

Migrating to a Service Oriented Architecture - Overview

Introduction - The Case for Developing a Service Oriented Architecture

Over the last four decades, software architectures have attempted to deal with increasing levels of software complexity. But the level of complexity continues to increase, and traditional architectures seem to be reaching the limit of their ability to deal with the problem. At the same time, traditional needs of IT organizations persist; the need to respond quickly to new requirements of the business, the need to continually reduce the cost of IT to the business, and the ability to absorb and integrate new business partners and new customer sets, to name a few. As an industry, we have gone through multiple computing architectures designed to allow fully distributed processing, programming languages designed to run on any platform and greatly reduce implementation schedules, and a myriad of connectivity products designed to allow better and faster integration of applications. However, the complete solution continues to elude us. Now Service Oriented Architecture (SOA) is being promoted in the industry as the next evolutionary step in software architecture to help IT organizations meet their ever more complex set of challenges. Is it real, though, and even if it can be outlined and described, can it really be implemented? The thesis of this paper is that the promise of SOA is true; that after all the hype has subsided, and all the inflated expectations have returned to reality, we will find that an SOA, at least for now, is the best foundation upon which an IT organization can take its existing assets into the future, and build its new application systems as well. This is the first in a series of papers intended to help customers better understand the value of an SOA, and to develop a realistic plan for evaluating their current infrastructure and migrating it to a true Service Oriented Architecture.

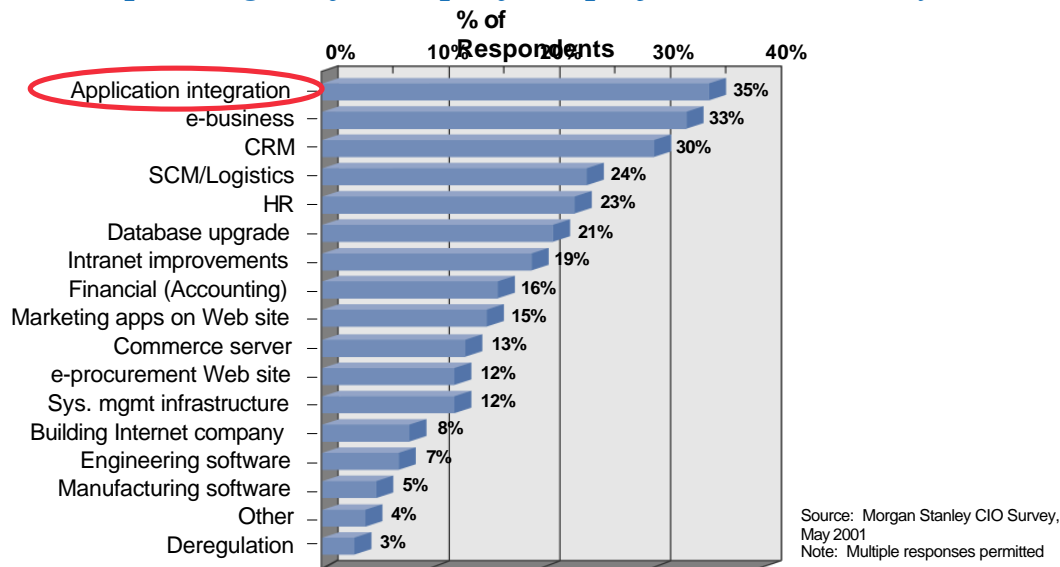
For some time now, the existence of Web services technologies has stimulated the discussion of Services Oriented Architectures (SOAs). The discussion isn't a new one; the concept has been developing for more than a decade now, ever since CORBA extended the promise of integrating applications on disparate heterogeneous platforms. Problems integrating those applications have always arisen, often because so many different (and non CORBA compliant) object models became popular; thus many architects and engineers became so bogged down in solving technology problems that the promise of developing a more robust architecture that would allow simple, fast, and secure integration of systems and applications was lost. The problems, however, persist, and become more complex every year. Basic business needs such as lowering costs, reducing cycle times, integration across the enterprise, B2B and B2C integration, greater ROI, creating an adaptive and responsive business model, and so on keep us looking for better solutions; but more and more, we are finding that "point solutions" won't solve the basic problem. The problem, in many cases, is the lack of a consistent architectural framework within which applications can be rapidly developed, integrated, and reused. More importantly, we need an architectural framework which allows the assembly of components and services for the rapid, and even dynamic, delivery of solutions. Many papers have been written about why particular technologies such as Web services are good, but what is needed is an architectural view unconstrained by technology. Let's begin by considering some of the fundamental problems that underlie our search for a better foundation, for how these problems are addressed will determine the success or failure of the effort.

Migrating to a Service Oriented Architecture - Overview

The First Problem - Complexity

Some things are always the same, particularly the business problems facing IT organizations. Corporate management always pushes for better IT utilization, greater ROI, integration of historically separate systems, and faster implementation of new systems; but some things are different now. Now we find more complex environments. Legacy systems must be reused rather than replaced, because with even more constrained budgets, replacement is cost-prohibitive. We find that cheap, ubiquitous access to the Internet has created the possibility of entire new business models, which must at least be evaluated since our competition is already doing it. Growth by merger and acquisition has become standard fare, so entire IT organizations, applications, and infrastructures must be integrated and absorbed. In an environment of this complexity, point solutions merely exacerbate the problem, and will never lead us out of the woods. Systems must be developed where heterogeneity is fundamental to the environment, because they must accommodate an endless variety of hardware, operating systems, middleware, languages, and, oh yes, data stores. The cumulative effect of decades of growth and evolution has produced the complexity that now tortures us. With all these business challenges for IT, it is no wonder that application integration tops the priority list of many CIOs:

Top strategic software platform project over the next year



Another Problem - Redundant and Non-Reusable Programming

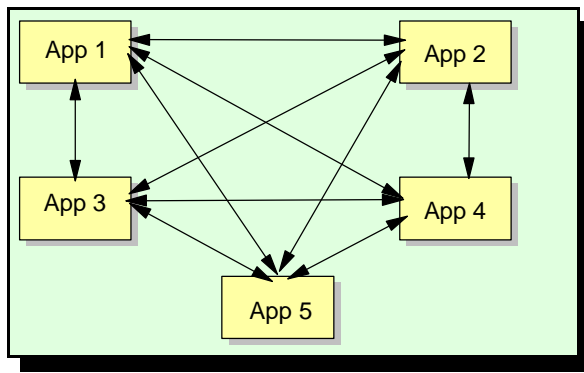
Consider a bank that has separate "silos" - self contained application systems that are oblivious to other systems within the bank. The first of these application systems may have been an excellent design, as well as the second, third, and so on, but each was produced by and for a different line of business within the bank, and was a separately funded, isolated project. Thus, for example, the function of "get account balance" is repeated in the ATM system, the branch teller delivery system, and the

Migrating to a Service Oriented Architecture - Overview

credit card scoring system, even if they access the same account data in the same database. Now suppose the bank must develop an Internet service, on-line banking, or an on-line loan origination system for its customers if it is to remain competitive. The new system will just add to the problem of all the redundant programming already in place, unless somehow the existing code can be reused.

The Real Integration Killer - Multiplicity of Interfaces

Consider also the $n(n-1)$ integration problem. All organizations face integration problems of some sort; perhaps because of a corporate merger, a new business alliance, or just the need to interconnect existing systems. If n application systems must be directly interconnected, it will produce $n(n-1)$ connections, or interfaces. In this diagram, each arrowhead represents an interface:



Consequently, if another application system A^{n+1} must be integrated, it will require that $2n$ new interfaces be generated, documented, tested, and maintained. While in the diagram above, the set of five applications require 20 direct interfaces, the addition of a sixth application will require ten new interfaces! Worse yet, the code in each of the existing applications must be modified to include the new interfaces, thus generating substantial testing costs. Immediately, we look for the optimum solution that produces the minimum number of interfaces (n) for n applications, with only one new interface for each additional system added, but find that it can't be done by direct connection.

And What of the Future?

Over the last four decades the practice of software development has gone through several different programming models. Each shift was made in part to deal with greater levels of software complexity and to enable the assembly of applications through parts, components, or services. More recently, Java gave us platform-neutral programming, and XML gave us self-describing, and thus platform-neutral, data. Now Web services has removed another barrier by allowing the interconnection of applications in an object-model-neutral way. Using a simple XML-based messaging scheme, Java applications can invoke DCOM-based, CORBA-compliant, or even COBOL applications. CICS or IMS transactions on a mainframe in Singapore can be invoked by a COM-based application driven by Lotus Script running on a Domino server in Munich. Best of all, the invoking application likely has no

Migrating to a Service Oriented Architecture - Overview

idea where the transaction will run, nor what language it is written in, nor what route the message may take along the way. A service is requested, and an answer is provided.

Web services are more likely to be adopted as the de facto standard to deliver effective, reliable, scalable, and extensible machine-to-machine interaction than any of its predecessors, as a result of the timely convergence of several necessary technological and cultural prerequisites. These include:

- A ubiquitous, open-standard, low cost network infrastructure, and technologies that make for a distributed environment much more conducive to the adoption of Web services than both CORBA and DCE faced
- A degree of acceptance and technological maturity to operate within a network-centric universe that requires interoperability in order to achieve critical business objectives, such as distributed collaboration
- Consensus that low-cost interoperability is best achieved through open Internet-based standards and related technologies
- The maturity of network-based technologies (eg, TCP/IP), tool sets (IDE's, UML, etc.), platforms (eg., J2EE), and related methodologies (.e.g. OO, services, etc.), that provide the infrastructure needed to facilitate loosely-coupled and interoperable machine-to-machine interactions -- a state far more advanced than what CORBA users experienced.

Service Oriented Architecture allows designing software systems that provide services to other applications through published and discoverable interfaces, and where the services can be invoked over a network. When we implement a Service Oriented Architecture using Web services technologies, we create a new way of building applications within a more powerful, and flexible programming model. Development and ownership costs are reduced as well as implementation risks. SOA is both an architecture and a programming model, a way of thinking about building software.

On the horizon, however, are even more significant opportunities. First, there is Grid Computing, which is much more than just the application of massive numbers of MIPS to effect a computing solution; it also will provide a framework whereby massive numbers of *services* can be dynamically located, relocated, balanced, and managed so that needed applications are always guaranteed to be securely available, regardless of the load placed on the system. This, in turn, makes obvious the need for the concept of On-Demand Computing, which might be implemented on any configuration, from a simple cluster of servers to a network of 1024-node SP2s. The user needs to solve a problem, and wants the appropriate computing resources applied to it - no more, no less - and only pay for the resources actually used.

The effective use of these new capabilities will require the restructuring of many existing applications. Existing monolithic applications can run in these environments, but will never use the available resources in an optimal way. This, along with the problems previously discussed, leads us to the conclusion that a fundamental change must be made - the conversion to a Service Oriented Architecture.

Migrating to a Service Oriented Architecture - Overview

Requirements For a Service Oriented Architecture

From the problems discussed above, it should be clear that an architecture should be developed that meets *all* of our requirements, and that those requirements include:

- 1). First and foremost, *leverage existing assets*. Existing systems can rarely be thrown away, and often contain within them great value to the enterprise. Strategically, the objective is to build a new architecture that will yield all the value that we hope for, but tactically, the existing systems must be integrated such that, over time, they can be componentised or replaced in manageable, incremental projects.
- 2). *Support all required types or "styles" of integration*. This includes:
 - User Interaction – being able to provide a single, interactive user experience
 - Application Connectivity – communications layer that underlies all of the architecture
 - Process Integration – choreographs applications and services
 - Information Integration – federates and moves the enterprise data
 - Build to Integrate – builds and deploys new applications and services
- 3). *Allow for incremental implementations and migration of assets* - this will enable one of the most critical aspects of developing the architecture: the ability to produce incremental ROI. Countless integration projects have failed due to their complexity, cost, and unworkable implementation schedules.
- 4). Include a development environment that will be built around *a standard component framework*, promote better reuse of modules and systems, allow legacy assets to be migrated to the framework, and allow for the timely implementation of new technologies.
- 5). *Allow implementation of new computing models*; specifically, new portal-based client models, grid computing, and on-demand computing.

A Service Oriented Architecture - Not Just Web Services

The advent of Web services has produced a fundamental change, because the success of many Web services projects has shown that the technology does in fact exist, whereby we can implement a true Service Oriented Architecture. It allows us to take even another step back and not just examine our application architecture, but the basic business problems we are trying to solve. From a business perspective, it's no longer a technology problem, it is a matter of developing an application architecture and framework within which business problems can be defined, and solutions can be implemented in a coherent, repeatable way.

Migrating to a Service Oriented Architecture - Overview

First, though, it must be understood that “Web services” does not equal “Service Oriented Architecture”. Web services is a collection of technologies, including XML, SOAP, WSDL, and UDDI, which allow us to build programming solutions for specific messaging and application integration problems. Over time, we can reasonably expect these technologies to mature, and eventually be replaced with better, more efficient, or more robust ones, but for the moment, they will do. They are, at the very least, a proof of concept that SOAs finally can be implemented. So what actually does constitute a Service Oriented Architecture?

SOA is just that, an architecture. It is more than any particular set of technologies, such as Web services, it transcends them, and in a perfect world, is totally independent of them. Within a business environment, a pure architectural definition of an SOA might be something like “an application architecture within which all functions are defined as independent services with well defined invocable interfaces, which can be called in defined sequences to form business processes”. Note what is being said here:

- 1). All functions are defined as *services*. This includes purely business functions, business transactions composed of lower-level functions, and system service functions. This brings up the question of granularity, which will be addressed later.
- 2). All services are *independent*. They operate as “black boxes”; external components neither know nor care how they perform their function, merely that they return the expected result.
- 3). In the most general sense, the interfaces are *invokable*; that is, at an architectural level, it is irrelevant whether they are local (within the system) or remote (external to the immediate system), what interconnect scheme or protocol is used to effect the invocation, or what infrastructure components are required to make the connection. The service may be within the same application, or in a different address space within an asymmetric multiprocessor, on a completely different system within the corporate Intranet, or within an application in a partner’s system used in a B2B configuration.

In all this, the interface is the key, and is the focus of the calling application. It defines the required parameters and the nature of the result; thus, it defines the nature of the service, not the technology used to implement it. It is the system’s responsibility to effect and manage the invocation of the service, not the calling application. This allows two critical characteristics to be realized: first, that the services are truly independent, and second, that they can be managed. Management includes many functions, including:

- a). Security - authorization of the request, encryption and decryption as required, validation, etc.
- b). Deployment - allowing the service to be redeployed (moved) around the network for performance, redundancy for availability, or other reasons.
- c). Logging - for auditing, metering, etc.

Migrating to a Service Oriented Architecture - Overview

- d). Dynamic rerouting - for fail over or load balancing
- e). Maintenance - management of new versions of the service

The Nature of a "Service"

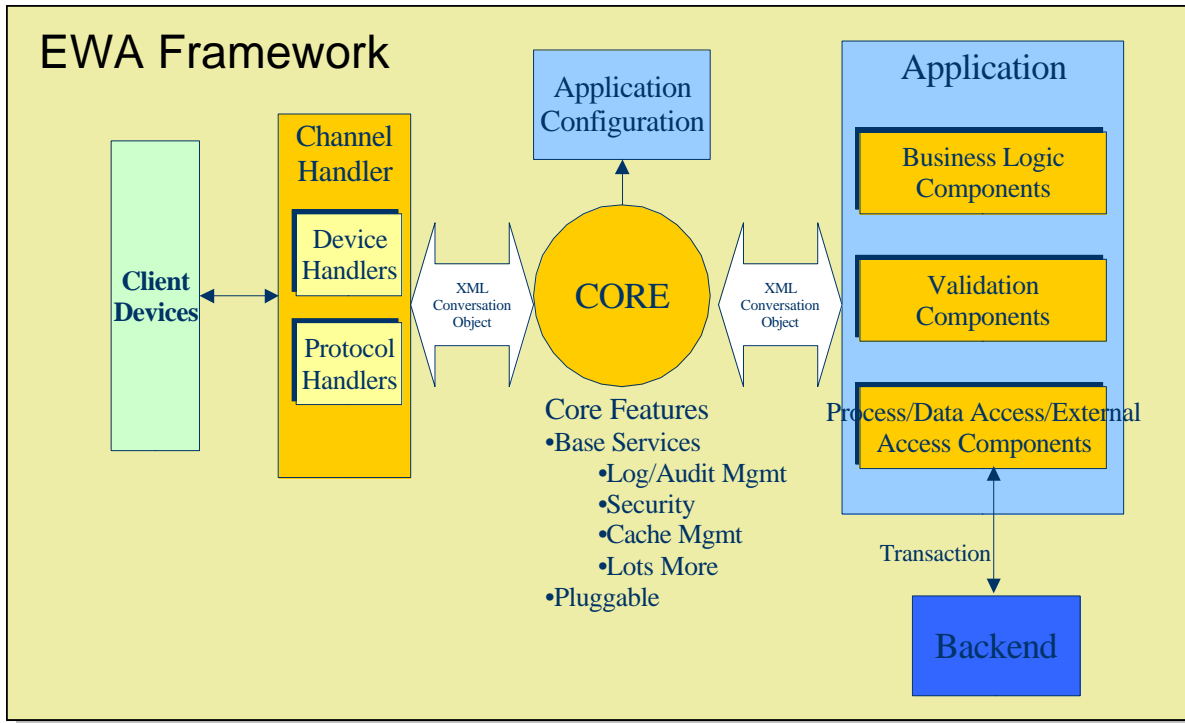
What then is a service? As previously stated, typically within a business environment, that means business functions, business transactions, and system services. Examples of business functions might be: **getStockQuote**, **getCustomerAddress**, or **checkCreditRating**. Examples of business transactions might be **commitInventory**, **sellCoveredOption**, **scheduleDelivery**. Examples of system services might be **logMessageIn**, **getTimeStamp**, **openFile**. Note the difference in the types of services. Business functions are, from the application's perspective, non-system functions that are effectively atomic. Business transactions may seem like a simple function to the invoking application, but they may be implemented as composite functions covered by their own transactional context. They may involve multiple lower-level functions, transparent to the caller. System functions are generalized functions that can be abstracted out to the particular platform, for instance, Windows or Linux. A generic function such as **openFile** might be provided by the application framework to effectively virtualize the data source, and used regardless of the type and location of the real source of the data.

This may seem like an artificial distinction of the services; one could assert that from the application's perspective, all the services are atomic, and it is irrelevant whether they are business or system services. The distinction is made merely to introduce the important concept of *granularity*. The decomposition of business applications into services is not just an abstract process; it has very real practical implications. Services may be low-level or complex high-level (fine-grained or course-grained) functions, and there are very real tradeoffs in terms of performance, flexibility, maintainability, and re-use, based on their definition. This process of defining services is normally accomplished within a larger scope - that of the Application Framework. This is the actual work that must be done; that is, the development of a component-based Application Framework, wherein the services are defined as a set of reusable components that can in turn be used to build new applications, or integrate existing software assets.

There are many such frameworks available today; within IBM, several frameworks such as EWA, JADE, and Struts (from Jakarta) are being used in customer integration scenarios. Taking EWA (pronounced "Eva"), from IBM Software Group's Advanced Technology Solutions team, for example, at a very high level, the framework looks like the diagram below. Within this framework, an application is defined by a configuration, which describes the components of the application, as well as the sequence and method of their invocation. Input is received and passed to the application in a source-neutral way, so, for instance, the addition of an Internet connection to a bank application with existing ATM access is transparent to the application logic. The front-end device and protocol handlers make that possible. System level services are provided by the core, and special-purpose access components enable connection to backend enterprise applications, so that they may remain in place, or

Migrating to a Service Oriented Architecture - Overview

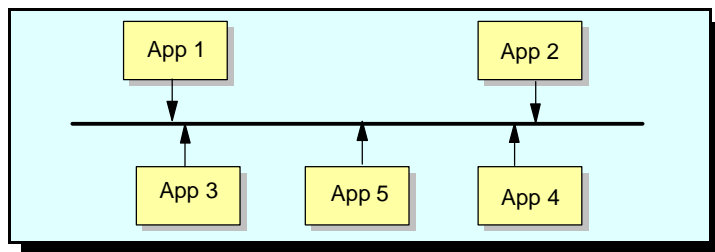
be migrated over time. While EWA is fully J2EE compliant, it can connect to external DCOM or CORBA component based systems.



Today, EWA contains over 1500 general and special-purpose components, thus greatly reducing the amount of code that must be written for a new application. Another paper in this series will examine Application Frameworks in detail, along with what a user may expect in the process of developing one.

Addressing the Old Problems

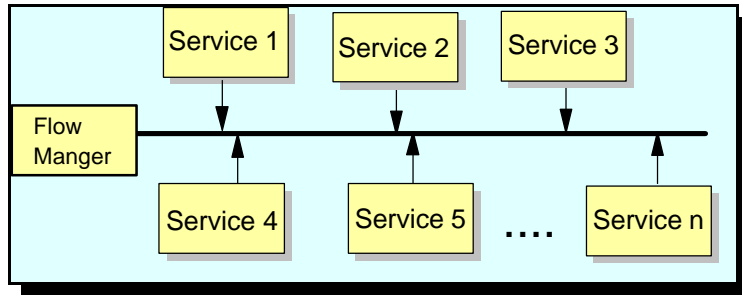
Returning now to the first integration scenario we discussed, and the search for a scheme that minimizes the number of required interfaces such as is drawn in the following figure:



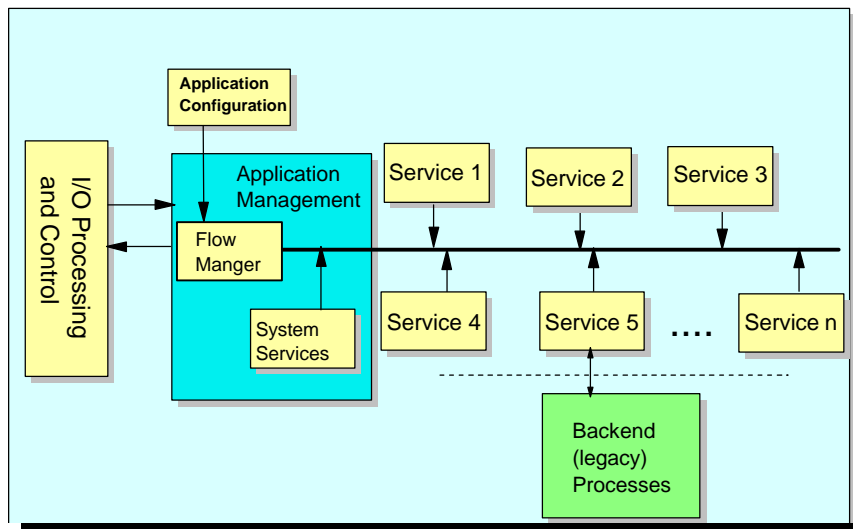
This may look like an overly simplistic view, but it should now be clear that within a framework such as EWA, this view is the starting point. Now we add the architectural concept of the Service Bus,

Migrating to a Service Oriented Architecture - Overview

represented below by the heavy center line, and a service or flow manager to connect the services and provide a path for service requests. The flow manager processes a defined execution sequence, or service flow, that will invoke the required services in the proper sequence to produce the final result. The Business Process Execution Language, or BPEL, is an example of such a technology for defining a process as a set of service invocations.



From here, we need to determine how to call the services, so we add application configuration. Next, virtualize the inputs and outputs. Finally, provide connectivity to backend processes, thus allowing them to run as-is, and allowing migration in the future. Now the high-level picture is at least structurally complete, and looks like this:



It should not be at all surprising that this picture bears some resemblance to a block diagram of EWA; at the highest level, any robust application framework must provide these functions. From here, however, the real work begins: building the 1500 components that put flesh on this skeleton. This is why many IT architects choose to implement within an existing framework; the process of decomposing the existing applications into components for the framework is work enough, without reinventing all the other general-purpose and system components known to be needed. However we approach it, the

Migrating to a Service Oriented Architecture - Overview

architecture can be implemented using technologies and frameworks that exist today, and so we come full circle, back to the beginning, where the process starts with an analysis of the business problems that must be solved. We can do this now, confident in the knowledge that our architecture will be, in fact, implementable.

Integration Requirements Within The Architecture

So far in this discussion, integration has been confined to application integration via component based services, but integration is a much broader topic than this. When assessing the requirements for an architecture, several types or "styles" of integration must be considered. One must consider not only application integration, but also integration at the end user interface, application connectivity, process integration, information integration, and a "build to integrate" development model.

Integration at the end user interface is concerned with how the complete set of applications and services a given user accesses are integrated to provide a usable, efficient, and consistent interface. It is an evolving topic, and the new developments, for the near term, will be dominated by advances in the use of portal servers. While portlets can already invoke local service components via Web services, new technologies, such as Web Services for Remote Portlets will enable content and application providers to create interactive services that plug and play with portals via the Internet, and thereby open up many new integration possibilities.

Application connectivity is an integration style concerned with all types of connectivity that must be supported by the architecture. At one level, this means things such as synchronous and asynchronous communications, routing, transformation, high speed distribution of data, and gateways and protocol converters. On another level, it also relates to the virtualization of input and output, or sources and sinks, as we saw in EWA's Channel and Protocol Handlers. Here the problem is the fundamental way data moves in and out of, and within, the framework that implements the architecture.

Process integration is concerned with the development of computing processes that map to and provide solutions for business processes, integration of applications into processes, and integrating processes with other processes. The first requirement may seem trivial, that is, that the architecture allow for an environment within which the basic business problems can be modeled, but insufficient analysis at this level will spell doom for any implementation of the architecture, regardless of its technical elegance. Integration of applications into processes may include applications within the enterprise, or may involve invocation of applications or services in remote systems, perhaps those of a business partner. Likewise, process level integration may involve the integration of whole processes, not just individual services, from external sources, such as supply chain management or financial services that span multiple institutions. For such application and process integration needs, technologies such as BPEL4WS may be utilized, or the application framework may use a program configuration scheme such as the one seen in EWA. In fact, a higher-level configuration scheme may be constructed using BPEL4WS at a lower level, and then driven by an engine that provides more function than just flow

Migrating to a Service Oriented Architecture - Overview

management. Before any of this is built, however, the architectural requirements must be understood first, and then the appropriate infrastructure built.

Information integration is the process of providing a consistent access to all the data in the enterprise, by all the applications that need it, in whatever form they need it, without being restricted by the format, source, or location of the data. This requirement, when implemented, may involve "adapters" and a transformation engine, but typically it is more complex than that. Often the key concept is the virtualization of the data, which may involve the development of a "data bus" from which data is requested using standard services or interfaces by all applications within the enterprise. Thus the data can be presented to the application regardless of whether it came from a spreadsheet, a native file, an SQL or DL/I database, or an in-memory data store. The format of the data in its permanent store may also be unknown to the application. The application is further unaware of the operating system that manages the data, so native files on an AIX or Linux system are accessed the same way they would be on Windows, OS/2, ZOS, or any other system. The location of the data is likewise transparent; since it is provided by a common service, it is the responsibility of the access service to retrieve the data, locally or remotely, not the application, and then present the data in the requested format.

Lastly, one of the requirements for the application development environment must be that it takes into account all the styles and levels of integration that may be implemented within the enterprise, and provide for their development and deployment. To be truly robust, the development environment must include (and enforce) a methodology that clearly prescribes how services and components are designed and built, to facilitate reuse, eliminate redundancy, and simplify testing, deployment, and maintenance.

All of the styles of integration listed above will have some incarnation within any enterprise, even though in some cases they may be simplified or not clearly defined; thus they must all be considered when embarking on a new architectural framework. A given IT environment may have only a small number of data source types, so information integration may be straightforward. Likewise, the scope of application connectivity may be limited. Even so, the integrating functions within the framework must still be provided by services, rather than being performed ad hoc by the applications, if the framework is to successfully endure the growth and changes over time that all enterprises experience.

Benefits of Deploying a Service Oriented Architecture

An SOA can be evolved based on existing system investments rather than requiring a full-scale system rewrite. Organizations that focus their development effort around the creation of services, using existing technologies, combined with the component-based approach to software development will realize several benefits:

- **Leverage existing assets** -- This was the first, and most important, of our requirements. A business service can be constructed as an aggregation of existing components, using a suitable SOA framework and made available to the enterprise. Using this new service only requires knowing its

Migrating to a Service Oriented Architecture - Overview

interface and name. The service's internals are hidden from the outside world, as well as the complexities of the data flow through the components that make up the service. This component anonymity lets organizations leverage current investments, building services from a conglomeration of components built on different machines, running different operating systems, developed in different programming languages. Legacy systems can be encapsulated and accessed via Web service interfaces.

- **Infrastructure, A Commodity** -- Infrastructure development and deployment will become more consistent across all the different enterprise applications. Existing components, newly developed components, and components purchased from vendors can be consolidated within a well-defined SOA framework. Such an aggregation of components will be deployed as services on the existing infrastructure resulting in the underlying infrastructure beginning to be considered more as a commodity element.
- **Faster time-to-market** -- Organizational Web services libraries will become the core asset for organizations adapting the SOA framework. Building and deploying services with these Web services libraries will reduce the time-to-market dramatically, as new initiatives reuse existing services and components, thus reducing design, development, testing and deployment time.
- **Reduced cost** -- As business demands evolve and new requirements are introduced, the cost of enhancing and creating new services by adapting the SOA framework and the services library, for both existing and new applications, is greatly reduced. The learning curve for the development team is reduced as well, as they may already be familiar with the existing components.
- **Risk mitigation** -- Reusing existing components reduces the risk of introducing new failures into the process of enhancing or creating new business services. As mentioned earlier, there is a reduced risk in the maintenance and management of the infrastructure supporting the services, as well.
- **Continuous Business Process Improvement** -- An SOA allows a clear representation of process flows identified by the order of the components used in a particular business service. This provides the business users with an ideal environment for monitoring business operations. Process modeling is reflected in the business service. Process manipulation is achieved by reorganizing the pieces in a pattern (components that constitute a business service). This would further allow for changing the process flows while monitoring the effects, and thus facilitates continuous improvement.
- **Process-centric Architecture** -- The existing architecture models and practices tend to be program-centric. Applications are developed for the programmer's convenience. Often, process knowledge is spread between components. The application is much like a black box, with no granularity available outside it. Reuse requires copying code, incorporating shared libraries, or inheriting objects. In a process-centric architecture, the application is developed for the process. The process is decomposed into a series of steps, each representing a business service. In effect, each service or component functions as a sub-application. These sub-applications are chained

Migrating to a Service Oriented Architecture - Overview

together to create a process flow capable of satisfying the business need. This granularity lets processes leverage and reuse each sub-application throughout the organization.

And The Future - New Models, New Requirements

So far, this discussion has centered around concepts related to meeting existing business requirements, better utilization and reuse of resources, and integration of existing and new applications. But what if a completely new model for application development emerges? Will the notion of a Service Oriented Architecture still be meaningful or required? Actually two new concepts are already beginning to be implemented: Grid Computing and On-demand Computing. While these models are distinct and have developed separately, they are closely related, and each make the evolution to an SOA even more imperative.

Grid Computing

An in-depth discussion of Grid Computing is beyond the scope of this introduction, but a couple of points are worth mentioning. First of all, Grid Computing is much more than just the application of large numbers of MIPS to effect a computing solution to a complex problem. It involves the virtualization of all the system resources including hardware, applications, and data, so that they can be utilized wherever and however they are needed within the "grid". Secondly, previous sections have already discussed the importance of virtualization of data sources and the decomposition of applications into component based services, so it should be easily understood that a true SOA should better enable getting maximum resource utilization in a grid environment.

On-Demand Computing

On Demand is also not in the scope of this discussion but again we would be remiss in not providing a brief connection between On Demand and SOA. Web services is an enabling technology for SOA, and SOA is an enabling architecture for On Demand applications. Applications must operate in an SOA framework in order to realize the benefits of On Demand.

Web Services On Demand is a subset of the On Demand message which covers a wide spectrum. On one end of this spectrum we have a focus on the application environment, and on the other end, a focus on the operating environment which includes items like infrastructure and autonomic computing. Business transformation leverages both the application and operating environments to create an on demand business. At the heart of this on demand business will be Web Services on demand where application level services can be discovered, reconfigured, assembled and delivered on demand with "just-in-time" integration capabilities.

The promise of Web services as an enabling technology is that it will enhance business value by providing capabilities such as services on demand, and over time, will transform the way IT organizations develop software. It quite possibly may even transform the way business is conducted and products and services are offered over the web in communities of interest that include trading

Migrating to a Service Oriented Architecture - Overview

partners, customers and other types of business partnership. What if all of your applications shared the same transport protocol? What if they all understood the same interface? What if they could participate in, and understood the same transaction model? What if this were true of your partners? Then you would have applications and an infrastructure to support an ever changing business landscape, you would have achieved On Demand. Web Services and SOA make this possible for applications.

Summary

Service Oriented Architecture is the next wave of application development. Services and SOA are all about designing and building systems using heterogeneous network addressable software components. SOA is an architecture with special properties, comprised of components and interconnections that stress interoperability and location transparency. It often can be evolved based on existing system investments rather than requiring a full scale system rewrite; it leverages an organization's existing investment by taking advantage of current resources, including developers, software languages, hardware platforms, databases, and applications, and will thus reduce costs and risks while boosting productivity. This adaptable, flexible style of architecture provides the foundation for shorter time-to-market and reduced costs and risks in development and maintenance. Web services is a set of enabling technologies for SOA, and SOA is becoming the architecture of choice for development of responsive, adaptive new applications.

About the Authors:

Kishore Channabasavaiah received a Bachelors degree in Mechanical Engineering from the Bangalore University, India. He is currently an Executive Architect in the Chicago Innovation Center of IBM Global Services. He provides thought leadership for eBusiness Integration solutions with focus on Web Services and end-to-end solutions. His current focus is in web application solutions, conducting technical solution reviews, Web Services, Service Oriented Architecture and Pervasive Computing.

Kerrie Holley received a Bachelor of Arts in Mathematics degree and a Juris Doctorate in law degree from DePaul University. He is currently a Distinguished Engineer in IBM Global Services and a Chief Architect in the eBusiness Integration Solutions where he provides thought leadership for the Web Services practice. His current focus is in software engineering best practices, end-to-end advanced web development, adaptive enterprise architecture, conducting architecture reviews, Web services and Service Oriented Architecture.

Edward M. Tuggle, Jr. received a Bachelor of Science in Mathematics degree from the University of Oklahoma, and is currently a Senior Software Engineer on IBM Software Group's jStart Emerging Technology Solutions team. He worked with IBM in operating systems design, development, and maintenance for twenty three years, for the past six years in Java and other emerging technologies, and is now specializing in Web services and Service Oriented Architecture.