On demand operating environment solutions
White paper

Migrating to a service-oriented architecture

by Kishore Channabasavaiah and Kerrie Holley, IBM Global Services, and Edward M. Tuggle, Jr., IBM Software Group

April 2004

Contents

Introduction: the case for developing a service-oriented architecture

Over the last four decades IT systems have grown exponentially, leaving companies to handle increasingly complex software architectures. Traditional architectures have reached the limit of their capabilities, while traditional needs of IT organizations persist. IT departments still need to respond quickly to new business requirements, continually reduce the cost of IT to the business and seamlessly absorb and integrate new business partners and customers. The software industry has 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 be elusive.

Now, service-oriented architectures (SOAs) are being promoted as the next evolutionary step to help IT organizations meet their ever more-complex challenges. But questions remain: Are SOAs real? And even if they can be outlined and described, can they actually be implemented? This white paper discusses how the promise of SOA is true. That after all the publicity has subsided, and all the inflated expectations have returned to reality, IT organizations will find that SOAs provide the best foundation upon which an IT organization can build new application systems, while continuing to capitalize on existing assets. This white paper is the first in a series intended to help you better understand the value of an SOA, and to help you develop a realistic plan for evaluating your current infrastructure and migrating it to a service-oriented architecture.

For some time now, the existence of Web services technologies has stimulated the discussion of 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 these disparate applications arose, often because so many different (and non-CORBA-compliant) object models became popular. As a result, many architects and engineers became so bogged down in solving technology problems that developing a more robust architecture that would allow simple, fast, and highly secure integration of systems and applications was lost. Unfortunately, the problems persist, and become more complex every year. Meeting basic business needs drive your search for a better solution. Needs like lowering costs, reducing cycle times, integrating systems across your enterprise, integrating business-to-business (B2B) and business-to-consumer (B2C) systems, achieve a faster return on your investment, and creating an adaptive and responsive business model. But more and more, you're finding that point-to-point solutions won't solve the basic problem: the lack of a consistent architectural framework that enables you to rapidly develop, integrate and reuse applications. More importantly, you need an architectural framework that allows you to assemble components and services to deliver dynamic solutions as your business needs evolve. This white paper will go beyond discussing why particular technologies such as Web services are good. It will provide an architectural view unconstrained by technology. To begin, you should consider some of the fundamental problems that underlie your search for a better foundation. How you address these problems will determine your level of success.

Problem 1: complexity

Some business problems facing your IT organization are consistently the same. Corporate management pushes for better utilization of IT resources, greater return on investment (ROI), integration of historically separate systems and faster implementation of new systems. But some things are different now. Environments are more complex. Budget constraints and operating efficiencies require you to reuse legacy systems rather than replace them. Inexpensive, ubiquitous access to the Internet has created the possibility of entire new business models that you have to evaluate to keep pace with your competitors. Growth by merger and acquisition has become standard fare, so entire IT organizations, applications, and infrastructures must be integrated and

According to Aberdeen Group, surveys of Global 2000 CIOs consistently identify the cost, complexity and integration time of enterprise application integration (EAI) and business-to-business (B2B) integration as one of their top concerns. Even with tightening budgets and lower profit margins, the business benefits of a solid integration strategy are so compelling that CIOs predict they'll spend between 35 percent and 60 percent of their budgets on integration projects.[1]

absorbed. In an environment of this complexity, point-to-point solutions merely exacerbate the problem, and will never really meet the challenge. You must develop systems that incorporate heterogeneity as a fundamental part of your IT environment, so they can accommodate an endless variety of hardware, operating systems, middleware, languages and data stores. The cumulative effect of decades of growth and evolution has produced the complexity you're now dealing with. With all these business challenges for IT, it is no wonder that application integration tops the priority list of many CIOs.

Problem 2: redundant and nonreusable programming

Like many companies, your application portfolios may have grown as a result of mergers and acquisitions. As a result, you may be dealing with redundant applications – or applications with function that can't easily be reused. Perhaps each business unit within your organization has acted separate from every other unit, effectively hindering any coordinated effort to create reusable functional assets or services. Collectively this redundancy increases both cost and time to market to deploy new products or services, because changes have to be made in each application or system affected. This lack of reuse ultimately requires more resources – and often more time – to deliver new applications.

Problem 3: multiple interfaces

Consider 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, the process will produce *n*(*n-1*) connections, or interfaces. In Figure 1, each arrowhead represents an interface.

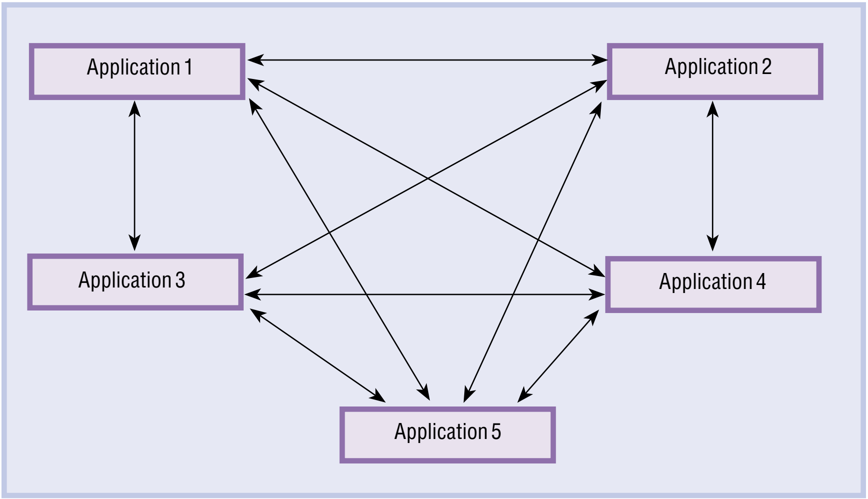

Figure 1. The n(n-1) integration problem

Consequently, if you must integrate another application system $A_{(n+1)}$, you will need to generate, document, test and maintain *2n* new interfaces. In Figure 1, the set of five applications requires 20 direct interfaces. Adding a sixth application would require ten new interfaces. And to further increase complexity, you must modify the code in each of the existing applications to include the new interfaces, generating substantial testing costs. To reduce this cost and complexity, you need an optimum solution that produces the minimum number of interfaces *n* for *n* applications, with only one new interface for each system added. However, it can't be done by direct connection.

What about 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 architects to assemble applications through parts, components or services. More recently, Java™ technology has provided platform-neutral programming. XML has provided self-describing, platform-neutral data. Now Web services have removed another barrier by allowing applications to interconnect in an object-model-neutral way. For example, using a simple XML-based messaging scheme, Java applications can invoke Microsoft® .NET applications or CORBA-compliant, or even COBOL, applications. So, IBM CICS® or IBM IMS™ transactions on a mainframe in Singapore can be invoked by a .NET application which in turn may be invoked by an agent running on an IBM Lotus® Domino® server in Munich. Best of all, the invoking application doesn't have to know where the transaction will run, what language it is written in or 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 their predecessors. The timely convergence of several necessary technological and cultural prerequisites have contributed to this adoption, including:

- *A ubiquitous, open-standards-based, low-cost network infrastructure, and technologies that offer a distributed environment much more conducive to the adoption of Web services than both CORBA and Distributed Computing Environment (DCE)-faced environments*
- *A degree of acceptance and technological maturity to operate within a network-centric environment that requires interoperability 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 (such as TCP/IP); tool sets (integrated development environments [IDEs] and Unified Modeling Language [UML]) platforms (such as Java 2 Platform, Enterprise Edition [J2EE]) and related methodologies (such as object-oriented [OO] technology and services), 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.*

SOA can be both an architecture and a programming model, a way of thinking about building software. An SOA enables you to design software systems that provide services to other applications through published and discoverable interfaces, and where the services can be invoked over a network. When you implement an SOA using Web services technologies, you create a new way of building applications within a more powerful, flexible programming model. You can reduce your development and ownership costs – and your implementation risk.

On the horizon, however, are even more significant opportunities. First, grid computing, which is much more than just the application of millions of instructions per second (MIPS) to effect a computing solution. Grid computing will also provide a framework that will enable you to dynamically locate, relocate, balance and manage massive numbers of services so you can guarantee that needed applications are always securely available, regardless of the load placed on your system.

This framework, in turn, gives rise to the concept of on demand computing, which could be implemented on any configuration, from a simple cluster of servers to a network of 1024-node IBM SP2™ systems. If a user needs to solve a problem and wants the appropriate computing resources applied to it – no more, no less – you can pay only 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. These circumstances, along with the problems previously discussed, mean that your infrastructure must undergo a fundamental change – the conversion to an SOA.

Requirements for an SOA

From the problems previously discussed in this paper, it should be clear that it's important to develop an architecture that meets all of your requirements. These requirements should include the ability to:

- Leverage existing assets.
  This is your most important requirement. Existing systems can rarely be thrown away, and probably contain within them data that is of great value to your enterprise. Strategically, the objective is to build a new architecture that will yield all the value that you hope for, but tactically, your existing systems must be integrated so that, over time, they can be componentized or replaced in manageable, incremental projects.

- Support all required types of integration.
  These include user interaction (to provide a single, interactive user experience), application connectivity (to deliver a communications layer that underlies the entire architecture), process integration (to choreograph applications and services), information integration (to federate and move your enterprise data) and build to integrate (to build and deploy new applications and services).

- Allow for incremental implementations and migration of assets.
  Fulfilling this requirement will enable one of the most critical aspects of developing the architecture: the ability to produce incremental ROI. Countless integration projects have failed because of their complexity, cost and unworkable implementation schedules.

- Build around a standard component framework.
  You must include a development environment that is built around a standard component framework to 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.

- Allow implementation of new computing models.
  Specific examples of this requirement include new, portal-based client models, grid computing and on demand computing.

An SOA—not just Web services

The advent of Web services has precipitated a fundamental change in how IT infrastructures can be developed, deployed and managed. The success of many Web services projects has shown that technology does exist that can enable you to implement a true SOA. It allows you to take another step back and examine your application architecture – as well as the basic business problems you're trying to solve. From a business perspective, it's no longer just a technology problem, it's a matter of developing an application architecture and framework within which you can define business problems and implement solutions in a coherent, repeatable way.

First, though, it's important to understand that Web services does not equal SOA. Web services is a collection of technologies, including XML, Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL) and Universal Description, Discover and Integration (UDDI), which allow you to build programming solutions for specific messaging and application integration problems. Over time, these technologies can be expected to mature, and eventually be replaced with better, more-efficient, more-robust technology. But for the moment, the existing technologies are sufficient, and have already proven that you can implement an SOA today.

What actually constitutes an SOA? An SOA is exactly what its name implies – an architecture. It's more than any particular set of technologies, such as Web services. It transcends these technologies – and, in a perfect world, is totally independent of them. Within a business environment, a pure architectural definition of an SOA might be *an application architecture within which all functions are defined as independent services with well-defined invokable interfaces, which can be called in defined sequences to form business processes.* Note the components of this definition:

- *All functions are defined as services. This includes purely business functions (such as create a mortgage application or create an order), business transactions composed of lower-level functions (such as get credit report or verify employment) and system service functions (such as validate identification or obtain user profile). This brings up the question of granularity, which will be addressed later.*
- *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.*
- *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). It doesn't matter 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 an SOA, the interface is the key, and it is the focus of the calling application. It defines the required parameters and the nature of the result. This means that it defines the nature of the service, not the technology used to implement it. The system must effect and manage the invocation of the service, not the calling application. This function 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:

- *Security, to authorize requests, encrypt and decrypt data as required, and validate information.*
- *Deployment, to allow the service to be moved around the network to maximize performance or eliminate redundancy to provide optimum availability.*
- *Logging, to provide auditing and metering capabilities.*
- *Dynamic rerouting, to provide fail-over or load-balancing capabilities.*
- *Maintenance, to manage new versions of the service.*

The nature of a service

What is a service? As previously stated, typically within a business environment, a service can be a simple business capability (such as getStockQuote, getCustomerAddress or checkCreditRating), a more complex business transaction (such as commitInventory, sellCoveredOption or scheduleDelivery) or a system service (such as logMessageIn, authenticateUser). Business functions are, from the application's perspective, nonsystem 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 individual 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, such as Microsoft Windows® or Linux.

This may seem like an artificial distinction of the services. You could assert that from the application's perspective, all services are atomic; it's 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 practical implications. Services may be low-level or complex high-level (fine-grained or course-grained) functions, and there are very real tradeoffs in performance, flexibility, maintainability and reuse, based on their definitions. The level of granularity is a statement of a service's functional richness. For example, the more coarse-grained a service is, the richer the function offered by the service. Services are typically coarse-grained business functions, such as openAccount, because this operation might result in the execution of multiple finer-grained operations, such as verifyCustomerIdentity and createCustomerAccount. 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 be used to build new applications, or integrate existing software assets.

There are many such frameworks available today; within IBM, several frameworks, such as Enterprise Workframe Architecture (EWA), JADE[2] and Struts (from Jakarta), are being used in client-integration scenarios. Taking EWA, from the IBM Software Group Advanced Technology Solutions team, for example, at a very high level, the framework looks like Figure 2. Within this framework, a configuration defines an application. It also describes the components of the application, as well as the sequence and method of its invocation. Input is received and passed to the application in a source-neutral way. So, for instance, adding 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 features, and special-purpose access components enable connection to back-end enterprise applications, so that they can remain in place, or be migrated over time. While EWA is fully J2EE technology-compliant, it can connect to external DCOM or CORBA component-based systems.

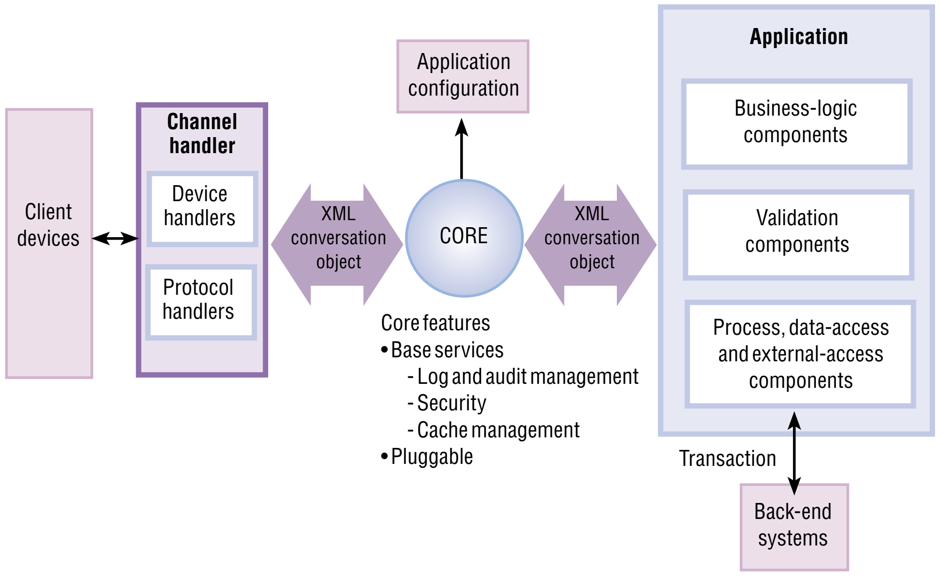

Figure 2. The EWA framework

Today, the EWA framework contains over 1,500 general and special-purpose components, greatly reducing the amount of code you have to write to create a new application.

Addressing the old problems

Returning now to the first integration scenario discussed, how do you find a scheme that minimizes the number of required interfaces, such as is drawn in Figure 3?

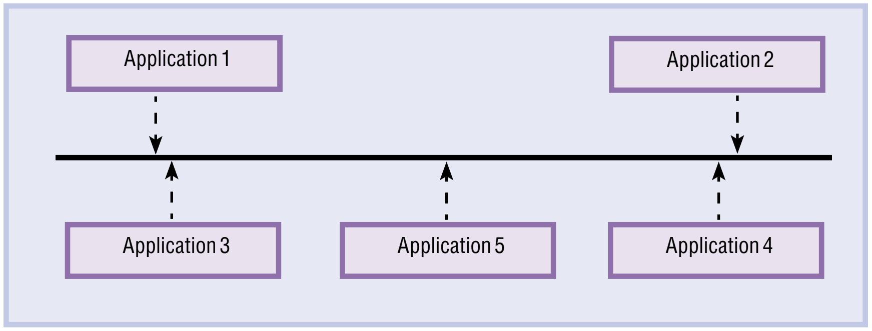

Figure 3. Application integration

Figure 3 may look like an overly simplistic view, but it illustrates how, within a framework such as EWA, this view is the starting point. Now you can add the architectural concept of the service bus, represented in Figure 4 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.

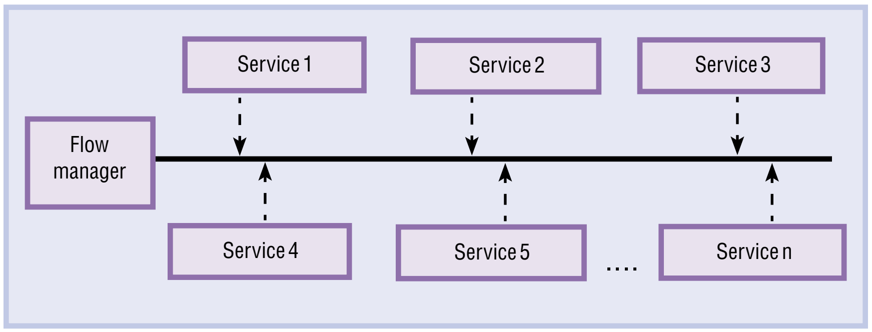

Figure 4. Service integration

At this point, you need to determine how to call the services, so you add application configuration. Then, virtualize the inputs and outputs. Finally, you provide connectivity to back-end processes. The result is a comprehensive framework that allows processes to run as-is, and provides for their future migration. Now, as Figure 5 shows, the high-level picture is at least structurally complete.

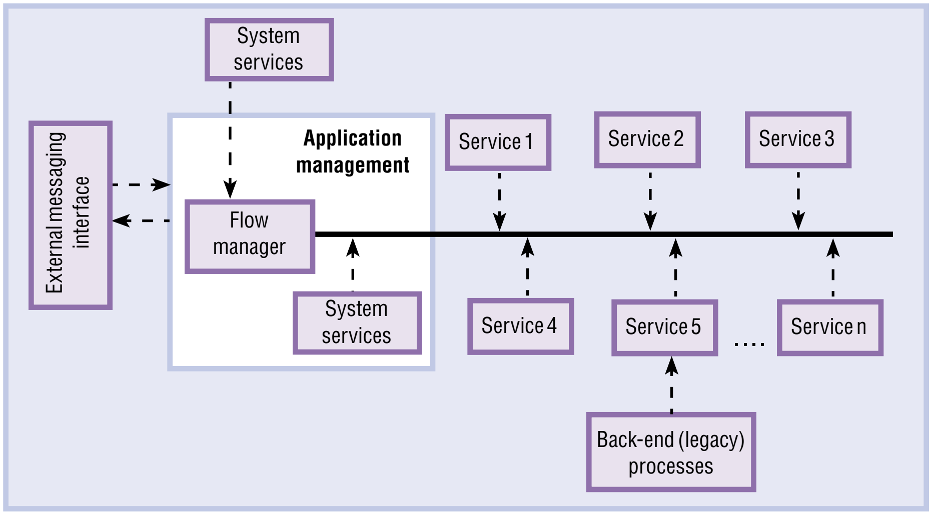

Figure 5. The completed framework

You shouldn't be surprised that Figure 5 bears some resemblance to a block diagram of EWA. At the highest level, any robust application framework must provide these functions. Now, however, the real work begins: building the 1,500 components that put flesh on this skeleton. 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; thus, many IT architects choose to implement within an existing framework. Regardless of how you approach it, you can implement this architecture using technologies and frameworks that exist today—which brings you back to the beginning, to an analysis of the business problems that you need to solve. You can address these problems with the confidence that your architecture will be implementable.

Integration requirements within the architecture
So far in this white paper, the discussion of integration has been confined to application integration through component-based services, but integration is a much broader topic than this. When assessing the requirements for an architecture, you must consider several integration types. You 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 through 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 through 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 issues 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 in the channel and protocol handlers in Figure 2. Here the problem is the fundamental way data moves in and out, 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 application processes, and integrating processes with other processes. The first requirement may seem obvious; that is, that the architecture should allow for an environment within which the basic business problems can be modeled. However, insufficient analysis at this level can present significant challenges for any implementation of the architecture, regardless of its technical sophistication. Integration of applications as part of 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 (SCM) or financial services that span multiple institutions. For such application and process integration needs, you can use technologies such as BPEL for Web services (BPEL4WS). Or the application framework may use a program-configuration scheme, such as the one seen in EWA. A higher-level configuration scheme can be constructed using BPEL4WS at a lower level, and then driven by an engine that provides more function than just flow management. Before any of this is built, however, the architectural requirements must be understood first, so you can build the appropriate infrastructure.

Information integration is the process of providing consistent access to all the data in your 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 only adapter software and a transformation engine; however, typically, the process is more complex. Often the key concept is the virtualization of the data, which may involve the development of a data bus from which data can be requested using standard services or interfaces by all applications within your enterprise. So the data can be presented to the application regardless of whether it came from a spreadsheet, a native file, a structure query language (SQL) or other 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 unaware of the operating system that manages the data, so native files on an IBM AIX® or Linux system are accessed the same way they would be on Windows, IBM z/OS® or virtually any other system. The location of the data is also transparent; because it is provided by a common service, it is the responsibility of the access service, not the application, to retrieve the data, locally or remotely, and then present the data in the requested format.

Last, one of the requirements for the application-development environment must be that it takes into account all the styles and levels of integration that can be implemented within your 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, all styles must be considered when embarking on a new architectural framework. Your IT environment may have only a small number of data source types, so information integration may be straightforward. Or 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 an SOA

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

- Leveraging existing assets
  This benefit is the first, and most important, of the requirements discussed earlier in this paper. You can construct a business service as an aggregation of existing components, using a suitable SOA framework and made available to your enterprise. Using this new service requires knowing only its interface and name. The service's implementation specifics (such as its component architecture) or discrete functional components—as well as the complexities of the data flow through the components that make up the service—are transparent to callers. 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 using Web services interfaces. More important, legacy systems can be transformed, adding value as their functionality is transformed into services.

- Infrastructure as a commodity
  Infrastructure development and deployment will become more consistent across all your different enterprise applications. Existing components, newly developed components and components purchased from a range of vendors can be consolidated within a well-defined SOA framework. Such an aggregation of components will be deployed as services on the existing infrastructure. As a result, the underlying infrastructure becomes more of a commodity. Over time, as services become more loosely coupled from the supporting hardware, you can optimize the hardware because the service assembler is no longer dependent upon the hardware environment on which the service operates at run time.

- Faster time-to-market
  Organizational Web services libraries will become your organization's core assets as part of your SOA framework. Building and deploying services with these Web services libraries will reduce your time to market dramatically, as new initiatives reuse existing services and components, reducing design, development, testing and deployment time in the process. As services reach critical mass in your organization or trusted network, the larger ecosystem emerges – enabling you to assemble composite applications using services, rather than developing custom applications.

- Reduced cost
  As business demands evolve and new requirements are introduced, the cost to enhance and create 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. You will also reduce the risk in the maintenance and management of the infrastructure supporting the services.

- 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 – and provides 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 function allows you to change the process flows while monitoring the effects to facilitate 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 among 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 subapplication. These subapplications are chained together to create a process flow capable of satisfying the business need. This granularity lets processes leverage and reuse each subapplication throughout your organization.

The future: new models, new requirements

So far, this white paper discussion centers around the need to increase speed of business changes, and to improve business performance and efficiency. These requirements mandate a set of IT imperatives for flexibility where SOA becomes a key enabler. But what if a completely new model for application development emerges? Will the notion of an SOA still be meaningful or required? The answer is a resounding, yes. Two new, emerging concepts are 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 makes the evolution to SOA even more imperative. Representing every application, resource or business capability as a service with a standardized interface allows you to quickly combine new and existing applications to address changing business needs and improve operational effectiveness – the essence of SOA. As a result, SOA becomes the DNA of grid computing and on demand computing.

Grid computing

An in-depth discussion of grid computing is beyond the scope of this paper, but a couple of points are worth mentioning. First, grid computing is much more than just the application of large numbers of millions of instructions per second (MIPS) to effect a computing solution to a complex problem. It enables you to divide resources into multiple execution environments by applying one or more concepts, such as hardware or software partitioning, or time-sharing, machine simulation, emulation and quality of service. This virtualization, or on demand deployment, of all your distributed computing resources lets you use them wherever and however they are needed within the grid. Virtualization is simply a form of resource management for devices, storage, applications, services or data objects. Hence, applying SOA allows you to maximize resource utilization in a grid environment. You can deploy and migrate a services ecosystem onto appropriate nodes in a grid environment to respond efficiently to changes in your internal and external business environment.

On demand computing

An in-depth discussion of on demand computing is also beyond the scope of this paper. But, again, SOA can be an essential prerequisite for on demand computing. SOA is an enabling architecture for on demand applications. Thus, applications must operate in an SOA to realize the benefits of on demand. Web services is an enabling technology for SOA. As a subset of on demand computing, Web services on demand is simply business services exposed using Web services standards.

On demand computing can cover a wide spectrum. One end of this spectrum focuses on the application environment; the other end focuses on the operating environment, which includes items like infrastructure and autonomic computing. Transforming your business means leveraging both the application and operating environments to create an on demand business. At the heart of your on demand business will be business 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 could even transform the way business is conducted and the way you offer your products and services over the Web to your entire value chain. 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 – and you would have become an on demand business. Web Services and SOA can make this possible for applications.

Summary

SOA is the next wave of application development. Web services and SOA are about designing and building systems using heterogeneous network-addressable software components. SOA is an architecture with special properties, comprising 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 your organization's existing investment by taking advantage of current resources – including developers, software languages, hardware platforms, databases and applications – and can help reduce costs and risks while boosting productivity. This adaptable, flexible 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.

For more information

To learn more about SOAs and how IBM can help you build an SOA for your enterprise, visit:

ibm.com/software/info/openenvironment/soa/
or
ibm.com/services

[1] Tom Dwyer, Using Composite Applications to Lower Integration Costs, Aberdeen Group, (April 2003).

[2] Jade provides the base application infrastructure for a JSP/Servlet application. This infrastructure consists of a programming model based on hundreds of engagements and IBM best practices and a set of Java utilities and proven practices for building Web applications.

e business on demand™

G224-7298-00