



Search for: _____ within _____

[Search help](#)

[IBM home](#) | [Products & services](#) | [Support & downloads](#) | [My account](#)

developerWorks > Web services

developerWorks



Migrating to a service-oriented architecture, Part 1

Introduction and overview

Level: Intermediate

Average rating:  (29 ratings)

[Kishore Channabasavaiah](#), Executive Architect, IBM Global Services
[Kerrie Holley](#), Distinguished Engineer, Chief Architect - eBusiness Integration Solutions, IBM Global Services
[Edward M. Tuggle, Jr.](#), Senior Software Engineer, IBM Software Group

16 December 2003

This is the first in a series of papers intended to help you better understand the value of a service-oriented architecture (SOA), and to develop a realistic plan for evaluating your current infrastructure and migrating it to a true service-oriented architecture. It is intended that after reading this paper, you will understand why it is claimed that a SOA is the best platform for carrying existing assets into the future, as well as enabling the rapid and correct development of future applications. Additionally, you should have a better understanding of the major considerations in planning such a migration.

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, greatly reducing 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, you will find that a SOA, at least for now, is the best foundation upon which an IT organization can take its existing assets into the future as well as build its new application systems. This is the first in a series of papers intended to help you better understand the value of a SOA, and to develop a realistic plan for evaluating your 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

Contents:

[The case for developing a service-oriented architecture](#)
[Requirements for a service-oriented architecture](#)
[Summary](#)
[Resources](#)
[About the authors](#)
[Rate this article](#)

Related content:

[Web Services Architectures and Best Practices](#)

Subscriptions:

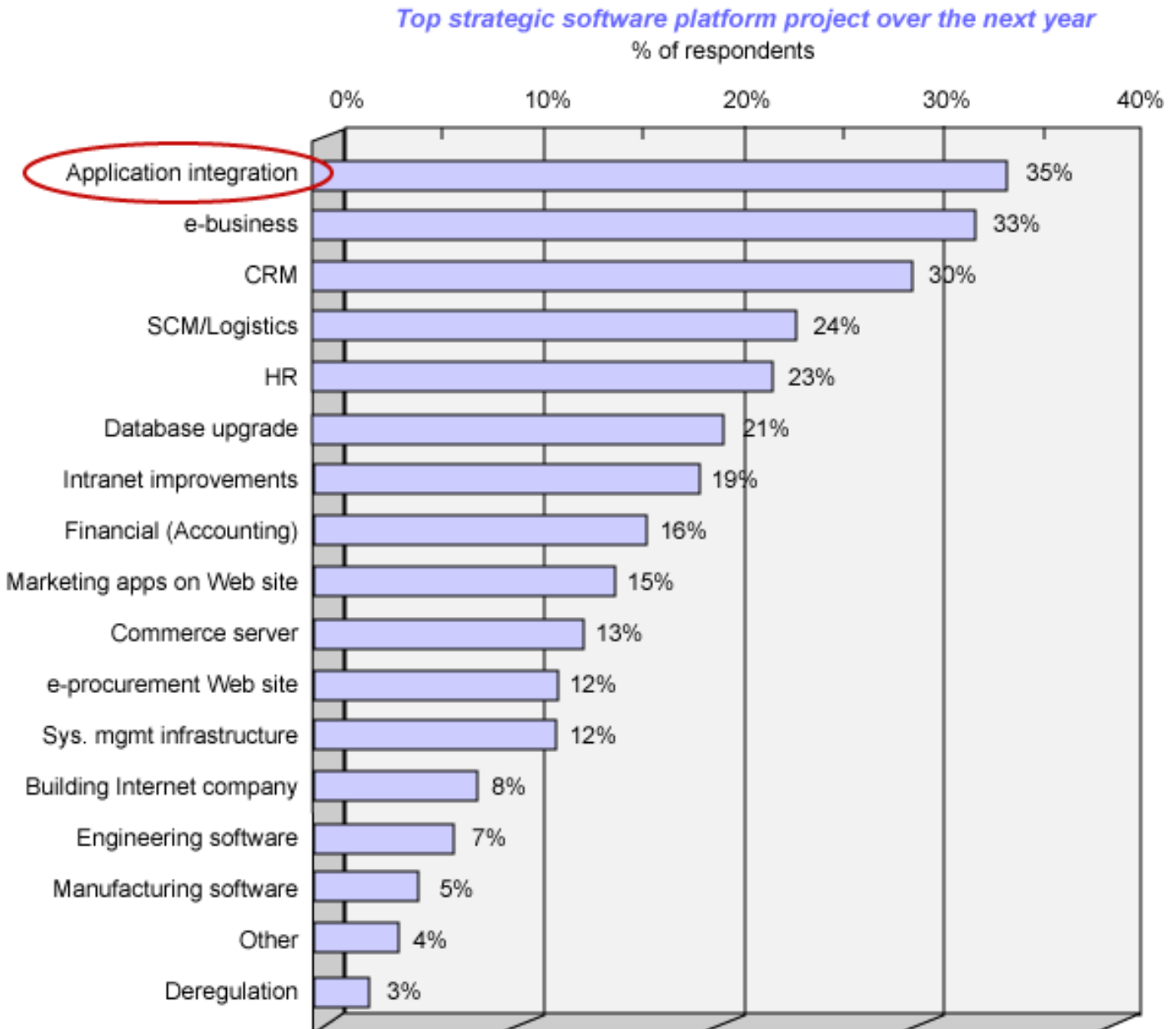
[dW newsletters](#)
[dW Subscription \(CDs and downloads\)](#)

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.

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 you find more complex environments. Legacy systems must be reused rather than replaced, because with even more constrained budgets, replacement is cost-prohibitive. You find that cheap, ubiquitous access to the Internet has created the possibility of entirely new business models, which must at least be evaluated since the 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 data stores. The cumulative effect of decades of growth and evolution has produced severe complexity. With all these business challenges for IT, it is no wonder that application integration tops the priority list of many CIOs, as shown in [Figure 1](#).

Figure 1. CIO priorities



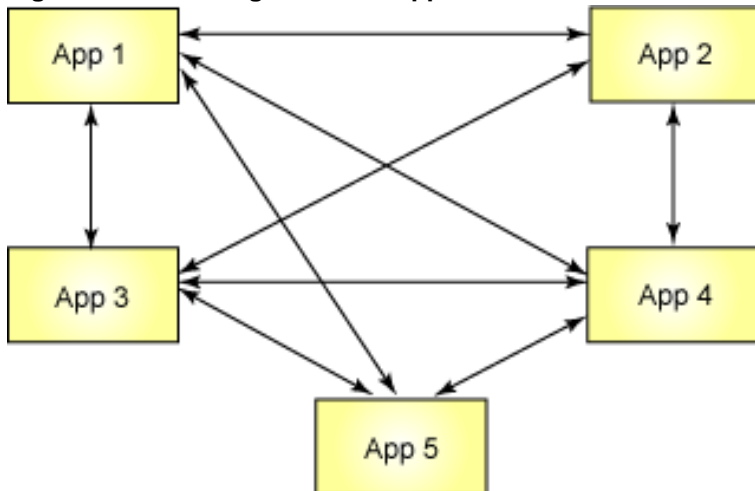
Another problem - redundant and non-reusable programming

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 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 [Figure 2](#), each arrowhead represents an interface.

Figure 2. Direct integration of n applications



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, you 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 technology contributed platform-neutral programming, and XML contributed 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 most likely has no idea 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 more genuine 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 (such as TCP/IP), tool sets (IDE's, UML, etc.), platforms (such as J2EE platforms), and related methodologies (such as 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 you implement a service-oriented architecture using Web services technologies, you create a new way of building applications within a more powerful and flexible programming model. Development and ownership costs as well as implementation risks are reduced. 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 -- paying 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. This, along with the problems previously discussed, leads to the conclusion that a fundamental change must be made -- the conversion to a service-oriented architecture.

Requirements for a service-oriented architecture

From the problems discussed above, it should be clear that an architecture should be developed that meets all 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 hoped for, but tactically, the existing systems must be integrated such that, over time, they can be componentized 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 you can implement a true service-oriented architecture. It lets you take another step back and not just examine your application architecture, but the basic business problems you 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.

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 let you build programming solutions for specific messaging and application integration problems. Over time, you 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 can finally 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 a 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 invocable; 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:

1. Security -- authorization of the request, encryption and decryption as required, validation, etc.
2. Deployment -- allowing the service to be redeployed (moved) around the network for performance, redundancy for availability, or other reasons
3. Logging -- for auditing, metering, etc.
4. Dynamic rerouting -- for fail over or load balancing
5. Maintenance -- management of new versions of the service

Summary

In this first part, you have briefly examined some of the problems that lead to consideration of a SOA, and the requirements then placed on the new architecture. Part 2 will examine the nature of a service, constructing an application framework of service-based components, and some of the future computing environments that will make the development of a SOA even more imperative.

Resources

- See IBM developerWorks [Web services zone](#) for Web services whitepapers and tools.
- For customer project case studies on Web services and service-oriented architecture, see the [IBM jStart Web site](#).
- Read the article on [Best practices for determining the proper level of granularity of services within a SOA](#) (*developerWorks*, October 2003).
- Read about [Accessing CICS transactions as services within a SOA](#) (IBM).
- The [Zackman Framework for Enterprise Architecture](#) will be the subject of another paper in this series on SOA. Out of the Zachman Framework has developed several major architectural frameworks, including:

- o The Federal Enterprise Architecture Framework (FEAF)
- o A framework, for Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance (C4ISR) architecture
- o The Open Group Architecture Framework (TOGAF)

About the authors

Kishore Channabasavaiah received a Bachelors degree in Mechanical Engineering from Bangalore University, India. He is currently an Executive Architect in the Chicago Innovation Center of IBM Global Services. He provides thought leadership for e-business Integration solutions with a 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. You can contact Kishore at [kishorec at us.ibm.com](mailto:kishorec@us.ibm.com).

Kerrie Holley received a Bachelor of Arts degree in Mathematics 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 e-business 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. You can contact Kerrie at [kholley at us.ibm.com](mailto:kholley@us.ibm.com).

Edward M. Tuggle, Jr. received a Bachelor of Science degree in Mathematics from the University of Oklahoma, and is currently a Senior Software Engineer on the IBM Software Group jStart Emerging Technology Solutions team. He worked with IBM in operating systems design, development, and maintenance for 23 years, for the past 6 years in Java technology and other emerging technologies, and is now specializing in Web services and service-oriented architecture. You can contact Edward at [b391747 at us.ibm.com](mailto:b391747@us.ibm.com).



What do you think of this document?

Killer! (5)

Good stuff (4)

So-so; not bad (3)

Needs work (2)

Lame! (1)

Comments?