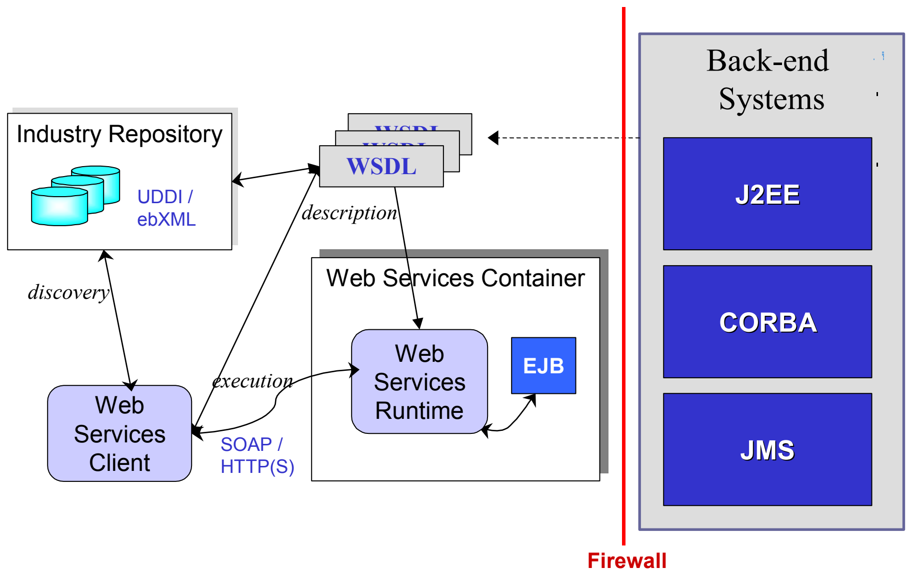

Figure 1: Basic Web Services Architecture

To summarize, Web services technology uses XML and other standards to expose business functionality. The business services are described in WSDL, which is advertised in a UDDI repository. A client discovers the service and invokes it by sending SOAP messages that conform to the WSDL description.

Web services were originally envisioned as providing interaction over the Internet, and that is typically how they are used today. But they are certainly not limited to that environment. Today, we are seeing simple business functions, such as CORBA objects, EJB or Java classes, or the targets of JMS messages, being exposed as Web services. Software, such as IONA's Orbix E2A e-Business Platform, provides tools that examine such objects and automatically express their interface descriptions in WSDL.

A Web services runtime component (such as the one generated by Orbix E2A) handles the reception of the message, the translation of the XML to another format if necessary, and invocation of the back-end business functionality. In some systems, these runtime capabilities will be encapsulated within a Web service container.

2.1 Web Service Characteristics

Looking at Web services from a slightly different angle, we can say that they have the following characteristics:

- A Web service exposes a well-defined service-based interface described in WSDL.
- A Web service is registered and can be located through UDDI or another Web service repository.
- A Web service communicates using higher-level, XML-based messages sent over standard protocols such as SOAP.
- A Web service is implemented inside the enterprise using existing (or new) business functionality.

So far, the basic architecture shown in Figure 1 supports the characteristics that we desire in a Web service.

It is important to note that there are two basic location or discovery paradigms:

- Static, in which you find a service that you know about.
- Dynamic, in which you discover a service dynamically.

Static discovery is much like existing naming or directory mechanisms in use today. Dynamic discovery is similar to Web search engine capabilities but is not prevalent today. This simply means that Web service discovery is one of those areas in which change is likely, and this potential for change has to be addressed by the enterprise architecture.

2.2 Creating Higher-Level Business Functions

Although Web services are similar in many ways to other service-style software constructs, the characteristics of Web service usage are very different and must be taken into account. The Web service designer must deal with the basic physics of widely distributed applications.

One of the most common mistakes made with CORBA and J2EE programming, is to design a distributed object as though it were the same as a local object. This usually results in too much—and mostly unnecessary—network traffic and interaction. This consideration is even more important with Web services.

To illustrate this, lets examine some system characteristics. When we analyze system performance, we measure the duration of local invocations in units of microseconds, or with today's processors, even nanoseconds. When we measure the latency of LAN messages we use units of milliseconds. But when we measure message latency over the Internet, we measure in seconds. Thus a distributed request over the Internet is between a thousands and a million times more expensive than making a local call.

While networking overhead is obviously an issue for Web services, this does not mean that we should avoid using networks. But it does mean that we have to define our network service interfaces intelligently.

The foremost goal of interface design for Web services is to increase request granularity, which is to say that we must design higher-level interfaces than those required in other service paradigms. In other words, a Web service must provide more value and pass more information in a single request.

This goal is achieved by defining interfaces in terms of providing a specific business service, rather than in terms of getting and setting specific data values. Web service interfaces should enable business to take place via a single exchange of messages. One guideline we can use to help us define such interfaces is that they should expose a valuable business function, for example a sales transaction for a specific item.

In terms of implementation, we create such a higher-level business service by combining a set of more primitive or fundamental business functions. We can call these combinations *Business Compositions* and design them to follow the guidelines laid out for service-oriented interfaces, that is, exposing a valuable business function and exchanging data via a single XML message exchange.

One of the major advantages of this approach is that we can create many different compositions from the same set of primitive functions. For example, a credit verification primitive could be used by all services that require immediate payment.

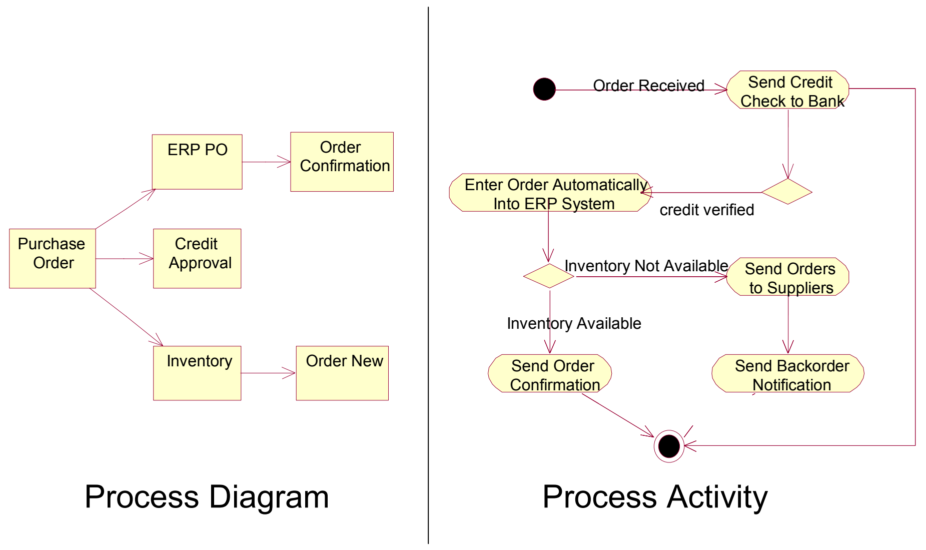

Figure 2: Sample Business Composition

Figure 2 shows a sample business model that describes a business composition for purchase order processing. On the left is a process diagram, where each box represents a fundamental or primitive business process. The composite purchase order process uses the ERP system, the credit approval system, and the inventory system.

This kind of diagram shows a usage relationship, which is critical for understanding dependency management as well as for visualizing the overall picture. To keep the models simple, interfaces and parameters are not shown.

On the right of Figure 2 is a process activity diagram that shows the flow of the purchase order process. Both process and process activity diagrams are useful and necessary for good design.

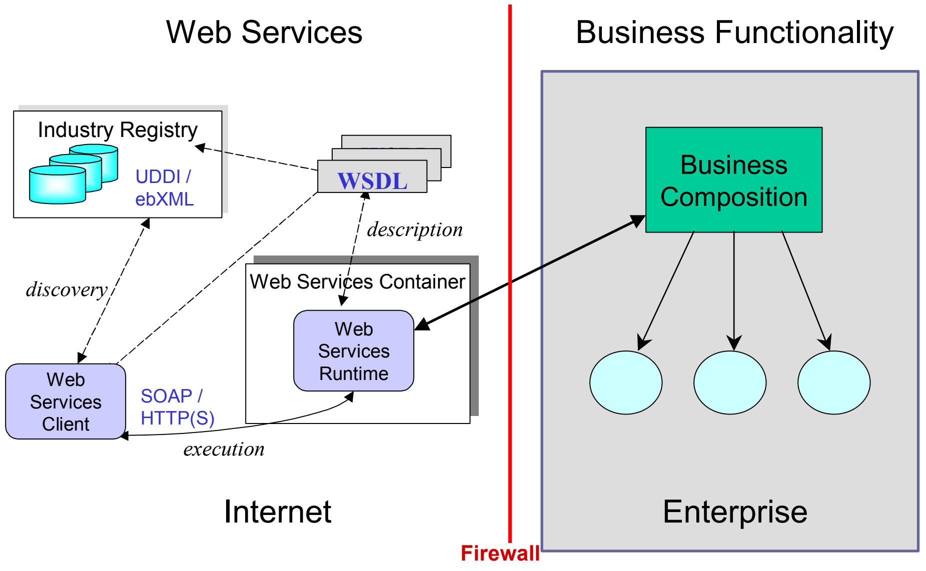

Figure 3: Improved Web Services Architecture

Figure 3 illustrates an improvement over the Web services architecture shown in Figure 1. Instead of exposing low-level classes as Web services, this architecture uses a higher level, higher value business function to expose as a service interface.

Changing the level of interface did not affect the underlying Web service technology architecture, and this is an important point. This means we can use Web services at a variety of levels within an enterprise, as described below.

2.3 Document Based Processing

One of the characteristics of higher-level interfaces is that more data is passed in a single message exchange. Many of these higher-level services will use a document-based communications approach, with data organized into an XML business document, as opposed to an RPC style.

The difference between the RPC style and the document processing style is that the RPC style passes a small number of individual data items in a request, and synchronously gets a small number of reply data items in return. The document processing style passes a collection of data in the form of a business document.

The document contains all of the information required for business processing. It typically contains far more data than the amount passed in RPC-style parameters. Because it is formatted in XML, the document's structure is self-describing, which makes it easy for subsets of information to be extracted for different processing steps. While a document processing request/response could be synchronous, an asynchronous approach is more common.

Since these documents can contain a great deal of data, document processing can be complex. A superb paradigm for both document processing (even for simple documents) and business composition, is that of a business process model (for example, a workflow definition accompanied by an execution service (for example, a workflow engine), as shown in Figure 4.

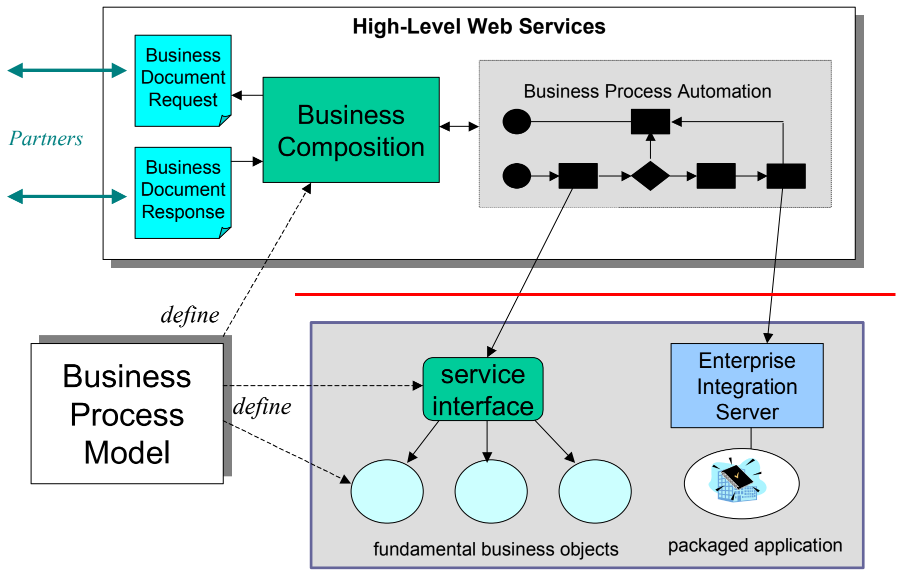
High-Level Web Services
Business Document Request
Business Document Response
Partners
Business Composition
Business Process Automation
define
Business Process Model
define
service interface
Enterprise Integration Server
fundamental business objects
packaged application

Figure 4: Document-Based Web Services

At the top of Figure 4 are the different components of a high-level Web service. Receipt of a request document from a partner is shown at the upper left.

The information in a request document corresponds to the steps in a business composition, and the business composition is defined as a Business Process Model. The process model expresses a sequence of steps, conditionals branches, transformations, and so on. A business process service executes the steps in the model.

Steps within the model can invoke fundamental/primitive business processes, which themselves have service-oriented interfaces. Steps may also invoke processes within packaged applications, such as an ERP system, typically through an enterprise integration server or through specialized adapters.

Information resulting from the request is then formatted into the response business document and sent back to the partner. Of course, this architecture supports other scenarios. The enterprise might initiate a dialogue with a request to a partner and then process the response. The architecture supports both directions of flow.

Note the critical role of the business process model. It is used to define the business compositions and the fundamental business objects. More detailed versions of the business model also define the service-oriented interfaces and the associated process flows.

2.4 The Importance Of Business Models

Most of the business applications developed in the past forty years do not have a business model associated with them. So it is perfectly reasonable to ask why business models are needed.

The fact that most problem domains and applications are not formally modeled is one of the reasons that application evolution is so complex and difficult, and why the IT industry has largely failed to deliver on the promise of software reuse.

Figure 2 showed a very simple example of a business model. In practice, a business model is normally begun at higher level than that shown, and typically includes:

- A Domain Model, which describes standard industry domain entities. For example, the insurance industry has entities such as *party*, *risk*, *beneficiary*, and so on. These insurance domain entities are described in standards published by the ACORD consortium, so individual enterprises don't have to develop a complete domain model themselves.

- An Enterprise Model, which describes the high-level processes and entities that a particular enterprise adds to a domain model. To continue the insurance industry example, an enterprise may be in the re-insurance sector, so only a portion of the insurance domain model would apply. The enterprise of course has its own set of processes for doing business, and these processes represent the company's added value and competitive advantage. The Enterprise Model represents the entire enterprise.

- Internal models, which provide the details on interfaces, process flows, and so on, for individual components and applications.

Thus the business model provides the understanding needed before we can design the fundamental re-usable business objects that span applications in the enterprise. Designers and architects need to understand what the enterprise looks like and what the present and future requirements are. Only then can they correctly identify the abstractions that apply universally. And once those abstractions are identified, they can be mapped to existing systems.

Failure to identify these widely applicable abstractions means failure to correctly identify reusable assets.

But if processes and entities are correctly identified, they can be recombined and reused in many new ways. This makes systems much more flexible and therefore better able to respond to the ever changing competitive environment. It is no overstatement to say that a correct business model, and the creation of a supporting infrastructure of services commonly used by applications, are the keys to dramatically improving the speed of application development.

On the other hand, those who want to conduct IT business as usual will continue to create business processes that aren't flexible enough to span applications and that will result in new, similar-but-different processes for different applications. This path leads to the same problems of redundant components, duplicate processes, and incompatible data that Web services promise to solve.

Of course, Web services alone cannot not solve these problems. But once the foundation of modeling, infrastructure, and architecture is in place, Web services do promise standard technology that reduces the cost and complexity of producing easily accessible business processes that can be re-used, and thereby integrated with other business processes. The goal of identifying and designing the correct processes is now more important than ever, and the business model is how we achieve that goal.

Software architecture models, including business models, are represented in the Unified Modeling Language (UML), in the form of activity, collaboration, class, and component diagrams. UML is based on an OMG standard, and the OMG is promulgating an entire methodology based on formal modeling. See www.omg.org/mda for more information about the OMG's Model Driven Development program.

2.5 Additional Requirements

Many B2B collaborations—and by extension many Web services—will be subject to some kind of Service Level Agreement (SLA). An SLA may range from something as simple as specifying a minimum and maximum response time, to something as complex as having different service guarantees for different pricing levels.

For example, an Application Service Provider might offer two levels of subscription, basic and premium. Those who opt for a premium SLA would pay a higher price and in return get, for example, guaranteed throughput, response time, or up time. Customers who pay for a premium SLA will want to be able to monitor service to ensure that they are actually getting the premium level of service.

Further, collaboration between business partners will require some kinds of shared context to be passed with the business documents. A shared context is necessary for security and for transaction control, and potentially for billing information, e-commerce market membership information, and so on.

Finally, collaborations may take hours or days to execute. Clearly, we cannot use tradition two-phase commit (2PC) transactions to assure the atomicity of these interactions. The issue here is the duration of resource locks and the impact on concurrency. Instead, a new mechanism for extending atomicity will need to be used.

Work is under way in developing solutions and standards for all of these issues.

3 Fundamentals of Distributed System Architecture

Before presenting the enterprise Web services architecture, let us review some of the fundamentals of distributed system architecture. Two of the most important concepts in distributed system architecture are tiers and layers.

Architectural tiers and architectural layers both describe a logical separation of functions such that each tier or layer has a specific set of roles and responsibilities.

The logical separation for architectural tiers—that is, the boundaries between tiers—are chosen/designed to support distribution, scalability, and reuse. Logical tiers can be mapped to any number of different physical computer network topologies. At one end of the distribution spectrum, all tiers might reside on the same machine. In more complex environments, a single logical tier might run on multiple machines (in the form of clusters or "machine farms").

The logical separation for architectural layers is chosen based on the need to separate infrastructure capabilities (for example, communication) from general services (for example, logging), from business logic. The relationship between architectural tiers and layers is shown in Figure 5 on page12, which portrays three architectural layers (in green) and four architectural tiers (in blue).

The lowest layer is the Infrastructure layer, which provides the underlying technical and communications capabilities. Generally, the functionality in this layer is purchased from a middleware vendor; frequently, the purchased middleware will be customized. Although this layer is important, it is not the subject of this paper.

The Services layer contains common utility functions that are useful in multiple tiers and by multiple classes of applications. Services include capabilities such as XML parsing and persistence.

The Application layer is where application and business functionality is implemented and where we actually apply the roles and responsibilities of the four architectural tiers.

The four-tier architectural model evolved from the classic three-tier model. In our experience, the four-tier model is more suitable to an enterprise that must support a wide range of client applications, devices, and access channels. In the four-tier model, the "classical presentation tier" has been divided into user and workspace tiers. Also, some logic that was typically implemented in the enterprise tier under the three-tier model has been moved into the workspace tier:

- The User Tier is responsible for presentation and device independence, for example supporting both a Web browser and a WAP-enabled phone.

- The Workspace Tier is responsible for a maintaining a user session, and for manipulation of user data associated with that session.

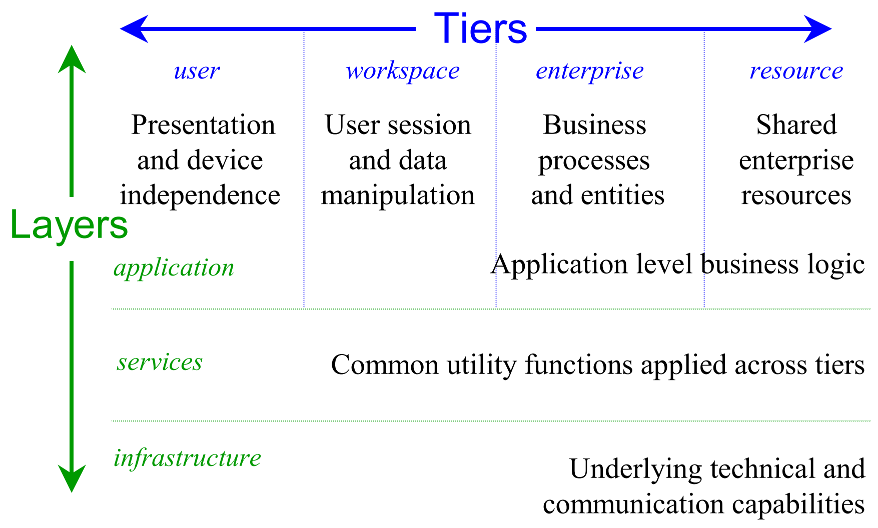
Tiers
user
workspace
enterprise
resource
Presentation and device independence
User session and data manipulation
Business processes and entities
Shared enterprise resources
Layers
application
Application level business logic
services
Common utility functions applied across tiers
infrastructure
Underlying technical and communication capabilities

Figure 5: Architectural Foundations

Together, the User and Workspace Tiers support all of the interaction with a single user or partner. Together, the Enterprise and Resource Tiers provide resources and services to all users:

- The Enterprise Tier is responsible for implementing business processes and entities and making their functions available via service-oriented interfaces.
- The Resource Tier is responsible for the management and access of shared enterprise resources such as legacy systems, packaged applications, and databases.

Thus the User and Workspace Tiers together support a single user, and there will be many instances of these tiers to support multiple users. The Enterprise and Resource Tiers support all users and there is typically one instance of these tiers in the enterprise.

We can now consider the Web service architecture in the context of tiers and layers, as shown in Figure 6 on page13. The infrastructure layer is based on a Web services platform that provides communications capabilities, such as HTTP and SOAP for external interaction, as well as other protocols such as IIOP for internal communications. This layer also provides basic services such as logging, configuration, and management.

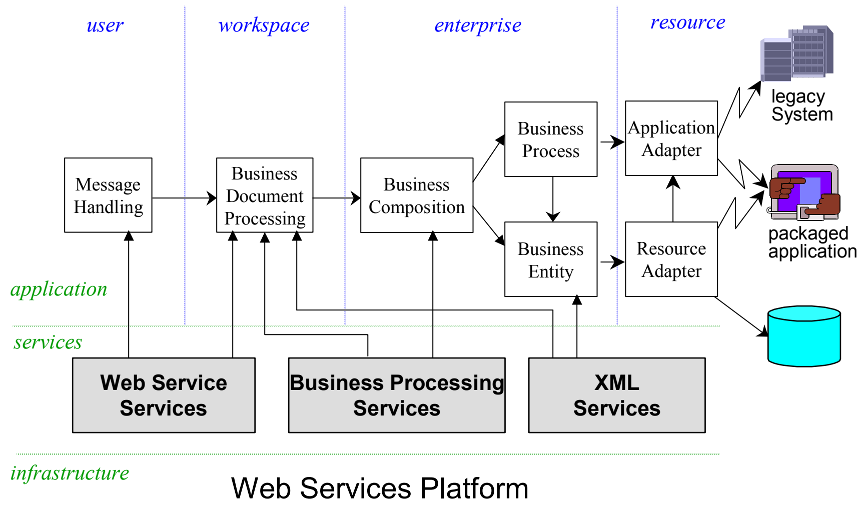

Figure 6: Advanced Web Services Architecture

The Service layer provides those common services that span multiple tiers, specifically services needed to support Web services, business processing, and XML processing.

Now consider the individual tiers of the application layer. The User Tier of the application layer provides for initial message handling and makes use of Web Service Services such as an *identity service*, which establishes shared identity between partners, and security services such as authentication.

The Workspace Tier processes the XML business document extracted from a received message. Document processing relies on XML services such as parsing, transformation, and persistence, as well as Business Processing Services such as the Business Process Engine for executing process definitions, and so on. Customization of processing also relies on the identity service and the shared context it provides.

The Enterprise Tier is responsible for implementing business functions, which may be exposed as business compositions. The compositions are constructed from primitive business processes and entities and this construction capability relies on the business processing and persistence services.

Finally, the Resource Tier exposes existing enterprise resources such as legacy systems, databases, and packaged applications, to the business processes in the enterprise tier.

As Figure 6 shows, there are three main groupings, or packages, within the services layer. The Web Services Service package provide capabilities specifically related to Web services, and includes:

- The identity service, which establishes shared identity between partners and which may provide customized processing for a specific partner.

- Service Level Agreement processing – Service Level Agreements are negotiated with each partner and specify things such as maximum and average response time, allowable down time, and so on. This service enforces and monitors these agreements.

- Security, which provides authentication and authorization, non-repudiation, and so on. Security may be tied to existing mechanisms within the enterprise.

- Business Transaction Services, which provide an all-or-nothing outcome for long-lived business collaborations. This is not the same as the tradition 2PC transaction support in many existing systems (where transactions are typically short-lived) but is built upon such systems.

The Business Process Package provides services for executing business process models, specifically:

- Business Process Engine, which can execute Business Process Models

- Auditing service to track each step in a process execution and to enable restart and recovery

Finally, the XML Service Package provides services related to XML processing:

- Parsing and creating XML Documents

- Transforming XML from one schema to another, or to and from a non-XML representation

- Persisting XML document state

3.1 Tiers and Responsibilities

The **User Tier** is responsible for performing security authentication and authorization (at least to the extent that User Tier controls whether a partner request is rejected or allowed to proceed), and then establishing the shared identity context for the principals involved in a collaboration. The User Tier is also responsible for enforcing and monitoring Service Level Agreements. Thus the User Tier is the gateway to the system and provides the initial message processing for an interaction.

The **Workspace Tier** has two main responsibilities. Firstly, it manages the session state for the user. This is particularly important for long-lived and asynchronous Web services.

Secondly, it requests services of the Enterprise Tier. But before Enterprise Tier services can be invoked, the data in the incoming document must be extracted and rationalized.

So the first step is to parse the XML document that was passed into the workspace. The document will apply to a specific business process or composition, which will have an associated Business Process Model. Both the document and the model will be passed to the BPM service for execution. Specific instances of the BPM may be customized for individual users (for example, different trading partners may require different data transformations or have different rules applied based on SLAs).

Activities, or steps, within the business process will require certain transformations, for example, between XML schemas or from XML to other formats, and these transformations are performed by the transformation service. Intermediate results may be stored using the persistence service. Rather than passing the entire XML document between each step of the process, a “persistence ID” may be passed so that each step can retrieve the appropriate data.

Thus the Workspace Tier primarily provides document processing functions.

The **Enterprise Tier** is responsible for providing business functionality and managing the integrity of enterprise resources. In doing so, it may enforce system-level business rules and begin (and thus delimit the scope of) traditional 2PC transactions.

The business function may be a relatively new capability, such as an EJB in the Enterprise Tier, but it is more likely that the business function is implemented in a legacy system or packaged application. In this case, the Enterprise Tier must interact with the Resource Tier.

The business function in the Enterprise Tier may itself be composed of finer-grain functions – in other words, it may be a business composition. Complex compositions may be expressed as Business Process Models, in which case the Enterprise Tier will also make use of the business processing services.

Thus, the Enterprise Tier is essentially responsible for making business functions available via service-oriented interfaces.

The **Resource Tier** is responsible for providing access to the system’s shared resources and applications. The responsibilities of this tier have remained largely unchanged as n-tier architectures have evolved over the past several years.

In making the resources of applications available, data transformation and manipulation may be required. This kind of processing may be quite complex as evidenced by the capabilities of many EAI systems today. It too will make use of model processing engines. In this case, the models to be executed tend to have more data integration steps than business processing steps, but can still be processed by the same engine and infrastructure.

In deciding how/what to map these resources to, we draw on the business model, which will have identified the fundamental business processes and entities. We use the constructs identified in the business model as a starting point, and map them onto the existing systems, rather than the other way around.

This point needs to be made very strongly. If we instead start with the existing systems and let them drive the definition of business processes and entities (as happens quite often), we tend to get new processes that look just like the existing processes. Software objects or components defined this way are likely to be brittle and offer very limited flexibility and reuse. Rather than having a set of fundamental reusable processes, the result is a much larger set of similar, overlapping, and inconsistent processes.

Mapping systems to the business model in this way also provides the opportunity to present their respective resources to the rest of the enterprise in some standard or canonical format. Typically, the canonical format includes some elements of the enterprise data model to represent the entities, and a technology standard, such as

EJB, to present a ubiquitous interface. Web services provide both the interface and the metadata support needed to define the entities

Canonical formats are increasingly provided via a Web service. For example, resource vendors like SAP and Oracle are planning to provide Web service interfaces directly into their applications.

In summary, the Resource Tier is basically responsible for making resources available to the enterprise.

3.2 The Process Automation Trap

We now see that process automation can be applied at several points during the processing of a business document request. Most commonly, it is used in processing the business document itself, and in the orchestration of a business collaboration. In complex enterprises it is also used at the application integration level.

A common architectural pattern in use today embraces the coincidental appearance of process automation in multiple places, and combines all of the automation-based processing into the same component. This process automation model or pattern would be better characterized as the process automation trap.

The problem is that it ignores what we as an industry have learned from building Web-based systems. The very first rule of architecture for modern systems is to separate presentation from business functionality. And even though it is relatively complex, the business document is just another form of presentation. And so, by combining document processing and business processing in the same model or pattern, we lose the ability to reuse the business logic in support of different access channels.

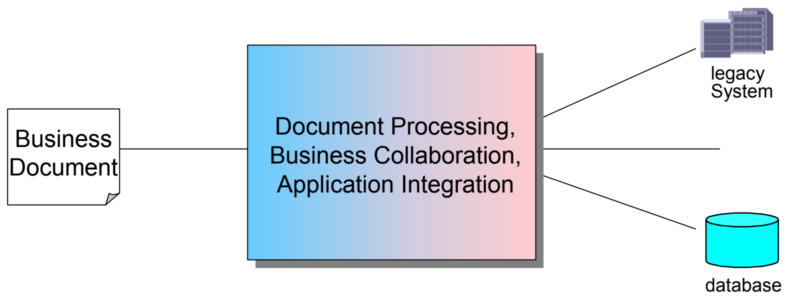
Business Document
Document Processing, Business Collaboration, Application Integration
legacy System
database

Figure 7: The Process Automation Trap

The result of this improper combination is shown in Figure 7, which should be contrasted with the architectural separation illustrated in Figure 8 and Figure 9. This is one of the reasons that a good architecture clearly separates the responsibilities of business document processing from those of providing business functionality. And there are more reasons for this separation when the entire enterprise is considered.

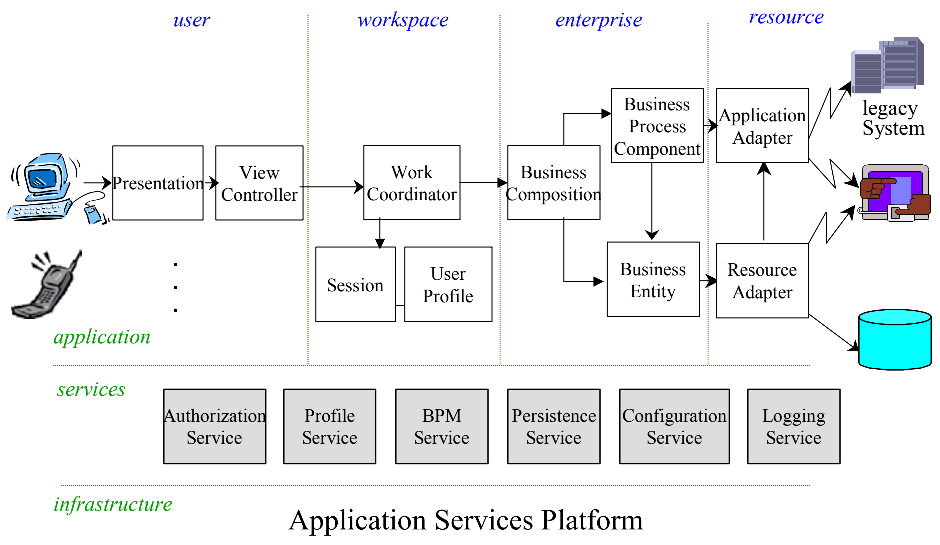

Figure 8: Web Interface Architecture

Figure 8 illustrates an architecture that we have been using for the past several years to implement Web-based user interface systems at clients in the insurance, telecom, finance, and manufacturing industries. It is based on the same three-layer, four-tier model described above. Each tier in this architecture has essentially the same responsibilities as in the Web services architecture described in Section 3.1 on page 14. Although the enterprise and resource tiers are the same, the implementation of responsibilities in the user and workspace tiers is completely different.

Web services promise to be the next technology revolution. But, like other advances that have come before it, Web services must work in the context of existing systems. We still operate and will continue to build traditional systems, and no one believes that Web services will replace GUI applications. Web services may well be integrated with GUI applications, but they are intended for a completely different use and will not replace them.

Again, enterprises will need to build both GUI and Web service applications. Many of these applications will need to share the same business logic, although the channels that access that logic will be very different. We can think of the user-workspace combination as providing that access channel to the business logic. The architecture supports many different access channels, which come together at the enterprise tier boundary to make common use of business services, as shown in Figure 9.

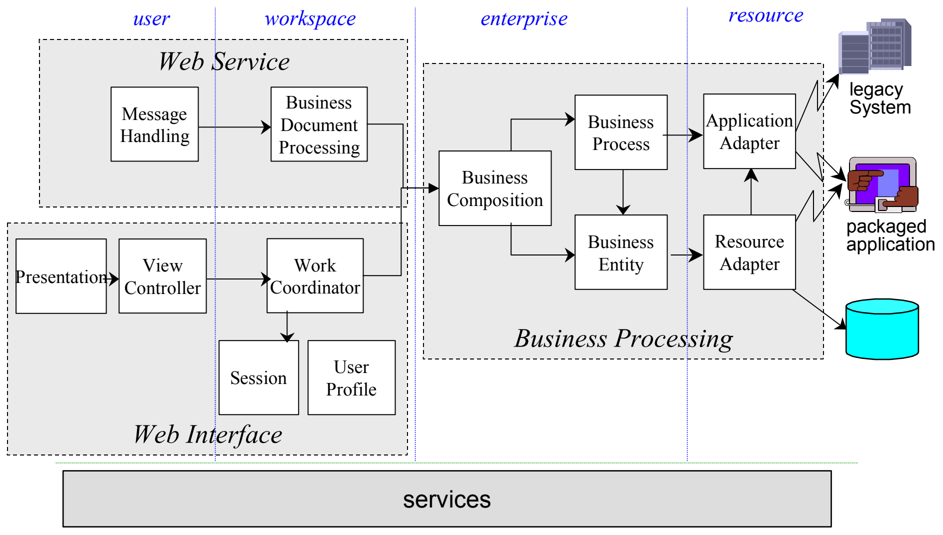

Figure 9: Combined Architecture

We have in fact implemented this aspect of the architecture in cooperation with several customers. These customers continue to build fat clients for some of their applications, but also need new Web based interfaces and functions. The user-workspace channel is implemented as a fat client in Delphi or VB, which shares business functions at the enterprise level with their new, Web base interfaces.

3.3 Web Services Enterprise Architecture

To summarize the main points of the Web services enterprise architecture:

It is implemented on three layers: infrastructure, services, and application. It is implemented in four tiers: user, workspace, enterprise and resource. These are logical tiers and layers, each with specific roles and responsibilities, and are orthogonal to physical distribution.

The Enterprise Tier provides business processes and compositions to all users of the enterprise. It does this by providing service-oriented interfaces to those processes, and works in coordination with the Resource Tier.

Individual users of the system use those business processes through an access channel made up of a User Tier / Workspace Tier combination. The most common access channels are for Web-based user interfaces, and now we must provide an access channel for Web services.

In the context of Web interfaces, the boundary between the User and Workspace Tiers provides for device independence; in other words, many different devices can access the same portal and display functions. This boundary also allows many different business documents (or different variations of the same document) to use common document processing capabilities.

Regardless of whether the user is a Web service, or a person using a Web browser, the same business processes are used, as shown in Figure 10.

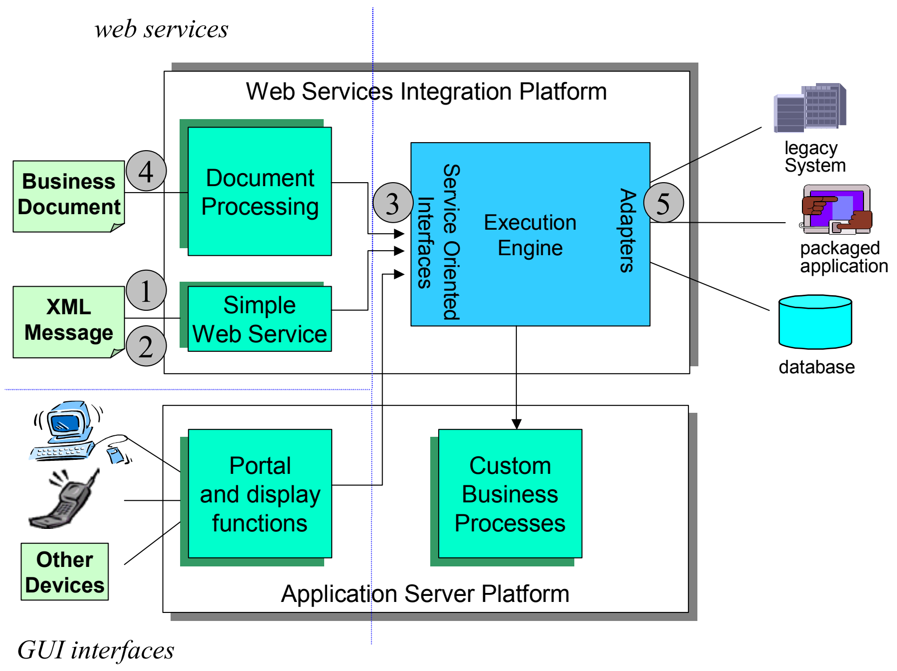

Figure 10: Enterprise Architecture

This figure shows that we can provide higher-level, higher-value business functions via service-oriented interfaces, instead of (or in addition to) exposing low-level classes as Web services. The rest of the architecture is unchanged. In other words, changing the level of the interface does not affect the underlying Web service technology architecture. This lets us use Web services at a variety of levels, and for a variety of purposes, within an enterprise.

The numbers in gray circles in Figure 10 correspond to the following usages:

1. Data Providers—simple single or multi-source data provider applications such as stock quotes, or financial portals

2. Subscription Based Notification—connections to wireless and other remote devices

3. Complex Business Functionality—ad-hoc interactions with consumers or trading partners

4. B2B Collaboration—standards-based (ebXML, RosettaNet, EDI, etc.), complex collaborations with trading partners

5. EAI—Integration within the enterprise

In other words, the same Web services technology can be use for simple and complex interactions both inside and outside of the enterprise. By applying the same technology to a wide range of problems, we can realize cost reductions in buying, deploying and especially operating software infrastructures.

3.4 Using Web Services for B2B Interaction

B2B collaboration between trading partners is characterized by the exchange of complex business documents over the Internet. B2B collaboration systems—that is, electronic exchanges and e-commerce marketplaces—were early pioneers in connecting trading partners and exchanging XML-based business documents over the Internet. Many of the techniques used in complex Web service applications come from these roots.

However, most B2B applications pre-dated open Web services standards, and this situation encouraged organizations such as RosettaNet and ebXML to create such open standards. These standards span several layers and include communication-level protocols such as SOAP and UDDI, and business-level protocols that deal with the exchange of specific documents. As Web services become more mature, these organizations are replacing lower-level, proprietary B2B communication protocols with standard Web services protocols.This means that B2B standards, and B2B standards bodies, can concentrate on higher level, value-added protocols that specify collaborations between trading partners, and leave the lower level communications protocols to Web service standards organizations. The value that B2B standards add include:

- Asynchronous, Reliable, Request/Response exchange of business messages
- Service Level agreements that specify a "business" level quality-of-service between trading partners
- Standard message exchange sequences (for example, request, acknowledgement, reply, acceptance).
- Standard schema and taxonomy.

The architecture presented in this paper is applicable to B2B applications, including those implemented via Web services, although some B2B specifics such as business protocol and trading partner management are excluded for simplicity.

3.5 Using Web Services for EAI

Service Oriented Architectures have been promoted for years as the best approach to achieving sustainable, evolvable Enterprise Application Integration solutions. Unfortunately, although EAI applications have been successful in point-to-point integration efforts, they have frequently failed in two major areas.

First, EAI solutions tended to be complex and expensive to implement because they relied on proprietary technology, which is expensive to acquire, operate and maintain. Worse, proprietary interfaces are usually difficult to implement and lead to vendor lock-in.

Second, these solutions tend to focus on data level integration, rather than application level (service-oriented) integration. Thus these implementations may solve specific problems but do not create generic or flexible processes that lend themselves to being reused in a variety of different, higher level business applications.

Web services provide a solution to these shortcomings in the EAI space. First, Web services are a standards-based solution. Vendor competition on standards-based software will result in increased choice of vendor and reduced cost for integrating applications. Further, Web service standards provide a higher-level solution for application integration because much of the lower level system programming has been commoditized into the Web service platforms themselves. This lowers both the cost and complexity of creating integration solutions.

Second, Web services promote the adoption of a service-based architecture within the enterprise. Web-services provide a low-cost and efficient way to expose business functionality with new interfaces. Coupled with a well-designed service-oriented architecture, this capability enables enterprises to effectively achieve reuse of business functionality and reduce time-to-market of new applications.

EAI systems address a different, but overlapping, problem space than the B2B and other Web service scenarios discussed above. The first step in implementing an EAI solution is to expose the existing enterprise functionality through a network-enabled interface. A Web Services Integration Platform then invokes that interface, using either a native mechanism or a messaging system like JMS.

Application data is formatted into a specially defined message and returned. A self-describing mechanism such as ASN.1 or XML is commonly used for packing the data into the message. The message format may be ad-hoc, or it may have been transformed into a system-defined canonical format.

Finally, a transformation to the target format is performed. This last step can be extremely complex, requiring message fan-in, fan-out, filtering, conditional evaluations, and even the need to request additional data.

The description capabilities of Web services, namely WSDL, can be used to describe and expose the existing enterprise functionality. WSDL is very flexible and powerful, and can describe additional information such as supported and potentially available transports. Then, the discovery capabilities of Web services can be used to dynamically locate application functions, both at design time and at run-time.

When the Web Services Integration Platform invokes the application, SOAP can be used to send the request and response messages. The message payloads (or data) can be defined in XML, which is the native format for SOAP. Because SOAP supports protocols other than HTTP, the SOAP messages can be sent over TCP/IP, JMS or any other appropriate protocol. Thus a protocol-aware system can use the additional information within the WSDL file to optimize communications within an enterprise.

Thus, Web services provide a standards-based solution to enterprise application integration that also promotes the creation of reusable enterprise assets. These standards simplify the creation of application integration “adapters,” and allow third party vendors to create integration components that can easily be used by the Web Service Integration Platform.

4 Industry Trends and Evolving Standards

It is usually a good idea to see how well an architecture lines up with events and trends in the industry. Today there is quite a bit of hype surrounding XML and Web services. But in spite of that, there is real value in using these technologies.

Web services are evolving to embrace the convergence of B2B, EAI, traditional middleware, and the Web. Web services are not a replacement for traditional middleware, but when used in combination with middleware and EAI techniques, Web services provide a simplified and standards-based approach to integration.

That simplicity promotes the use of Web services at many different levels in the enterprise. We have already mentioned that application vendors such as SAP and Oracle will soon offer direct Web services interfaces to their products. In addition, many large enterprises are using Web services for application-to-application integration within and across divisions. They expect, and are beginning to realize, simplifications and cost reductions based on using the same technology for both internal and external integration.

Some vendors are evolving toward a new, Web services-based integration platform that may well emerge as the foundation for future Web-based applications. Many of the capabilities that now exist in services layer of our architecture will be subsumed into those next-generation platforms.

Finally, the standards community is also busily at work on Web services. Some of the standards that should be tracked today, and which will affect our architecture are:

- Security
    - Security Services Markup Language (S2ML): S2ML enables secure e-commerce transactions using XML. It creates a common language for sharing security information in the context of B2B/B2C transactions. Authors of the S2ML specification are Bowstreet, Commerce One, Jamcracker, Netegrity, Sun Microsystems, VeriSign, and webMethods. Reviewers of the specification include Art Technology Group, Oracle, PricewaterhouseCoopers, and TIBCO.
    - 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 http://xml.coverpages.org/saml.html.
    - 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.
- Transactions

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

- Business Processes:
  - 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 Form (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.

- Business Standards:
  - RosettaNet: An independent business consortium of more than 400 companies dedicated to creating, implementing, and promoting, open e-business standards. The RosettaNet architecture includes Partner Interface Process (PIP) specifications, business and technical dictionaries, and the RosettaNet Implementation Framework (RNIF).
  - ebXML: ebXML is sponsored by UN/CEFACT and OASIS for global international standardization of business documents and processes. Work is ongoing in the areas of Messaging Services, Registries and Repositories, Collaborative Protocol Profile, and Implementation, Interoperability, and Conformance.

    Note that ebXML and RosettaNet standards are complementary; ebXML specifications are horizontal, while RosettaNet’s tend to be vertical. RosettaNet is adopting ebXML Business Process Schema to express their PIPs, and with RNIF 3.0, RosettaNet will use ebXML Messaging Services

- Business Modeling Standards:
  - UML Profiles: UML profiles are an OMG-defined mechanism for structuring extensions to UML. UML profiles support the definition of a UML vocabulary in support of a specific business domain such as e-commerce, or for specific implementation technologies such as EJB or CORBA.
  - Java Community Process (JCP): JCP is the way the Java platform evolves. It is an open organization of international Java developers and licensees whose charter is to develop and revise Java technology specifications, reference implementations, and technology compatibility kits. JCP was originally created by Sun Microsystems and has evolved from an informal process to a

formalized process overseen by representatives from many organizations across the Java community.

- ebXML Business Process Specification Schema (BPSS): BPSS provides the definition (in the form of an XML DTD) of an XML document that describes a business process. It identifies the roles, transactions, DTDs or schemas of business documents, document flow, legal considerations, security aspects, business level acknowledgments, and status. Such a Specification Schema can be used by a software application to configure the business details of conducting business electronically with another organization.

5 Architecture Review and Conclusions

We can now consider how well our architecture meets the principles and requirements of enterprise-class Web services as described in Section 2.

A strict separation of concerns is enforced by the architectural layers and tiers: by the explicit interfaces between layers, by the explicit boundaries between tiers, and by the explicit responsibilities of both tiers and layers.

Accommodation of change, for example, for future application versions, is achieved via strong support for the creation of business models, fundamental business process, and business compositions. And the services layer provides functions that enable the use of these mechanisms. The bottom line is that future versions and new applications can be created by building a few new processes, and then by recombining the new and existing process in different ways.

Industry trends are accommodated by using Web services at many different levels within the enterprise, and by preparing for the next-generation Web service integration platforms. Such integration platforms will support this architecture very well, and will significantly increase the portion of the services layer that can be purchased off-the-shelf.

New technologies and mechanisms are supported by the device and technology abstractions inherent in the layers and tiers. This allows new technologies to be incorporated into the enterprise without affecting business logic or disrupting existing applications.

The various emerging industry standards are encapsulated into specific services within the services layer. This minimizes the impact of change in these areas as standards are finalized or extended.

Finally, the architectural foundation of layers and tiers embody the fundamental mechanisms that promote consistency and reuse within the entire enterprise. And consistency and reuse in turn support the critical business requirements – of time to market, quality, and cost – that drive the adoption of technology in the first place.

The architecture described in this paper has been implemented and refined in numerous client engagements over the past four years. IONA Global Services has had the opportunity to validate its assumptions and assertions in real life situations and the correctness and applicability of this architecture has been proven.

IONA has witnessed how the consistency and reuse provided by this architecture helps current customers. They have experienced real and measurable improvements in time-to-market, quality and cost. There is every reason to believe that your enterprise can achieve similar benefits through the adoption of a Web services enterprise architecture.

6 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. *Preparing for Web Services White Paper,* December 2001.

7 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